AUTOMATED REPORT GENERATION FROM AGRICULTURAL ROBOT LOG DATA

Faris Elghlan
Timo van Leest
Michiel van den Berg

Client: J. Jacobs, Orchid N.V
Coach: D.Tax, Pattern Recognition and Bioinformatics group
Project coordinator: O. Visser
Technical University of Delft, July 14, 2017

Preface

Before you lies the final report of the bachelor thesis of Michiel van den Berg, Faris Elghlan and Timo van Leest. This bachelor thesis is the final part of the bachelor Computer Science at the Technical University of Delft. This report describes in detail the project that was conducted in ten weeks time and started April 2017. The project was conducted on behalf of Orchid N.V. as a bachelor graduation project. During this project we showed that we are capable to successfully execute a complete software development cycle.

We would like to thank our supervisor, David Tax, at the Technical University of Delft, for his assistance during the project and his tireless enthusiasm. In addition, we would like to thank Jan Jacobs at Orchid N.V. who introduced us to the company and gave a lot of suggestions and advice during the course of the project.

Faris Elghlan
Timo van Leest
Michiel van den Berg
Maasluis, July 14, 2017

Summary

This project was performed for Orchid N.V., an innovator in agriculture. The project focuses on their Scalar system. This is a system that consists out of multiple machines that automatically feed cows. While farmers are generally pleased with these systems, [redacted].

A daily report about the performance of the systems on the farm was the initial assignment for the project. The project started with a two week research phase in which the assignment was further investigated by visiting test farms and conducting interviews with farmers and Orchid employees. [redacted] farmers had to be informed about potential problems on their farm. This can be done by looking at patterns and outliers. [redacted].

After the research phase, three development cycles were executed. At the end of each design cycle, a new version of the product was reviewed within Orchid and at a test farm. The final product consists out of a cloud solution that processes logs to generate data and a web application dashboard, which visualizes this data. The web application dashboard shows an overview of the performance of the Scalar system on the farm and a list of points of attention, with interesting findings within the data. These could be potential problems. The reactions from Orchid and the farm were very positive. During the process there were not many problems and the project can be considered a success. [redacted]. The project was finalized with a final presentation at Orchid and a final presentation at Technical University of Delft.

Contents

1 Introduction

Orchid is a Dutch company founded in 1948 and currently has around 2000 employees. The company is an innovator in agriculture and develops robot systems for the cattle industry. These robot systems automate processes on farms that were previously performed by hand. This include machines for baling, robotic milking, barn hygiene, mowing and tedding. Farmers in more than 60 countries are using these machines. A large part of Orchid's operations includes patents on all kind of inventions.

One of their more recent systems involves the automatic feeding of cows and is called the Scalar. The Scalar system has been in production for five years. The only tasks the farmer will have to perform himself are filling the feed storage places and possibly resolving alarms or mechanical issues. The Scalar system not only greatly reduces the amount of labor required on a farm, but also helps with feeding efficiency, accuracy and the health of the cattle. The Scalar system consists out of multiple machines working together. There is the Feed grabber (FG) which grabs the different types of food and loads them into the Mixing and feeding robot (MFR). The MFR mixes the different food types, after which it drives towards the fences with cows at which it dispenses the food. This system can have up to two MFRs, enabling the system to feed hundreds of cows a total of over 10,000 kilograms of food a day. Apart from the Feed grabber (FG) and MFR, an additionally dispensing system is often used to pour smaller amounts of feed into the MFR. Finally there is also the Power distribution box (PDB), which is the unit that controls the dispenser and decides which actions may be performed, such that, for example, two MFRs do not drive into each other. The MFR and FG are shown in figure 1.

[redacted]

The results of this project will be described in this report. Section 2 will start with a problem description in which the problem will be further explained. This will be done with the results of interviews, that were conducted with farmers and employees of Orchid, at the start of the project. Section 3 will look at existing different solutions which can potentially be used for our product and explain some of the choices that were made. Section 4 will show the process of this project, how the ten weeks were planned into iterations and some of the earlier designs of the product, which gives some insight into the process. In section 5 the final product will be described, including some images showing the system. In section 6 the implementation of the system and the technical details will be discussed. Section 10 will conclude the report, summarizing the results and reflecting back on the project. Finally, section 11 will give some recommendations for future improvements of the system.

Figure 1: The MFR (left) and FG (right)

2 Problem description

This section will give a description of the problem this project tries to solve. The description is as we defined it during the research phase of the project, so there are a few differences with the final product. This will be reflected upon during later sections and the conclusion.

At the start of the project, the requirements of the final product had not been specified yet. There was of course the project description which can be found in Appendix A, but this description was not very detailed, nor precise. Therefore the project started with examining the needs of the client and determining requirements for the product. Notice that the requirements at the end of this chapter are the initial requirements, thus they are the same as the requirements of the research report in section B.3.

The project focuses on the Scalar system. The Scalar system consists out of multiple machines working together. There is the Feed grabber (FG) which grabs the different types of food and loads them into the Mixing and feeding robot (MFR). The MFR mixes the different feed types, after which it drives towards the fences with cows at which it dispenses the food. [redacted]. This has to be done by automatically detecting and classifying the problems and notifying the farmer of the issue as soon as possible. [redacted]. An example image of an overview was provided by the client, which can be found in figure 2.

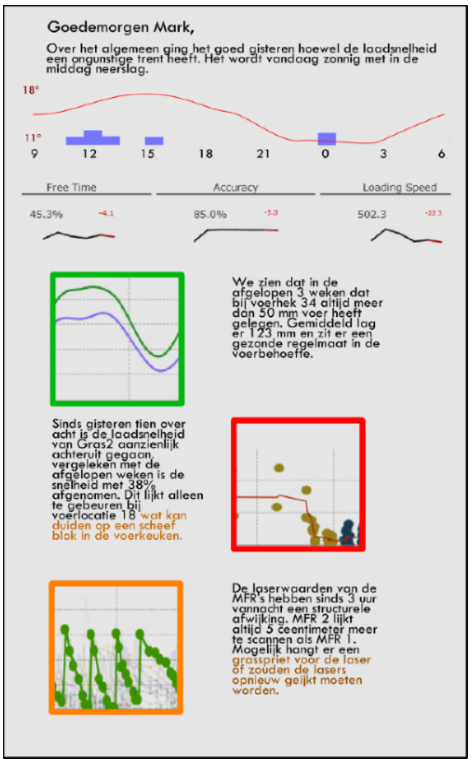


Figure 2: A draft design of a potential overview

To gain a better understanding of the problem, interviews were conducted with the some farmers and Orchid employees who often interact with farmers. Multiple visits to the actual farms were done as well, to gain more insight in the operations of the Scalar system.

The reports of each interview are given in Appendix C. [redacted].

[redacted].

- [redacted]
- [redacted]
  - [redacted]
  - [redacted]
  - [redacted]
  - [redacted]
- [redacted].
- [redacted]
- [redacted]

[redacted].

- [redacted]

  [redacted].

- [redacted]

  [redacted].

[redacted].

1. [redacted]
2. [redacted]
3. [redacted]

2.1 Initial requirements

This section will give the initial requirements which have been decided for the project, using the MoSCoW method. These requirements have been decided upon based on the problem description which was given earlier in this chapter, the interviews in Appendix C and in collaboration with the client. While these were the initial requirements, not all of them have been implemented for various reasons and some of them have changed. In the next section these changes will be talked about while the final product will be described.

2.1.1 Functional requirements

This section lists the functional requirements. Functional requirements describe the behavior and functions of the system.

Must haves

1. [redacted].
2. [redacted].
3. [redacted].

Should haves

4. [redacted].
5. [redacted].
6. [redacted].
7. [redacted].
8. [redacted].

Could haves

9. [redacted].
10. [redacted].
11. [redacted].

Won't haves

12. [redacted].

2.1.2 Non-functional requirements

This section lists the non-functional requirements. Non-functional requirements impose constraints on the design or implementation of the system.

Must have

i. The software must be kept private.

ii. All code must have documentation so it is clear what the purpose of each part of the code is.

iii. The software must be developed following the agile methodologies to allow for software adjustments if the user's requirements change.

iv. The developed software must be able to run in the Microsoft Azure[1] cloud, since Orchid already uses this.

Should haves

v. [redacted].

vi. [redacted].

vii. [redacted].

viii. [redacted].

[1] https://azure.microsoft.com/

3 Existing solutions

To limit the amount of code we have to write ourselves, we have looked into existing software solutions which can be reused. This section lists all of our findings.

3.1 Microsoft Azure

As stated in requirement iv., Microsoft Azure must be used to run the application on. Azure is a cloud platform, offering a lot of cloud services. These cloud services including: data storage, computation units, data analytics, networking, web applications, security and much more. All of these cloud services can be managed through the Azure portal, which can be seen in figure 3. In the center area the cloud services which we are using can be seen and on the left side new services can be added.

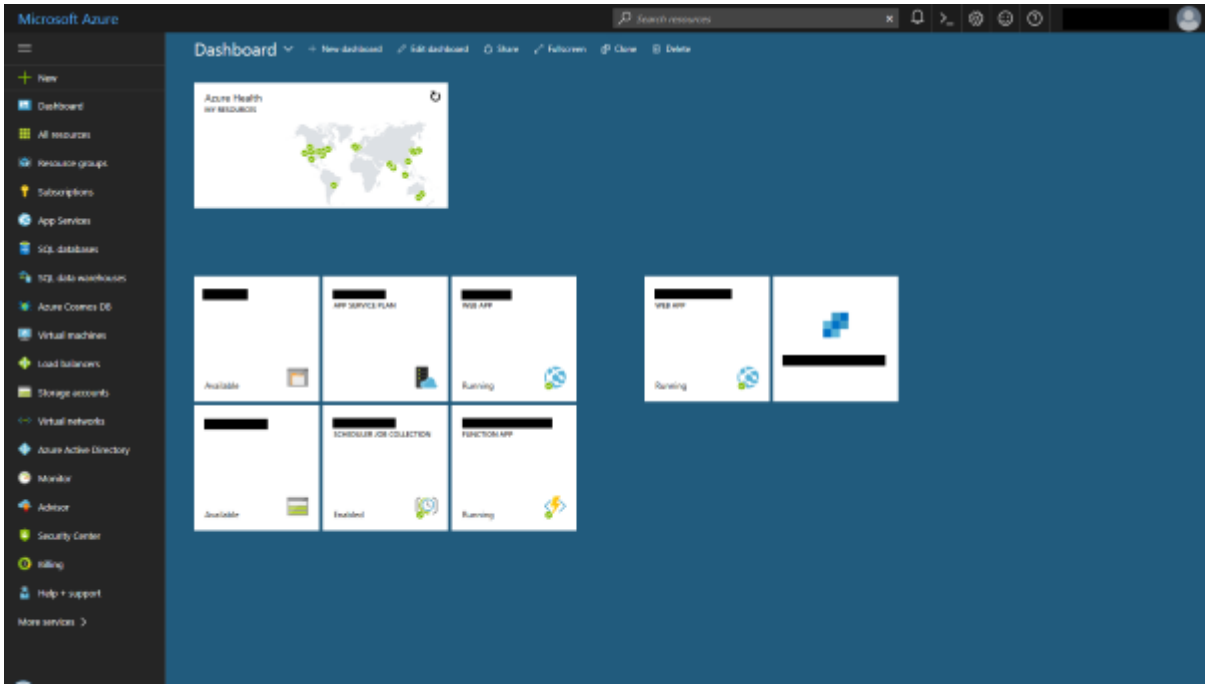


Figure 3: Microsoft Azure portal

Azure Machine Learning, as was discussed in the research report in section B.5.2, is also available through the portal. We did not use Azure Machine Learning in the final product, but this might still be interesting when the project is continued.

3.2 Pattern recognition

In sections B.5.1 and B.5.2 of the research report, we have discussed general Pattern recognition methods and the libraries which provide these methods. We did not end up using either of these libraries, as our use cases are simple and we prefer to have more control and knowledge of the exact implementation which was used. From the discussed novelty detection methods, probabilistic novelty detection has been used to find outliers and a much similar method has been used to determine if the data has changed, which we refer to as a level change. This can for example occur when a setting of the Scalar system is changed, resulting in a value being consistently higher or lower from that moment onward. More details about the implementation will be discussed in section 6.6.

3.3 Chartjs

Chartjs is a javascript charting library which can be used for data representation. We have used this library to generate our charts during this project. Chartjs is an open source project with a lot of documentation, it is also one of the most used charting libraries, so a lot of examples are available. Alternatively ASP.NET has a standard charting solution as well, but this solution is not very flexible and does not allow a lot of customization, making it unsuitable for our project. There are also a lot of other charting libraries, but these are either not free to use or not as flexible as chartjs. Another solution would have been to write our own library for chart representation, this would have given the most flexibility, but since there was only a limited amount of time for the project, this was not an option. Some examples of charts made with this library can be found in section 5.

3.4 Bootstrap

Bootstrap is a html, css and javascript framework that makes it very easy to create responsive websites. Responsive means that the website scales to the device of the user, providing the optimal experience for all different kinds of users. Bootstrap saves a lot of work and provides a lot of flexibility for the web design. Without needing to spend a lot of time, the web application works on all devices and scales automatically. If this framework was not available, there probably would not have been enough time to implement this during the project. In section 3.4 the web application is show on different devices.

3.5 Gitlab

Gitlab is a git repository manager used by Orchid. Git is a version control system that enables developers to work together in parallel on source code. It keeps track of changed files and when mistakes are made, going back to an old version is done in a few clicks. Also, being able to review new changes before merging them with the existing code base is one of the nice things about git. Gitlab is used for multiple projects within Orchid, so we have decided to use it as well.

3.6 Orchid's existing solutions

4 Process

In this section, the process of the project will be described. This includes how the ten weeks were planned, the design processes and additional activities which were performed during the project.

4.1 Scrum design cycles

During the project, weekly scrum sprints were used. Scrum is an agile development framework that can be used for managing product development. Having short development periods with weekly review meetings has a lot of advantages. It is clear what everyone is working on at all times and goals and plannings can be adjusted on a weekly basis in case things tend to go wrong, or requirements change. We chose three larger development cycles on top of the weekly sprints, each with an associate product release. By having three releases, we ensured that there were at least three moments to gather feedback.

4.1.1 First two weeks

The first two weeks of the project were spent on research and orientation. The project was more clearly defined during this phase. Farmers and employees of Orchid were interviewed, to find out what the exact requirements of the project should be. These interviews can be found in Appendix C. From all the suggestions, a selection of the most important and most common subjects was made. During the research phase, the different solutions and researched pattern recognition and outlier detection possibilities were also looked into.

Projects were created and initialized, so that the programming could start right after the research phase. New development tools, which could to be used during the project, were learned as well. More details of the research phase is given in Appendix B.

4.1.2 First iteration

During the first iteration, the first version of the report website was build. Some old code that processed logs into the cloud was refactored and additional unit tests for this code were written. An application was developed that processes the data from the cloud into files that could be interpreted by the website. While the original intention was to create an overview on the frontpage with multiple pages behind it with more information, the focus switched to a one-page dashboard design, as this was requested by our client. A design was made for the website layout which was later implemented. The website was developed with ASP.NET, as per the requirements and we used the Model view controller (MVC) pattern to keep the code clean. Also some work was put into pattern recognition, but less than was planned. A first version of the report website was released. Although it was not fully finished yet, it gave a good impression. A meeting was held with our supervisor from the Technical University of Delft and one of the test farms was visited to verify that we were on the right track. Our supervisor gave us some input for the pattern recognition and outlier detection. The farmer was already very surprised with the progress that was made in the first iteration and some feedback was gathered.

4.1.3 Second iteration

The gathered feedback from the first iteration was resolved in the second iteration. The overview website was also finished during this iteration. More work was put into the pattern recognition code. Some graphs of data over longer periods of time were generated, to determine which data was interesting to analyze and which patterns could potentially be found. While these long term analysis were not used in the overview website, they showed to be useful for different departments within Orchid. During the second iteration an upload to Software improvement group (SIG) was also planned, such that they

could evaluate our code. For this test were written and the code was cleaned up. At the end of the second iteration, our supervisor from the Technical University of Delft visited Orchid to get a better impression of the company and the work the project team was doing. Again, useful feedback was gathered and plans were made for the further implementation of the website.

4.1.4 Third iteration

During the third iteration, the outlier detection results were added to the website. Additionally, some effort was made to put the website into an already existing framework of Orchid, such that it could be viewed by the farmers more easily. During this iteration, also some finalizing tasks were done, such as writing the report and giving presentations at Orchid. One of the test farms was visited again to get some final feedback and verify that the product is useful. In order to do this, the farmer has agreed to test the application by using it for a few days and providing feedback of his experience. At the time of writing this report, this is still in progress. Another SIG upload was also prepared during the third iteration, paying extra attention to the comments on the last upload.

4.1.5 Last week

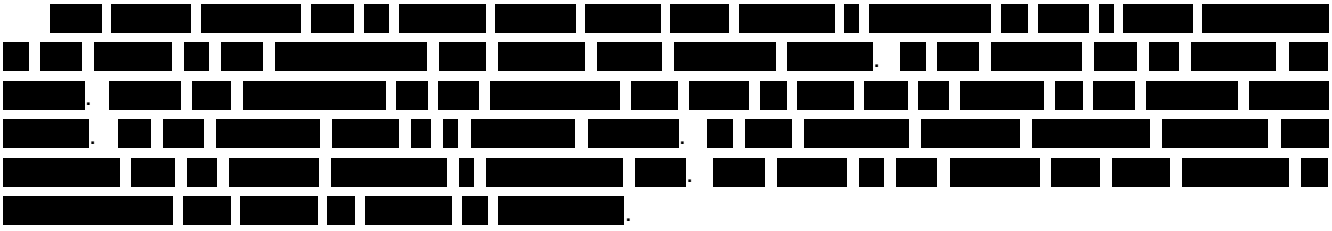
Since this report is handed in before this week, this section will only describe what is planned for the last week. For the last week, a final presentation at Orchid is planned, to show what was done during the project. Various employees will attend this presentation, including managers from different departments. Apart from showing what was done, the presentation will also show what is possible with data, hopefully motivating them to continue the development of these kinds of applications. The final week is also used to prepare the final presentation at Technical University of Delft, and handle some finalizing tasks on the application.

4.2 Website design

In this section, the layout design process throughout the project will be described. During the project, three major designs were made, of which two of them will be described in this section. The final design will be explained in section 5.

4.2.1 Design first iteration

Before the implementation of the website was started, a few initial sketches were made. The purpose of these sketches was to determine the components which were of interest to the client and customer, as well as come up with a layout. During a meeting with the supervisor from Orchid the decision was made to create more of a dashboard design, consisting of only one page. After this the feedback was processed and the final design sketch, as shown in section 4.2.1, was made.



[redacted].

4.2.2 Design second iteration

During the first and second iteration, the design was slightly modified according to the feedback and experience gained during the implementation. [redacted], but the positions of the blocks has changed. Subsubsection 4.2.2 shows an early implementation. The new positions of the blocks, creates a clearer overview and was easier to implement. [redacted].

[redacted]

4.3 SIG feedback

During the writing of this report, only feedback on the first upload to SIG has been received. The second upload can therefore not be elaborated upon. The first upload scored four out of five stars on the maintenance model, which means that the code is rated above average. The code did not score five stars because of minor issues with unit size and unit interfacing. The email with the feedback from SIG has been added to Appendix E.

A lower score for unit size indicates that some of our methods have more than one functionality, which should be split into multiple methods. SIG recommends to take a critical look at the longer methods in the code base and see if they can be split. Comments in a method can already indicate that a new functionality is started, which could be split off to a different method. Splitting these methods into shorter methods with one functionality improves understandability, maintainability and testability. Most of the methods were already shorter than 20 lines of code, with some exceptions of up to 30 lines of code, so improving on this point was rather easy. For the final product most methods were kept within 15 lines of codes, with 20 as a maximum. Some of the larger classes were also split up into multiple smaller classes.

A lower score for unit interfacing means that methods have an above average amount of parameters. This could indicate there are data clumps. It could also lead to difficulty when calling methods and longer, more complex methods. This issue has been resolved by checking all methods and reducing the amount of parameters to 4 or less.

Apart from these two points, SIG mentions that the amount of test code is promising and that these two points are just minor details which could be improved to get a perfect score. For the second iteration extra attention was payed to the points mentioned by SIG and other things were kept consistent with how they were done before.

Research on SIG guidelines

In order to improve the code even further, we have sought for guidelines on how SIG evaluates the code. All points which they check on have been listed below, with a short explanation for each of them. These guidelines have been found in one of SIG's featured publications[2].

- Unit guidelines
  - **Short units** - Methods should be small. SIG recommends the majority of the methods to be within 15 lines. Whitespaces and the starting and ending line of methods are not counted, long lines of codes are counted as multiple lines.
  - **Simple units** - Methods should have a low cyclomatic complexity, such that they are easier to understand and test. SIG recommends to use at most 4 branching operations, such as if statements, per method.
  - **Duplication** - Code should not be duplicated. This can cause bugs to be coppied, code to be adjusted in only one place and the code base to become larger.
  - **Unit interfacing** - The amount of parameters per method should be kept small to avoid data clumps and make code easier to understand. Data clumps are collections of parameters which are often passed together and have little meaning when on their own. Data clumps should be grouped within an object before they are passed as a parameter.
- Architectural guidelines
  - **Separate concerns in modules** - A class should have a single responsibility. SIG also gives 100 lines of code as a guideline for the length of classes.
  - **Loose coupling** - Basically the same as above, but for larger components. Components should not have too many dependencies on each other.
  - **Balanced components** - SIG recommends to have around 6 to 12 components, each of about equal size.
  - **Keep codebase small** - Code bases should be kept small, as this improves the maintainability. They recommend to split code bases at around 175,000 lines of java code. For other programming languages this number can be different.
- Enabling guideline
  - **Automate tests** - Try to test as many things as possible with automated tests. Things which are hard/impossible to test should be tested manually though (such as security).
  - **Write clean code** - Perform peer reviews to ensure that the code is understandable by other people.

During the project, we have not violated the unit guidelines often. There were only a few cases of large methods (at most 30-35 lines) and classes, but these have all been reduced in size. We found only two cases where methods had more than 4 parameters. The architectural guidelines are harder to violate, as the code base is relatively small, but we have also put a lot of effort to keep the larger architectural components clean. The enabling guidelines have also been followed. A lot of unit tests have been written, achieving a high amount of code coverage and all code has been reviewed by every member of the team before it was merged into the master branch.

[2] https://www.sig.eu/wp-content/uploads/2016/10/Real-World_Maintainable_Software_SIG.pdf

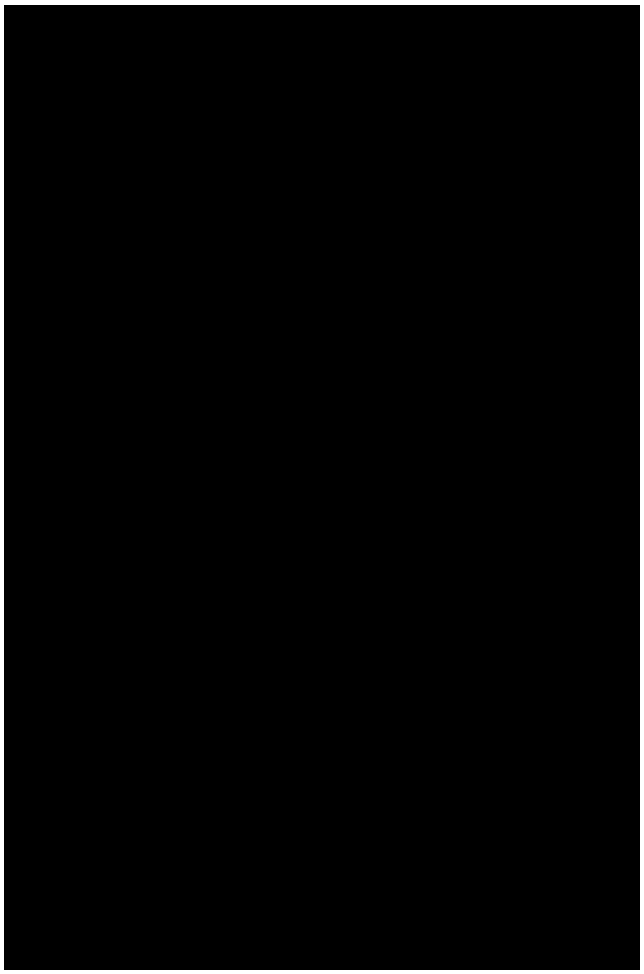
5 Product

This section will present the final product. The final product has different components, the use of these components will be explained as well.

5.1 Web application

For the final product we decided to create a dashboard web application that gives insight into the operations of the Scalar system on the farm. Different forms, such as emails, were considered as well, but a web application gave us the most flexibility and was also preferred by the client and farmers. [redacted]. This makes further distribution of the application relatively easy.

[redacted]. This section will now further explain how the product works and some of the choices that were made. In section 5.1 a screenshot of the web application is shown.



5.1.1 Overview

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

5.1.2 [redacted]

[redacted]

5.2 Use cases of the product

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

6 Implementation

In this section, the implementation of the website and the cloud processes to generate the data which is shown on the website is discussed.

6.1 Programming language and IDE

As was stated in requirement iv., the software must be able to run on the Microsoft Azure cloud platform. The log preprocessing code, which was mentioned in 3.6, is also running in the cloud. The log preprocessor is written in C# and this seems to be one of the best supported languages by Azure, so the decision was made to write our code in C# as well. The website has been made using the ASP.NET framework, following requirement viii.. This also required some JavaScript, to interact with existing JavaScript libraries.

As integrated development environment (IDE), we have used Visual Studio. Partly because Visual Studio was used for the existing code, but also because it seemed like the best IDE for .NET development and has useful integration with Azure. Especially the automatic publishing function of Visual Studio, to update the web application and cloud processes on Azure, showed to be very useful during the project. The NuGet package manager, which is built within Visual Studio was also very helpful.

6.2 Project structure

The project is separated into multiple Visual Studio projects and shared projects. This way, the different components were loosely coupled, thus increasing the maintainability and understandability of the code. For each project there is also an accompanying test project. There are four main projects:

LogPreprocessor - Reads the raw log files into objects and serializes them into a more convenient format. The logic to parse the raw log format has not been implemented during the project, so its details will not be discussed in this report. The model classes in which the data is stored, however, have been refactored and altered a lot, so these will be discussed.

ReportGenerator - Reads the serialized objects made by the LogPreprocessor and generates the data which is used by the website. This project includes pattern recognition to find outliers and level changes.

ReportWebsite - Displays the data generated by the ReportGenerator on a website.

The shared projects are listed below:

SharedCloudAccessManager - Manages the access to our cloud storage account. This includes methods to read and write data, iterate over farms and more useful utilities.

SharedT4C_CloudAccess - Manages the cloud access to the T4C cloud storage account. This storage account contains the raw log files, which are used only by the LogPreprocessor.

SharedLogPreprocessorModel - Contains the model classes of the LogPreprocessor project. These classes are used to serialize the preprocessed data of the LogPreprocessor, after which it is stored to the cloud and later deserialize it again when it is needed by the ReportGenerator.

SharedAnalysisModel - Contains the model classes of the ReportGenerator. These classes are used to serialize data by the ReportGenerator and deserialize it again in the ReportWebsite. This works similar to the serialization and deserialization of the SharedLogPreprocessorModel.

SharedUtilities - Contains utilities, such as C# extension methods.

The dependencies between the projects and shared projects are visualized in figure 4. Besides the shown dependencies, all main projects also have dependencies on the SharedUtilities and SharedCloudAccessManager projects. None of the main projects has a dependency on another main project.

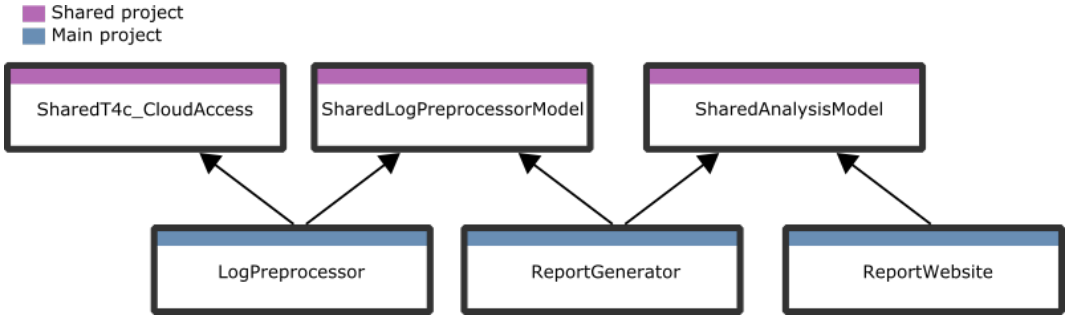


Figure 4: Dependencies on shared projects

6.3 Data network

To gain a better understanding of which steps are performed to retrieve, process and analyze the raw log data, figure 5 was made. First the log data is generated by the Scalar system and locally stored on the T4C PC. Once a day, the T4C PC will upload the log data to the cloud storage account of T4C. Next, the Preprocessor module fetches the data and stores the processed log files into our own Azure cloud store account. The fetching is done by polling once every hour if new data is available. This usually happens around 1am to 6am, so the potential one hour delay is not a problem. If needed, the Azure module which executes this code can also be upgraded, to allow for a higher frequency polling rate. After this, the ReportGenerater module (indicated as Analyzer) analyzes the preprocessed logs and stores the result into reports, which can be shown by the website application to the end user.

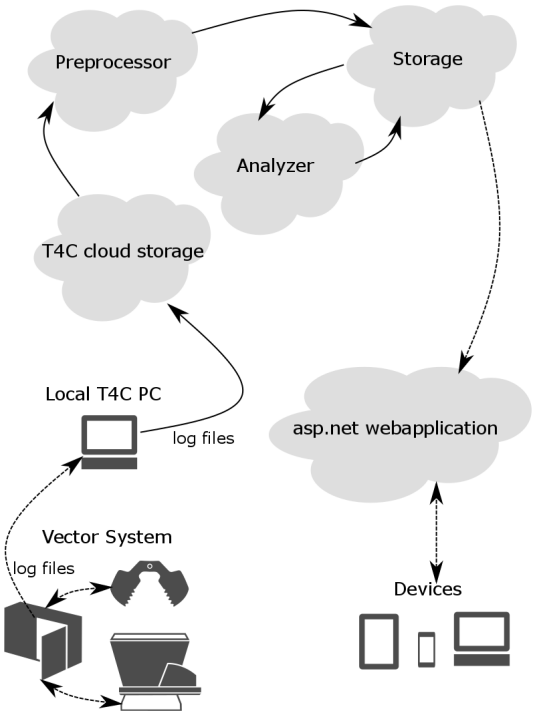


Figure 5: The network through which log data travels, to reach the end user

6.4 Interface to Azure

The first code that was written, is an interface to the Azure cloud storage account. This code now resides in the SharedCloudAccessManager project and is used by all of the main projects to store and retrieve data. The classes of this project are visualized in figure 6.

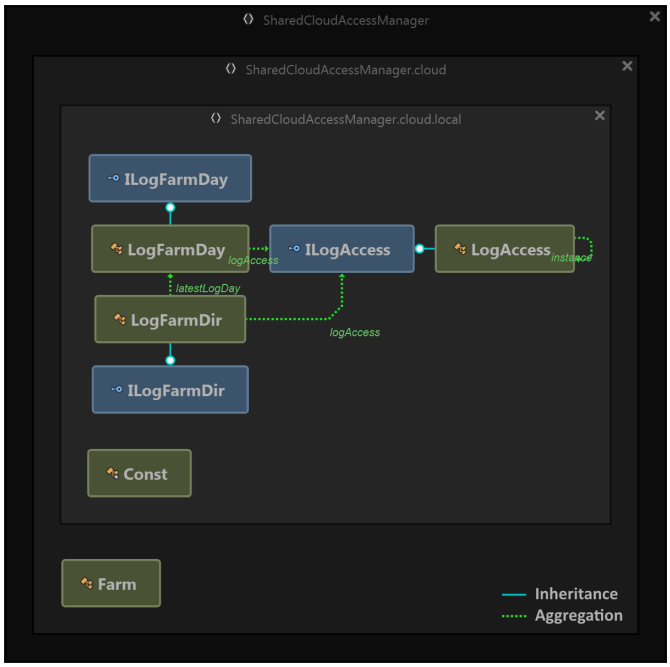


Figure 6: SharedCloudAccessManager structure, image made with ReSharper

As can be seen, the cloud access project is fairly simple. The *Farm* class stores the name, address and unique key of a farm and the *Const* class stores several constants, such as file names and the cloud access key. The other classes require some prior knowledge of the file structure, which is partly shown in figure 7.

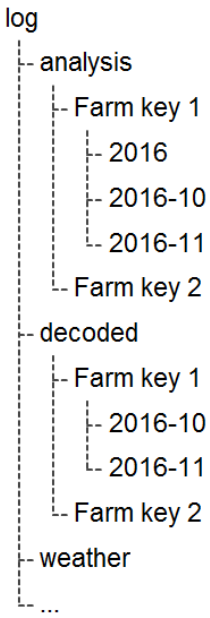


Figure 7: Cloud storage file structure

The directory *decoded* contains the preprocessed log files and the *analysis* directory contains the data analysis, which is created by the ReportGenerator and used by the ReportWebsite. These directories contain a subdirectory for each farm of which data is available.

The *LogAccess* class contains general methods to read objects from and write objects to the cloud, as well as methods to iterate over all farms of which data is available. The *LogFarmDir* class represents a farm. It can be used to get the newest or oldest logged day, as well as the newest analysis day. A log or analysis day can also be retrieved for an arbitrary date. Finally, the *LogFarmDay* class can be used to read or write the log, analysis, or weather data for a specific farm and date.

6.5 Raw log preprocessing

The log preprocessor module parses the raw logs and stores the read data into the model classes, which are shown in figure 8. Almost every class represents an activity, such as MfrWaiting, MfrLoading, FeedGrabberDumping, or PdbAlarm. These are activities of the MFR, FG, PDB, or dispenser. There are, however, also some classes which store data which are not activities. These are the classes: *SoftwareVersions*, *FeedNames*, *CommunicationPerformance* and *MissingLogs*.

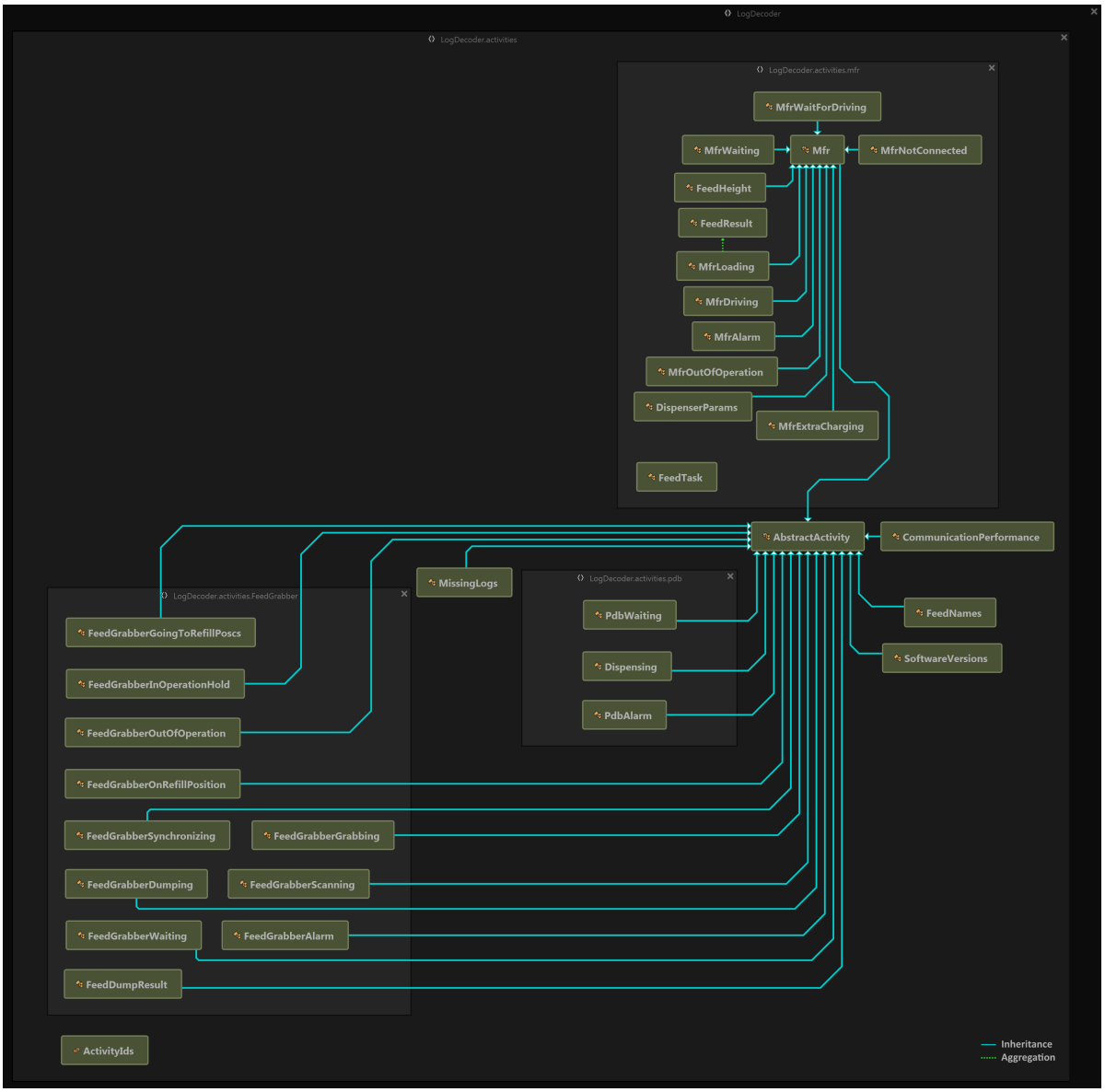


Figure 8: Preprocessor model structure, image made with ReSharper

The SoftwareVersions class stores the version numbers of software running on the robots and other devices. FeedNames contains a map from feed id (integer) to feed name (string). CommunicationPerformance contains the percentage of messages which was sent without communication errors. MissingLogs stores the moments of the day at which log data was missing. All other classes describe activities and their content should be obvious from their name.

6.6 Data analysis

As discussed in section B.5 of the research report, there is a need for analysis of the data. This section discusses the implemented methods to analyze the data, including both the implementation details and the quality and performance.

6.6.1 Normalization

To ensure that extreme outliers do not have a large impact on the data analysis algorithms, they are first filtered out. This process is called normalization of the data. To perform the normalization, a small sliding window is passed over the data. The size of this sliding window can be set to an arbitrary number. A sensitivity is also needed by the algorithm. The sensitivity determines how much a data point should deviate from the other data points, before it is marked as an outlier which should be filtered by the normalization algorithm. While the sliding window is passed over the data, the mean (μ) and standard deviation (σ) of the values within the window are calculated. If the current value deviates more than $sensitivity * \sigma$ from the mean value, it is filtered out by the normalization algorithm.

In figure 9, the data points within a single sliding window of 14 points have been visualized. Notice that there are actually 15 points in the figure: the point surrounded by a red circle is exactly in the center and is currently being checked. To check if this point is an extreme outlier, the mean and standard deviation of the 14 points surrounding it is calculated. In the figure, the black line indicates the mean and the purple dotted line the standard deviation, relative to the mean. If the sensitivity would for example be set to 5, the point in the center would be filtered by the normalization algorithm if it is further away than 5 times the standard deviation from the mean (black line). This is the case in the shown picture, as the point is 6.9σ away from the mean.

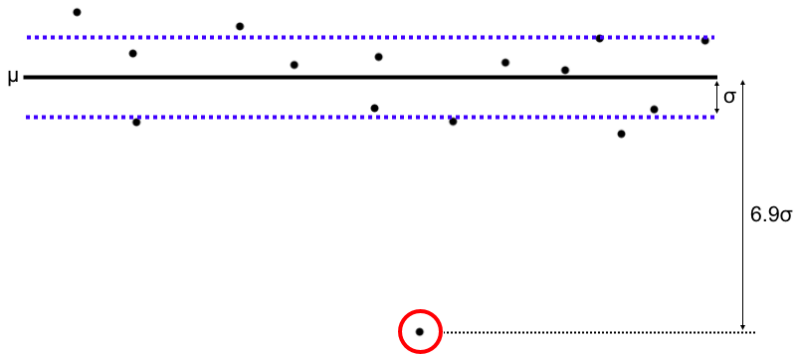


Figure 9: Data normalization example

6.6.2 Level detection

After the normalization process, the level detection algorithms is executed. With a level, we mean a part of the data set which is similar and clustered together. This is visualized in figure 10, where the levels of the data set have been separated by vertical lines.

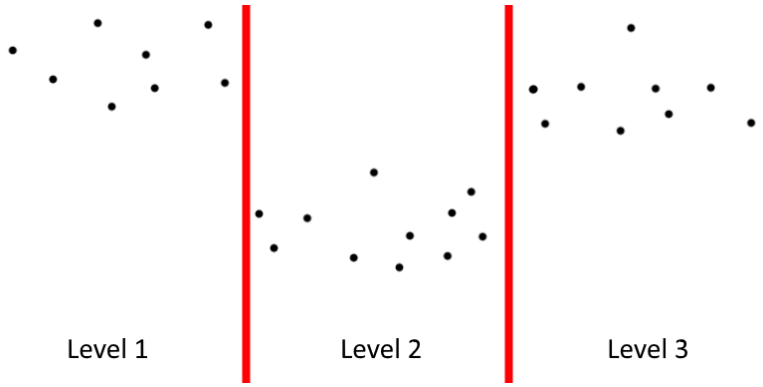


Figure 10: Example containing 3 data levels

A more mathematical way to describe a level would be to think of the data as values which are randomly picked from a normal distribution. If a data point is picked from the same normal distribution as the previous data point, they both belong to the same level. If enough consecutive data points belong to a significantly different normal distribution, these data points form a new level. The implementation of the level detection algorithm has been derived from this idea.

To check if a level change has occurred at a specific point, two consecutive sliding windows are used. Of each of these windows, the mean and variance of the data within each sliding window is calculated. Given these two values, the estimated normal distribution of each sliding window can be constructed. Finally the overlapping area of the two normal distributions is calculated to determine how similar the data in the two windows are. If this overlapping area is smaller than the inverse of the sensitivity, a level change is detected. Figures 11 through 14 visually show a few steps of this process. The used window size is 7 and figure 11 shows the used data set.

Figure 11: Example data with 2 levels

Figure 12 shows the two sliding windows at the first position of the data set. These two windows contain similar data, so the two normal distributions almost fully overlap. The normal distributions and the overlapping area are shown to the right of the graph.

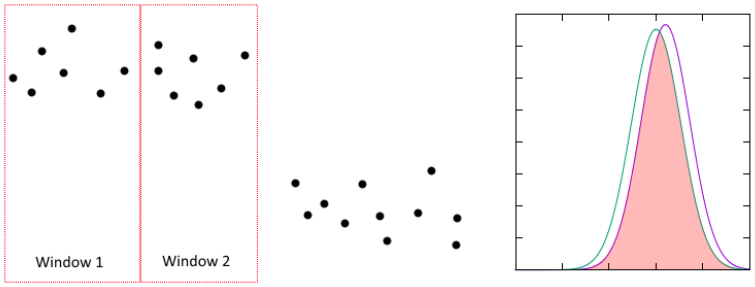


Figure 12: Example data with two windows over the same level

A few iterations later, the situation as is shown in figure 13 is reached. Here the normal distributions are fairly unsimilar.

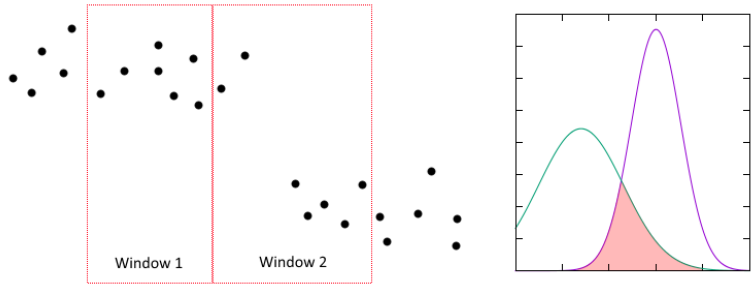


Figure 13: Example data with two windows partially over the same level

The position where the lowest similarity between the two windows is detected, is shown in figure 14. The first data point of the second sliding window will be marked as the start of a new level. To ensure other points around this area will not be marked as level change points as well, only a single level change point is allowed within a specific range. In our algorithm this range is set to be equal to the window size. If multiple level changes are detected within this range, the point at which the lowest equality between the two windows is detected, is chosen.

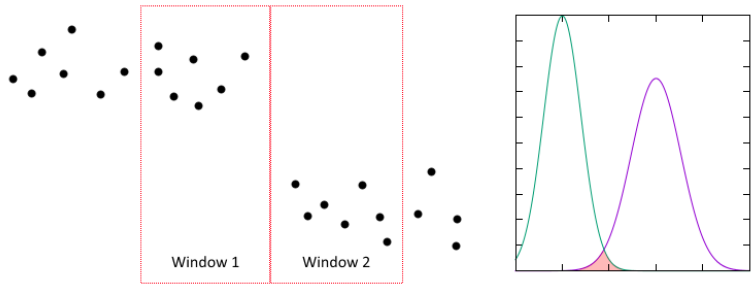


Figure 14: Example data with two windows over different levels

6.6.3 Outlier detection

The outlier detection algorithm is similar to the normalization algorithm, the only difference being that the outlier detection limits itself to finding outliers within a single level, thus being more accurate. The reason for this is that level changes often occur because of settings, or feed types, which are changed. If the levels which exist because of these changes would not be treated individually, the standard deviation would increase for the windows near the level transition points, resulting in fewer outliers being detected.

6.6.4 Performance

The data analysis code has been designed to run in linear time. To verify that the code really does run in linear time, we have run the code on various input sizes and plotted the resulting durations. The result is shown in figure 15. The graph clearly follows a linear trend, thus showing the dominant factor of our algorithm runs in linear time, or at least up to the tested input size. Currently, only much smaller input sizes are handled by the algorithm, but this shows that the used algorithms are scalable and can handle much larger input sizes if needed.

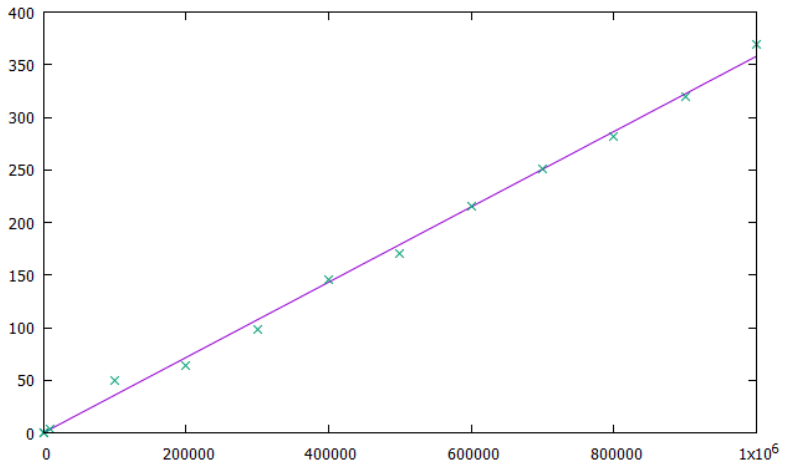


Figure 15: The runtime in milliseconds (y-axis) for various input sizes (x-axis)

6.6.5 Code structure

The structure of the data analysis code is shown in figure 16. The data that is generated by these classes, is stored in the model classes that are shown in figure 17. These model classes are downloaded by the web application to create the overviews.

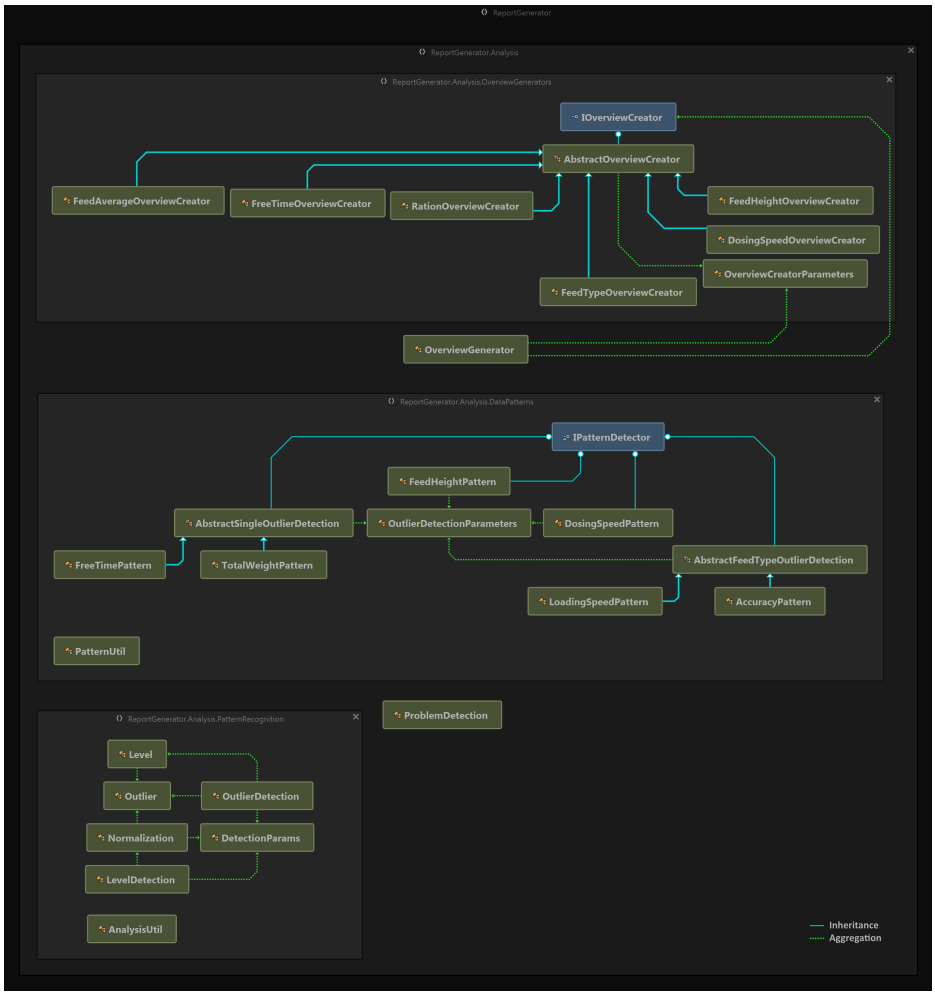


Figure 16: ReportGenerator structure, image made with ReSharper

As can be seen in figure 16, the analysis code contains multiple overview creators, specific pattern detector classes and the normalization, level detection and outlier detection code. The overview creators parse the preprocessed log data, aggregate some useful data and store it again. An example of this, would be the FreeTimeOverviewCreator, which sums the moments of time at which 'waiting' actions overlap for the FG and both MFRs.

The normalization, level detection and outlier detection code work as has been explained in sections 6.6.1, 6.6.2 and 6.6.3. First the data has to be normalized, then the level detection code is executed and finally, the outlier detection algorithm finds the outliers of each level.

The specific pattern detectors use the normalization, level detection and outlier detection code to find patterns in specific types of data. The FreeTimePattern class, for example, tries to find outliers in the free time data, which has been generated by the FreeTimeOverviewCreator. When the outliers have been found, the specific pattern detectors create a human-readable text, which can be added to the 'points of attention' list of the web application.

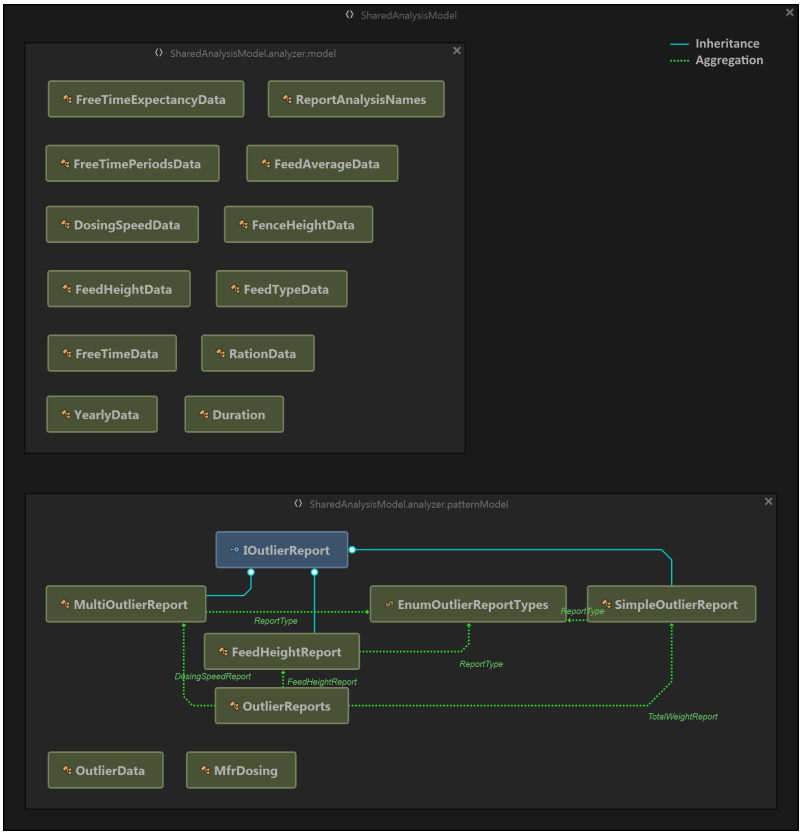


Figure 17: SharedAnalysisModel structure, image made with ReSharper

The SharedAnalysisModel contains the model classes of the ReportGenerator and is shown in figure 17. The classes in the analyzer.model namespace (shown at the top) store the data which is generated by the overview creators and the analyzer.patternModel namespace (shown at the bottom) store the pattern detection result, including the human-readable texts.

6.7 Web application

The website is implemented as an ASP.NET web application, using the MVC design pattern. The MVC pattern separates the application in three units: the model, the view and the controller. Each has its own responsibilities, keeping the code understandable and increasing the maintainability. The model is responsible for representing the data and keeping the data logic. The view is responsible for

the presentation of the data in the user interface, it should not contain any logic. In a web application it consists mostly of html and css files. The controller handles all the events. When a computer requests the web application for example, the controller will handle the request and load the data, using the model classes to deserialize it and pass the data on to the view, which will show the data in an html page. Our application reads the data from the cloud in the HomeController class, into different instances of model classes. By passing these on to the view, the cshtml files are able to represent the data. The cshtml files call javascript methods to draw the graphs on a canvas. Also some javascript is used to fill dropdown boxes with legends. Cshtml files are normal html files in which C# code can be used as well. This works very straightforward. The C# code is executed at the server side, sending the result in pure html to the client. figure 18 gives a visualization of the web application classes. The cshtml classes are missing from this overview, as the tool which was used to generate these pictures could not recognize them.

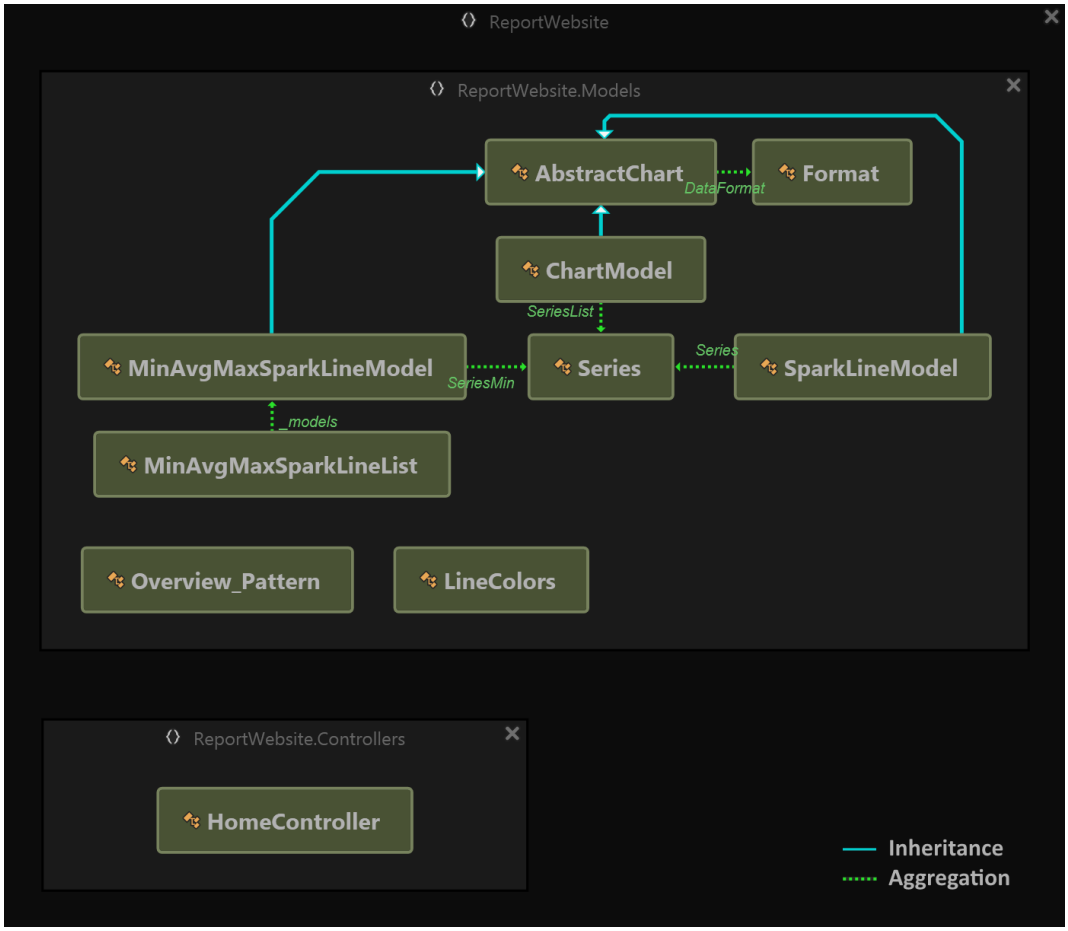


Figure 18: ReportWebsite structure, image made with ReSharper

6.8 Testing

Testing is an important part of the project. During the project we tried to cover all the written code with tests, to limit the amount of bugs and ensure the structure of the code is good. The usability of the product has also been tested manually by us and by the farmer.

6.8.1 Unit testing

After each produced piece of code, unit tests were written. With the unit tests, we tested the correct behaviour of methods and classes. All components which depended on other classes were tested by

mocking those classes, such that the code was tested in isolation. The C# Moq library[3] was used in order to mock classes.

To ensure we tested as much as possible, we used a coverage tool to give an overview of the coverage. We have written a total of 354 tests, which resulted in a code coverage of 94%. The final coverage can be found in figure 19. In the figure, we only expanded the namespaces with less than 90% coverage. The reason why these classes are not fully covered, is because they read or write cloud data, which makes them difficult to test.

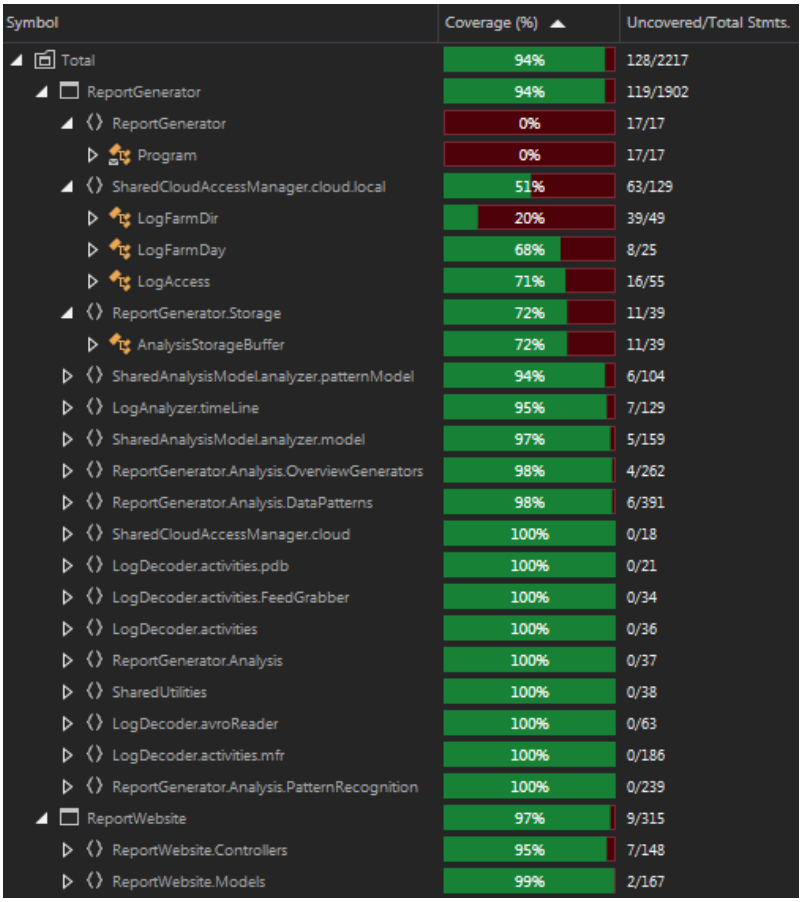


Figure 19: Code coverage - only namespaces with less than 90% coverage are expanded, image made with ReSharper

6.8.2 Verification

To verify if the product is what the user wants, we visited a farmer multiple times during the project. After each iteration, we planned a session with a farmer to gather feedback and discuss our new implementations. These sessions were helpful to test if the website was useful for the farmer and if the project was heading in the right direction. During the last visit, we asked the farmer to test the website for a few days and send us his experience with the product. We expect to receive this feedback a couple of days after the deadline of this report, so we cannot add the results.

Besides getting feedback from the farmer, we also communicated and discussed with employees at Orchid about the information we thought is useful for the website. The employees of Orchid have a good understanding of what the farmers might be interested in and what could be useful to them. Since traveling to the farmers took quite a long time, this was a convenient alternative for smaller questions.

[3]https://github.com/Moq/moq4/wiki/Quickstart

7 Ethics

The log data we use, is stored in the cloud. This has several benefits, but concerning ethics, the cloud has also some negative issues. In this section, some of the ethical issues will be discussed.

When the data is stored in the cloud, the owner of the data losses the control of it, because the data is not stored on his own computer. The cloud provider can access the data at any time and has the power to alter or delete the data. Costumers trust their cloud provider and the cloud provider should have interest in the security of the costumers data. Besides the cloud provider, the data can also end up in the hands of other people, when there is a data leak for example. The cloud provider is even permitted to share the data with third parties, if it is stated in the privacy policy of the provider. The current data rules and regulations provide support for the users of the cloud.

[redacted]. This data seems innocent, but can give other people insight in the farms business. The costumer, as well as the cloud provider, should be aware of this. [redacted]. The feeling of being watched can have impact on the farmers thrust in the system.

[redacted].

8 Client and customer reaction on the project final product

In this section, the reaction from the client at Orchid and a customer of Orchid (a farmer) will be discussed.

8.1 Reaction from the client at Orchid

The client is positive about the project. He has stated the product satisfies the expected functionalities and had no negative comments about the collaboration and our code. The only critical note is that he would have preferred us to ask for feedback from the customers (farmers) during the development process more frequently. At the start of the project we did visit three farmers to interview them to establish the requirements. Halfway during the project and near the end we visited one of the farmer again, to verify the results of the product, but visiting the other two farmers multiple times as well would have been preferred.

[redacted]

8.2 Reaction from a farmer (customer)

We have asked one of the farmers two times to give feedback on the product, once around halfway through the project, just after the first iteration and once at the end. During the first feedback session, the farmer was very positive about the state of the product. [redacted], there were only two minor issues. The first being that unnecessary decimal values were shown. These decimals have been removed. [redacted]. We explained that this would result in harder to understand graphs, since the loading speed depends on how many kg of feed and which feed types are in the currently loaded recipe. This graph is also meant to show the change in loading speed over time, so the exact values are less important. They agreed that it would be hard to show the loading time per MFR, so we have kept the loading speed as it was.

During the second feedback session, we have shown the product in its near-final state. Only the report indication colors (green, orange, or red line below the title of the different sections) were not implemented and the list of points of attention still had to be improved. The feedback at first sight was positive and we have requested the farmer to try out the application for a couple of days, to test which parts are useful and which should be improved. Unfortunately we will get the feedback after the deadline for the report, so this cannot be included.

9 Reflection on the requirements and used tools

In this section we will reflect on the requirements and the used tools and solutions. The requirements which were implemented are marked in green, while the non-implemented requirements are left black.

9.1 Functional requirements

Below is the list of all functional requirements. As can be seen, all 'must haves' have been implemented and some should haves were implemented too. Requirement 4 has not been implemented, because this information turned out to not be useful for the farmer. It could however be interesting to the service departments of Orchid, but this was not within the scope of the project. Missing requirements 6 and 7 have not been implemented, because they are hard to implement and because of a lack of time. Using labelled instead of unlabelled data would also be a lot more useful. Gathering labelled data, by detecting and marking problems at farms, would however require a considerable amount of time. The could haves are minor points, which can be seen as "nice to haves". Missing these is not a problem.

Must haves

1. [redacted].
2. [redacted].
3. [redacted].

Should haves

4. [redacted].
5. [redacted].
6. [redacted].
7. [redacted].
8. [redacted].

Could haves

9. [redacted].
10. [redacted].
11. [redacted].

Won't haves

12. [redacted].

[redacted].

9.2 Non-functional requirements

As can be seen, all of the non-functional requirements have been followed.

Must have

i. The software must be kept private.

ii. All code must have documentation so it is clear what the purpose of each part of the code is.

iii. The software must be developed following the agile methodologies to allow for software adjustments if the user's requirements change.

iv. The developed software must be able to run in the Microsoft Azure cloud, since Orchid already uses this.

Should haves

v. [redacted].

vi. [redacted].

vii. [redacted].

viii. [redacted].

9.3 Used tools

We are generally positive about the used tools. Microsoft Azure was easy to use, providing a clear user-interface and a lot of functionality. Sometimes finding the right documentation was however difficult, but we have eventually found the answer to all of our questions somewhere. Chartjs and bootstrap were also fairly easy to work with and produced good quality results. The only downside to chartjs was that some functions/parameters are undocumented, or at least we could not find the documentation. Gitlab also worked as expected: very similar to github, which we have used during earlier projects.

10 Conclusion

This project was started with the farmer in mind. Farmers were in need of a solution that would give them more insight into operation of the Scalar on their farm. Developing a solution that was directly for the customer was something that interested us and motivated us during the project. During the project, [redacted] was developed to give farmers more insight into the Scalar system operating on their farms. The reactions on the application have been positive so far and the application seems to fulfill the needs of the farmers. We are therefore pleased with the result and consider the project a success.

While the application is deployed and running continuously at the moment, [redacted]. It was not our task to collect the data, [redacted] hugely impacts the usability of our product. Also, the points of attention section of the web application can be further improved to detect more problems and show potential causes and solutions of the issues. This is something for which we did not have enough time, nor data, unfortunately. Further development is therefore needed. Hopefully the application build during this project shows the added customer value of such an application, such that development will continue.

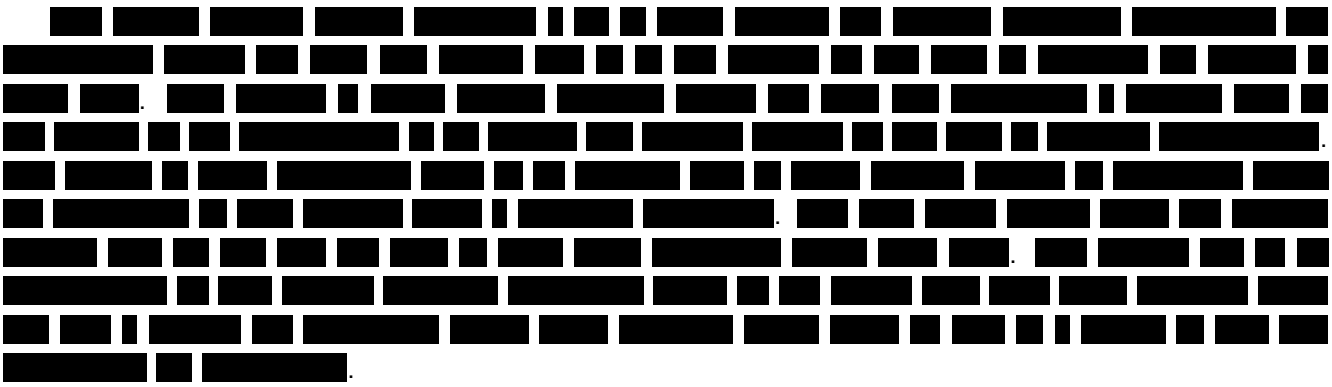
During the project we mostly worked at Orchid, at the development department. This made it easy to plan meetings, conduct interviews and ask questions to our supervisor and other employees of Orchid. Also visiting different farms at the start of the project was a good way to get a clear picture of the needs of the customer. Another good decision was to use a website for the daily overview. This made it easy to implement the dashboard into a framework of already existing applications, making deployment to the farmer very easy. Visiting the test farms was something that happened a few times, but could have been done more frequently. This would have improved the feedback and might have increased development speed.

As a project team, we had a lot of fun during the project and really enjoyed to work directly on something that was needed by farmers. Visiting the farms and seeing how the Scalar system worked made the project a lot more entertaining. We have also learned a lot by working together at Orchid and with lots of new technologies, such as C#, Microsoft Azure and ASP.NET. During previous projects of the bachelor, we usually did most of the work at home and only were together to discuss the progress and next steps. Actually working together on a product was a great experience and helped greatly with the transfer of knowledge and when there were development issues. Considering the solution that was developed and the positive reactions from farmers and Orchid employees, it seems only logical that development will continue.

11 Recommendations

A Project Description

This is the original project description as it appeared on BEPsys before the start of the project:



B Research Report

[redacted]

- [redacted]
- [redacted]
- [redacted]

[redacted]

[redacted]

B.1 Project plan

[redacted]

B.1.1 Goal

[redacted]

B.1.2 Approach

[redacted]

[redacted]

B.2 Problem description

[redacted]

[redacted]

- [redacted]
- [redacted]
  - [redacted]
  - [redacted]
  - [redacted]
  - [redacted]
- [redacted]
- [redacted]
- [redacted]

[redacted]

- [redacted]

  [redacted]

- [redacted]

  [redacted]

[redacted]

1. [redacted]
2. [redacted]
3. [redacted]

B.3 Initial requirements

[redacted]

B.3.1 Functional requirements

[redacted]

Must haves

1. [redacted]
2. [redacted]
3. [redacted]

Should haves

4. [redacted]
5. [redacted]
6. [redacted]

7. [redacted].
8. [redacted].

Could haves

9. [redacted].
10. [redacted].
11. [redacted].

Won't haves

12. [redacted].

B.3.2 Non-functional requirements

[redacted].

Must have

i. [redacted].

ii. [redacted].

iii. [redacted].

iv. [redacted].

Should haves

v. [redacted].

vi. [redacted].

vii. [redacted].

viii. [redacted].

B.4 Technical decisions

[redacted].

B.4.1 Framework and programming language

[redacted]

B.4.2 [redacted]

[redacted]

B.4.3 Testing and verification

[redacted]

B.5 Research on data analysis

[redacted]

B.5.1 Pattern recognition

[redacted]

Novelty detection

[redacted]

Linear regression

[redacted]

[redacted]

Clustering

[redacted]

Classification

[redacted]

Semi-supervised learning

[redacted]

B.5.2 Data analysis libraries

[redacted]

Accord.NET framework

[redacted]

[redacted]

Azure Machine Learning

[redacted]

[redacted]

[redacted]

B.6 Schedule

[redacted]

B.6.1 Iteration one

[redacted]

B.6.2 Iteration two

[redacted]

B.6.3 Iteration three

[redacted]

B.6.4 Last week

[redacted]

C Interviews

This appendix has been removed, as it contained sensitive information to Orchid.

D Infosheet

Title of the project: Automated report generation from agricultural robot log data.
Name of the client organization: Orchid N.V.
Date of the final presentation: Juli 4, 2017
Description: [redacted]

Members of the team:
Name: Michiel van den Berg
Role & contribution: Member of the Scrum Team. Tester.

Name: Faris Elghlan
Role & contribution: Member of the Scrum Team. Implementation of the pattern recognition algorithms and interface to the cloud storage account. Also main point of communication with the team.

Name: Timo van Leest
Role & contribution: Scrum master. Implementation of the ASP.NET web application.

All team members contributed to researching the requirements of the product and verifying the results, by visiting farmers and conducting interviews.

Client (Scrum Product Owner): Jan Jacobs, Orchid N.V.
TU coach: David Tax, Pattern Recognition & Bioinformatics group
Contacts:
Michiel van den Berg, m.r.vdberg95@gmail.com
Faris Elghlan, faris.elghlan@hotmail.com
Timo van Leest, t.vanleest@student.tudelft.nl
The final report for this project can be found at: http://repository.tudelft.nl

E SIG feedback

This appendix contains the emails with feedback from SIG about the maintainability of the software. Note that the second round of feedback has been received after the deadline of this report, so it is not referenced anywhere in the report. For anyone not speaking Dutch, the second feedback text basically says the following:

The size of the system, as well as the maintainability score have increased. Mainly the unit size has improved, unit interfacing only a little bit. The amount of test code has also increased and there is a good ratio between the production and test code. From this, SIG concludes that we have largely followed their advice.

E.1 Feedback first upload

De code van het systeem scoort 4 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door lagere scores voor Unit Size en Unit Interfacing.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. Binnen de langere methodes in dit systeem, zoals bijvoorbeeld de 'AvroReader.ReadAll'-methode, zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes. Commentaarregels zoals bijvoorbeeld 'Get the log file containing the action with the smallest time to ensure that the log lines are decoded in the correct order' zijn een goede indicatie dat er een autonoom stuk functionaliteit te ontdekken is. Het is aan te raden kritisch te kijken naar de langere methodes binnen dit systeem en deze waar mogelijk op te splitsen.

Voor Unit Interfacing wordt er gekeken naar het percentage code in units met een bovengemiddeld aantal parameters. Doorgaans duidt een bovengemiddeld aantal parameters op een gebrek aan abstractie. Daarnaast leidt een groot aantal parameters nogal eens tot verwarring in het aanroepen van de methode en in de meeste gevallen ook tot langere en complexere methoden.

Bij jullie project is de constructor van SoftwareVersions een uitschieter, idealiter zou het mogelijk moeten zijn om hetzelfde resultaat te bereiken zonder dat je daar een constructor met 100 string-argumenten voor nodig hebt.

De aanwezigheid van test-code is in ieder geval veelbelovend, hopelijk zal het volume van de test-code ook groeien op het moment dat er nieuwe functionaliteit toegevoegd wordt.

Over het algemeen scoort de code dus bovengemiddeld, dus bovenstaande aanbevelingen zijn voornamelijk kleine puntjes om een nog hogere score te bereiken tijdens de rest van de ontwikkelfase.

E.2 Feedback second upload

In de tweede upload zien we dat zowel de omvang van het systeem als de score voor onderhoudbaarheid is gestegen. De stijging wordt voornamelijk veroorzaakt door aanpassingen op het gebied van Unit Size, dat in de feedback op de eerste upload als verbeterpunt genoemd werd. Bij Unit Interfacing zien we minder verbeteringen, waardoor er op dat vlak weinig voortgang is geboekt sinds de eerste upload.

Ook is het goed om te zien dat jullie naast nieuwe productiecode ook aandacht hebben besteed aan het schrijven van nieuwe testcode. De verhouding tussen productie- en testcode ziet er ook goed uit.

Uit deze observaties kunnen we concluderen dat de aanbevelingen van de vorige evaluatie grotendeels zijn meegenomen in het ontwikkeltraject.

Acronyms

FG Feed grabber. 6, 7, 26, 31

IDE integrated development environment. 23

MFR Mixing and feeding robot. 6, 7, 26, 31, 35, 55

MVC Model view controller. 14, 31

PDB Power distribution box. 6, 26

SIG Software improvement group. 14, 15, 17, 18, 53

Glossary

.NET A software framework developed by Microsoft, containing a large class library and allowing for language interoperability between the supported languages. This is done by compiling each language into the Common Intermediate Language (CIL), which can then be executed in the Common Language Runtime (CLR), which provides functionality such as exception handling and garbage collection.. 23

ASP.NET ASP.NET is a server side web application framework. ASP stands for active server pages. It is a technology to create dynamic html pages for our web application. 12, 14, 23, 31, 38, 52

dispenser The device which can dispense smaller quantities of feed into the Mixing and feeding robot (MFR). 6, 26, 55

feed grabber The crane that grabs the food and puts in the Mixing and feeding robot (MFR). 6, 7, 54

git A version control system that keeps track of changes in files and enables developers to work collaborative on projects. 12

mixing and feeding robot The robot which mixes the food together and drives it to the feed fences, where it is dumped and the cows will eat it. 6, 7, 54, 55

model view controller Separates an application in a model, a view and a controller. Each part has its own responsibilities to keep the code understandable, readable and increasing maintainability. 14, 54

novelty detection Detecting deviations from regular data. This method is useful if a lot of regular data is available to train with and deviating behavior which has never occurred before, or does not occur often, has to be detected. 11

Orchid The codename for the company for which this project is conducted. Orchid is an innovator in agriculture and develops machines for the cattle industry. This name was used to avoid google searches on the real company name. 1–3, 6, 8, 10, 12, 14, 15, 33, 35–38, 51, 52, 56

outlier detection Detecting points in the data that do not conform with the other data in the pattern. 14, 15, 29, 31

pattern recognition Recognizing patterns and regularities in data. Closely related to machine learning. 11, 14, 23, 52

power distribution box The control unit which directs the dispenser and decides which actions may be performed. 6, 54

ReSharper A Visual Studio extension for .NET developers, offering tools such as code quality analysis, unit test coverage visualizations and more. 25, 26, 30–33

Scalar The codename of the feedings system as a whole. This name was used to avoid google searches on the real product name. 3, 6–8, 11, 19, 24, 38

software improvement group The software improvement group is the group that rates our code twice during the project. Pointing out points that can be improved and giving us a rating. 14, 54

T4C A group within Orchid, which manages the web interfaces to the robots made by Orchid and storage of data to the cloud. 23, 24

Technical University of Delft The technical university of Delft. This is the university where the students of this project are doing their bachelor study. 1–3, 14, 15

test farm A farm which Orchid uses to test their new software versions and new hardware. If anything new is developed it first has to be tested at a test farm before it can be released to other farms. 3, 14, 15, 38