Project Plan

Quinten Van Opstal — Dean Polimac — Cas Ebels
Erfan Mozafari — Žygimantas Liutkus

May 9, 2023

Contents

1 Introduction

This report outlines the basis laid out in order to deliver a monitoring software for Ruisdael. The report is divided into six sections. Section 2.5 deals with the analysis of the problem according to which the system will be modeled, as well as further information regarding stakeholders, users and existing products. Section 3 discusses the feasibility of creating such a system, and provides a risk analysis of it. Section 4 lays out the client requirements in accordance to the MoSCoW analysis. Section 5 brings forth the groups approach towards tackling the requirements and creating the given system. Lastly, Section 6 covers the development methodology of the project and briefly delves into the projected production process of the system.

2 Problem Analysis

Our client Ruisdael has provided us with the opportunity to participate and complete one of their projects. Ruisdael is an initiative that has many weather stations all over the Netherlands. The data collected by these stations is used by, but not limited to, organizations such as RIVM, KNMI, the Technical University of Delft, and many more. 2.1 goes into more detail about what problem the project will solve. 2.2 will cover the stakeholders related to the project. Chapter 2.3 covers what the project will be used for. Section 2.4 discusses alternative solutions. Lastly, section 2.5 discusses the final users and experts who could provide assistance with the projects design.

2.1 Problem description

The project has one main problem: locating and predicting when problems with the weather station's hardware/ connectivity will occur. Some examples are storage running out, unreliability within the network such as random failures in data transfers or random shutdowns, processes halting, and more. This can lead to problems since ACTRIS cloudnet uses this data, under the assumption that it is reliable. Hence, it would be desirable to identify users and resolve issues in situations where the weather stations have problems during the process of data collection.

2.2 Stakeholders

The stakeholders of this project are: Andre Castro, the Ruisdael data manager, who is our main point of contact. He will also be using our dashboard the most. Marc Schleiss, full-time professor at TU Delft who will also oversee the project. The Ruisdael Scientific Steering Committee, they should get better up-time for their weather stations via this project. The organizations that use data from Ruisdael are also stakeholders, since this dashboard can help locate problems faster so the date they receive will be more accurate, and there will

be less downtime. The developers, they implement the software. And they will deliver the end product. Individuals using data from Ruisdael, the data they use will be more reliable. The Tu Delft, since the developers are students from the Tu Delft. And they want anything that is associated with them to follow their standard. Finnish Meteorological Institute ACTRIS cloudnet is another stakeholder, given that their platform uses data provided by Ruisdael.

2.3 Use cases

The product will be used to monitor a data aggregation network and provide a dashboard to display any problems. By providing crucial information: whether the sensor is still up, and if there is a problem with it. These problems include but are not limited to available storage, the weather stations CPU usages, and malfunction in the data collection,

2.4 Existing products

The client already provided the suggestion of using Elastic stack, however, they are open to other open source alternatives. Further research has shown that HP ArcSight Enterprise Security Manager (Igor Anastasov, 2014) would be a potential alternative, however it is not open source, which conflicts with one of the requirements. Another alternative is Prometheus (Hitchcock, 2020), however Elastic stacks fits the project better, since it has built in functionality for heartbeat monitoring.

2.5 Users and experts

Users and experts can provide useful insight into the project design. One being Andre Castro as he is the Ruisdael Data Manager, Marc Schleiss as he is a professor that oversees the project. Additionally, the groups coach Johan Pouwelse is an expert, who could potentially provide assistance in dire cases.

3 Feasibility Study & Risk Analysis

This chapter discusses the achievability of the set goals and requirements as well as some possible problems that may arise throughout the project. Section 3.1 contains the feasibility study of the project. The following section 3.2 provides some possible risks as subsections that explain their significance and suggest plausible solutions to the problems.

3.1 Feasibility

After the initial meeting with the client and creating a rough draft of the demands, our group has deemed the project highly likely to be feasible over the 10 weeks of the course. The requirements are sufficiently extensive to occupy

the team until the end of the project while fully developing, testing and documenting the needed software and working no more than 32 hours per week on all parts of the course combined (development, teamwork and technical writing assignments). If the initial requirements are on track to being fully completed early, the client has indicated the possibility of expanding the project scope. Otherwise, in case the current scope of the tasks would pose infeasible, in a consensus with the client, the requirements could be changed, or a subset of them could be prioritized to finish some part of the project.

Another factor that improves the feasibility of this project is the availability of open-source programs and frameworks (ELK stack), which provide tools that accomplish parts of the requirements' core logic.

Lastly, to develop the requested software, access to actual or copy machines will be needed, as well as access to the server they communicate to. This way the product can be tested and refined to work effectively on the existing system. Their operating systems' versions should also be provided so that a correct version control environment (GitLab) could be set up.

3.2 Risk Analysis

In this section, some possible risks and problems that may be encountered throughout the project will be discussed.

3.2.1 High Network Strictness

The network of Ruisdael may have some security measures or firewalls in place, which could interfere with our implementation of the monitoring software. To circumvent this problem, spending additional time adapting the software to the firewalls in the network would be required. This may delay the project for as long as it would take to arrange a meeting and meet with the client to review this issue. Discussing receiving more permissions in the network or a different way of transferring data with them could help mitigate the issue. However, this complete process could delay the project by at least a week.

3.2.2 Misrepresented Test Environment

A test environment will be provided during the development process to test the product without accidentally harming the client's network. However, an issue may arise if the test environment does not reflect the actual production environment. In that case, the product may only be functional in irrelevant conditions.

3.2.3 Different Installation Across Nodes

The goal is to create a product that can be replicated across different machines in the Ruisdael network. However, it is not currently known if all devices operate on the same or compatible versions of systems. If these versions are incompatible with the same installation process, a delay may be caused in the development,

requiring the creation of different installations for different versions. This issue could be eliminated by updating the nodes' software to compatible versions if allowed by the client. This could be done relatively quickly (around one day) without delaying the project. Otherwise, different installation processes should be created for the outliers. However, if the variety of versions is too large, the project may become less feasible, as each version might require up to a day of work.

3.2.4 Specific Hardware Used

Ruisdael uses a variety of weather observation stations to complete the client's research. These stations and the data produced by them can differ. Finding anomalies within the data is one of the project's requirements. Access to this data and possibly the machines may be needed to fulfil this requirement. Insufficient data can lead to an unreliable anomaly detection model. This problem could be mitigated by requesting access to more weather stations and examining the data they produce.

3.2.5 Misunderstood Documentation

This issue can arise on both the client and the developer side. The developers may have trouble understanding the frameworks they choose to use if the description is vague or insufficient. On the other hand, the client requires extensive documentation from the developers on their product. Different views on the sufficiency of documentation may raise problems during the development process. These issues can be alleviated by searching for other sources of information for the developers and discussing the sufficiency of product documentation with the client.

3.2.6 Consistency of Provided Virtual Machines

The VMs (Virtual Machines) will provide access to the server and will be a point where the data from individual weather machines is sent. A problem may arise if there is an attempt to transfer data to a server that is not available. This could result in lost data. There is a requirement from the client that the data should only be sent when the server is available. Realizing this during development would mitigate this risk.

4 Requirements

In the previous chapter the feasibility & risk analysis of creating the system requested by the user is discussed. This chapter aims to clearly present all the requirements requested by the client. Section 4.1 gives an overview of all the functional requirements issued by the client, whereas section 4.2 covers all the non-functional requirements which the system should have.

4.1 Functional Requirements

4.1.1 Must Have

1. As a user I will be able to check whether a node is connected to the network.
2. As a user I will be able to observer how much storage is used/free in each node connected to the network.
3. As a user I will have insight into the data throughput of a node, i.e., bandwidth used by each node to send its data to the server.
4. As a user I will be able to observe the RAM and CPU usage of each node in the network (performance).
5. As a user I will be able to run a script which installs the monitoring software on a node.
6. As a user I will be able to see all the aggregated data monitored at each node in the network using a global view dashboard comprised of graphs and charts.

4.1.2 Should Have

1. As a user I should be authorized before requesting data from a node.
2. As a user I should have access to a web-dashboard displaying a global graph network visualization containing further visual and pop-up information on each node according to the section 4.1.1.
3. As a user I should be able to use a the Kibana created web UI which will depict the entire network as a whole, represented by a graph, where I can select each node and inspect all the information provided by the monitoring system for that node, which is a requirement stated in Section 4.1.1.
4. As a user I should be able to monitor the amount of data being transferred from each node in the network to the server.
5. As a user I should be able to monitor the amount of data being collected by each node in the network.

4.1.3 Could Have

1. As a user I could be able to run a script which will install the monitoring system for the server.
2. As a user I could be able to receive notifications if certain weather data collected by the nodes falls within the 95% confidence interval of that same data.

4.1.4 Won't Have

1. As a user I won't be able to use the system to set up a VPN tunnel to the Ruisdael network.

4.2 Non-functional

1. The system must be reliable, such that the monitoring data obtained from any node in the network is received without errors.
2. The system must be scalable, such that more nodes/servers can easily be added to the network.
3. The system must be secure, such that third party users outside of Ruisdael cannot access the system or the network through the system.
4. The user manual for both the server and the nodes must be well documented, such that inexperienced users can also use the system.
5. The installation process of the system must be thoroughly documented and presented in a easy to understand way for inexperienced users.
6. The system must be created using an open source software stack, preferably the ELK Stack.
7. A testing environment which includes unit, integration, and requirements testing where each should yield at least 75% test coverage.
8. The system must run on Ubuntu 22.04.2 LTS for the server, and on Debian GNU/Linux 11 (bullseye) for the nodes.

5 Project Approach

The previously mentioned requirements are expected to be implemented using a combination of approaches mentioned below. The overall solution can be modeled as a server-client architecture in which the server is tasked with asking the clients (in our case the raspberry pis) for their data. In doing so, necessary measurement data and other necessary node data such as free storage and online/offline status can be obtained. Some statistical outlier data analysis can also be performed on either the server or the clients. Automation of installations should not demand much modeling and structural thinking as it is a case of generalizing scripts and factoring for differences. Extensive testing shall be carried with the aid of docker images in order to replicate the environment of raspberries. Any other more specific aspects of the solution are expected to be ad-hoc dirty solutions tailored to the situation.

To address both the general and specific sides we have established the importance of developing acquaintance with the individual systems, the network and the ELK stack in order to think in the most optimal perspective to be able

to devise solutions at all stages of the project. The choice of ELK stack seems most sound one backed by the findings of Swaminathan and Dharur (2018) Dharur and Swaminathan, 2018 for "its ease of usage especially to those in the field of security who may not possess a vast amount of technical expertise". A gantt chart capturing all the aforementioned points above in form of MoScoW requirements can be observed below:

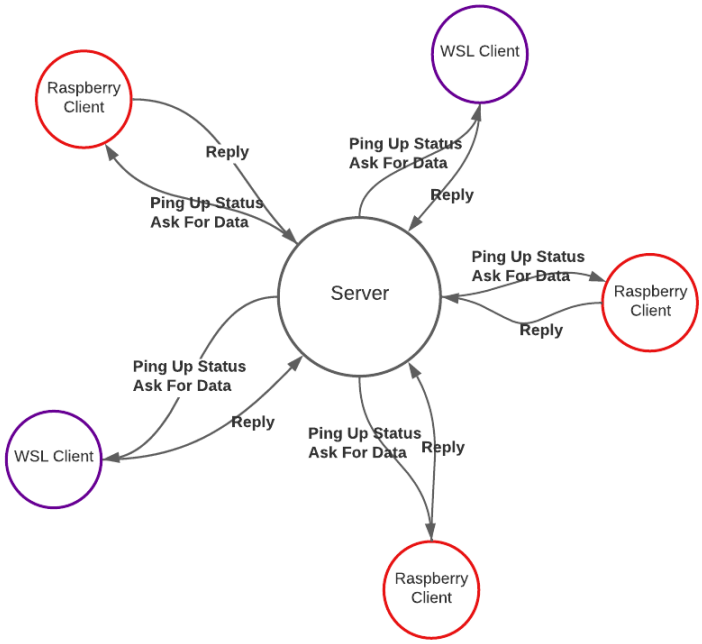

Figure 1: Planned Server-Client Architecture

6 Development Methodology

For the project to be completed in a structured manner set out in the previous chapter, a set of guidelines must be set out by the group, and self-enforced. This chapter aims to set out a clear set of rules and tools used for and in the project. In section 6.1 the chosen development approach will be explained and highlighted. In the next section, 6.2, a set of rules will be outlined for accepting new functionality. In section 6.3 the definition of when the project is finished is outlined. Continuing on, in section 6.4 a list of tools and technologies is presented. Finally, in section 6.5 our communication methods as a team are listed and their usage is explained.

6.1 Product Development Approach

We have identified two specific sub parts in our project. Specifically, a server that is responsible for the collecting and reporting of data, and a client (or node) that is able to send data to the server upon request. As an overarching approach, we will be using Scrum for the complete project. However, within sub-projects like the software we are developing for a node instance, we have determined that the use of the Waterfall approach is more suited. By using Scrum as the overarching development cycle, we can prioritize the features that are most valued at the time for the total product (Srivastava et al., 2017). Within the sub-projects (such as a node, or the server) we can then use the Waterfall approach to finish these features in full (Heriyanti and Ishak, 2020). This allows us to keep our progress between the server and client synced up and not deliver half-finished features with (for example) the server not having an

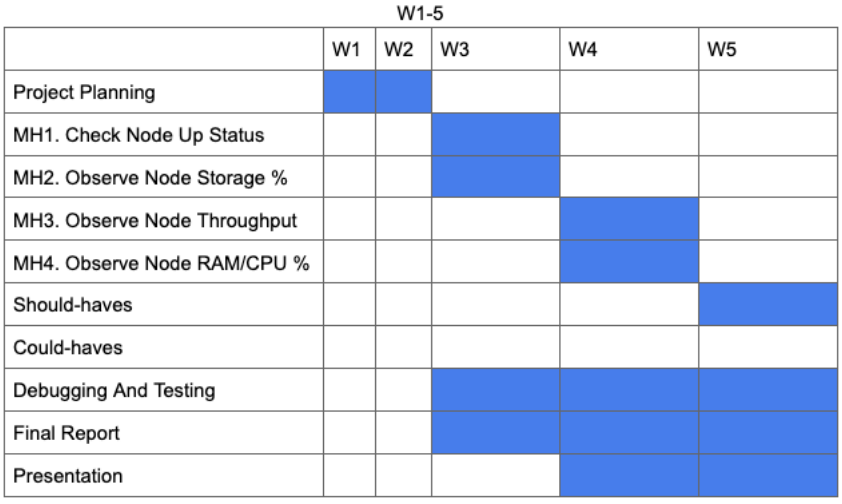

Figure 2: Gantt chart corresponding to weeks 1 to week 5

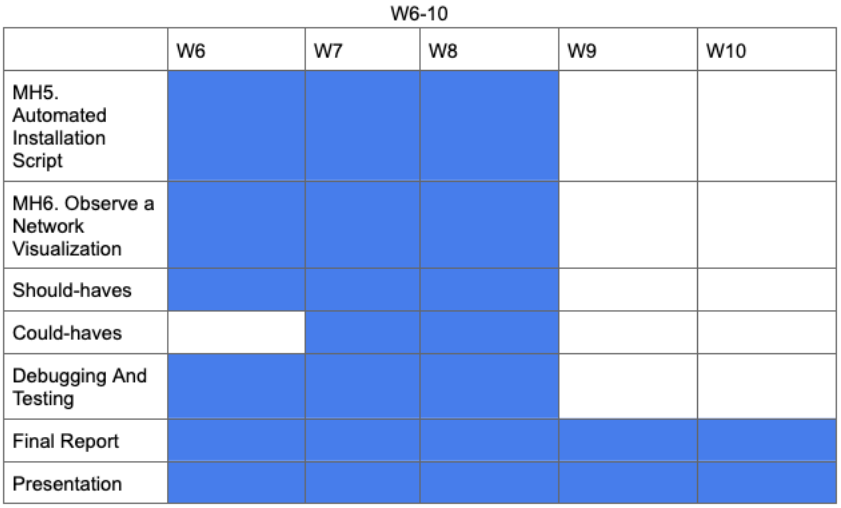

Figure 3: Gantt chart corresponding to weeks 6 to week 10

implementation for data a client can already send, while keeping the iterative approach of Scrum.

6.2 Requirements for New Functionality

After confirming the initial requirements with our client, Ruisdael, we have decided to only accept new functionality if the following criteria are met.

Time Constraints If we suspect this new functionality requires more hours of work then is reasonable for us to spend on the project, or if it will overrun deadlines in our development cycle as the project is coming to an end we will reject the additional requirements as out of scope. We can determine this by looking at the amount of hours we have left to spend on the project, and the amount of hours we expect to spend on the new functionality by comparing it to the amount of hours we have spent on similar functionality in the past.

Feasibility If we determine that the requested new functionality is beyond our current knowledge or ability, we will first have to determine if we are able to learn how to approach this new problem. This can be either through documentation, examples of existing programs or an expert outside of the group among other things. If we are not able to find sufficient sources, we may reject the new requirement based on lack of knowledge/resources.

After considering these two possible reasons for rejection, the new requirement is then assigned a priority according to the MoSCoW model, and will then be completed if or when possible.

6.3 Definition of Done

The project is considered "done" by us when the client side software can be installed without us intervening in the process. This is to say, we can run an install script and do not need to manually add components, update older versions of already installed software or manually take over the install process. The data and monitoring that this provides should then be visible in a web-based dashboard in real time. Finally, the installation process must be documented to such a degree that another party is able to take over the project and make change to it, without us having to explain our design to them. When this is possible, and at least the must haves of our MoSCoW requirements are met, we consider the project to be "done". This however does not take away that we will continue to work on the project, and add more features if possible.

6.4 Usage of existing Tools

As a group there are a few tools we have already identified to be essential to the development of our project, but as the project progresses more may be added.

Gitlab GitLab is a Git repository manager, which in itself provides version control, a CI/CD pipeline and collaboration features for software development teams. We will use this to store our project code.

Overleaf Overleaf is a web based collaboration tool to write and edit LaTeX documents, such as this report.

Python or NodeJS These two programming languages will be used to develop the client side software.

PIP or NPM These package managers are essential in addition to the already mentioned programming languages, and will significantly cut down complexity of installing software.

Elastic ELK Stack The ELK Stack is an open source set of programs designed for log management and analysis, and also allows for the visualization of data.

Kibana Kibana is an open source visualization platform that allows users to create dynamic dashboards from almost any data source.

Ansible Ansible allows users to automate IT tasks such as application deployment and configuration management, which will allow us to create an install script.

6.5 Communication as a Team

Within our group we are mainly communicating over WhatsApp and Discord. The former, WhatsApp, being mostly used to discuss scheduling with external

parties and alerting other group members of incoming messages that are addressed to the entire group. The latter, Discord, is mostly used to send files, keep track of meeting notes and keep an overview of our project planning and task division regarding assignments for ourselves. In addition, task division for the project based on the requirements is done on Gitlab using the Issue Tracker. External parties, such as our Project Supervisor, or the client are contacted via E-Mail to set up meetings or discuss matters. An exception to this is our TA, which we communicate with using Mattermost, a service set up and maintained by TU Delft.

Bibliography

Dharur, S., & Swaminathan, K. (2018). Efficient surveillance and monitoring using the elk stack for iot powered smart buildings, 700–705. https://doi.org/10.1109/ICISC.2018.8398888

Heriyanti, F., & Ishak, A. (2020). Design of logistics information system in the finished product warehouse with the waterfall method: Review literature. *IOP Conference Series: Materials Science and Engineering*, *801*(1), 012100.

Hitchcock, K. (2020). *Linux system administration for the 2020s*. Apress, Berkeley, CA. https://link-springer-com.tudelft.idm.oclc.org/chapter/10.1007/978-1-4842-7984-7_7

Igor Anastasov, D. D. (2014). Siem implementation for global and distributed environments. *2014 World Congress on Computer Applications and Information Systems (WCCAIS)*. https://doi.org/10.1109/WCCAIS.2014.6916651

Srivastava, A., Bhardwaj, S., & Saraswat, S. (2017). Scrum model for agile methodology. *2017 International Conference on Computing, Communication and Automation (ICCCA)*, 864–869. https://doi.org/10.1109/CCAA.2017.8229928