

BACHELOR THESIS

Transmission and Analysis of Vehicle Telemetry Data Using OBD-II and Cellular Networks

Submitted by Nicholas Walter

In fulfillment of the requirements for the degree
Bachelor of Science in Informatics

To be awarded by the
Fontys Hogeschool Techniek en Logistiek

Gifhorn, June 12, 2018

Information Page

Fontys Hogeschool Techniek en Logistiek
Postbus 141, 5900 AC Venlo

Bachelor Thesis

Name of student: Nicholas Walter
Student number: 2552736
Course: Informatics - Software Engineering
Period: 2018-02-15 - 2018-06-12

Company name: IAV
Address: Rockwellstraße 16
Postcode, City: 38518 Gifhorn
Country: Germany

Company coach: Dr. Arnd Eden
Email: `arnd.eden@iav.de`
University coach: Ferd van Odenhoven
Email: `f.vanodenhoven@fontys.nl`

Examinator: Jan Jacobs
External domain expert: O. van Roosmalen

Non-disclosure agreement: No

Summary

This report describes the project "Transmission and Analysis of Vehicle Telemetry Data Using OBD-II and Cellular Networks".

The project's aim was to improve the process of carrying out test runs with vehicles by implementing a software tool to gather vehicle telemetry data and global position and transmitting this data to a remote analysis server over encrypted mobile networks. Additionally, the possibility of carrying out live-analysis on the same data should be explored by means of a feasibility study.

Although the implementation of the transmission tool was completed otherwise without a flaw, an error was encountered with the usage of security certificates that are required for the encryption of the transmitted data. Because certificates cannot be loaded, connections cannot be encrypted. This error could not be fixed. Aside from two low-priority requirements, the software tool is otherwise feature-complete.

The feasibility study regarding live data analysis yielded the result that carrying out the live analysis on the target device is possible from a software point of view. However, it is expected that running both the transmission tool and live analysis on the same device would lead to performance issues. Furthermore not all required data may be easily obtainable.

The overall project was therefore not successful with respect to its originally formulated goals, although the created software product specifically can be completed with very little effort if a solution to the aforementioned problem can be found. This means that the results of the project can easily be used in further projects to fulfill their original purposes.

Declaration of Authorship

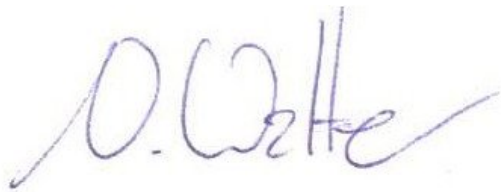
I, the undersigned, hereby certify that I have compiled and written this document and the underlying work / pieces of work without assistance from anyone except the specifically assigned academic supervisor. This work is solely my own, and I am solely responsible for the content, organization, and making of this document and the underlying work / pieces of work.

I hereby acknowledge that I have read the instructions for preparation and submission of documents / pieces of work provided by my course / my academic institution, and I understand that this document and the underlying pieces of work will not be accepted for evaluation or for the award of academic credits if it is determined that they have not been prepared in compliance with those instructions and this statement of authenticity.

I further certify that I did not commit plagiarism, did neither take over nor paraphrase (digital or printed, translated or original) material (e.g. ideas, data, pieces of text, figures, diagrams, tables, recordings, videos, code, ...) produced by others without correct and complete citation and correct and complete reference of the source(s). I understand that this document and the underlying work / pieces of work will not be accepted for evaluation or for the award of academic credits if it is determined that they embody plagiarism.

Name: **Nicholas Walter**
Student number: **2552736**
Place, Date: **Gifhorn, 2018-06-12**

Signature:

A handwritten signature in blue ink, appearing to read 'N. Walter', with a stylized flourish at the end.

Contents

| | |
|--|-------------|
| Summary | ii |
| Declaration of Authorship | iii |
| List of Figures | vii |
| List of Tables | viii |
| List of Abbreviations | ix |
| Glossary | x |
| 1 Introduction | 1 |
| 1.1 Context | 1 |
| 1.1.1 The Company | 1 |
| 1.1.2 The Problems | 1 |
| 1.1.3 Graduation Assignment | 2 |
| 1.1.4 Freematics ONE+ | 2 |
| 1.2 Report Structure | 2 |
| 2 Planning | 3 |
| 2.1 Deliverables | 3 |
| 2.2 Scope | 3 |
| 2.3 Software Development Framework | 4 |
| 2.4 Time Planning | 4 |
| 3 Feasibility Study | 5 |
| 3.1 Purpose | 5 |
| 3.2 Target System | 5 |
| 3.3 Research Questions and Methodology | 6 |
| 3.4 Results | 6 |
| 4 Analysis | 7 |
| 4.1 Stakeholder Analysis | 7 |
| 4.2 Risk Analysis | 8 |
| 4.3 Requirements Analysis | 9 |
| 4.3.1 Requirements Elicitation | 9 |
| 4.3.2 Requirements Evaluation | 9 |
| 5 Software Design | 11 |
| 5.1 Design Parameters | 11 |
| 5.1.1 Design Aims | 11 |
| 5.1.2 Design Constraints | 11 |
| 5.2 Basic Structure | 12 |

| | | |
|-----------|--|-----------|
| 5.2.1 | Setup and Loops | 12 |
| 5.2.2 | Modules | 12 |
| 5.3 | Module Design | 13 |
| 5.3.1 | Inter-Module Communication Design | 13 |
| 5.3.2 | DataKeeping Module | 14 |
| 5.3.3 | DataHandling Module | 16 |
| 5.3.4 | Time Keeping | 17 |
| 5.4 | Multi Threading Design | 18 |
| 5.5 | Test Design | 19 |
| 5.5.1 | Code Tests | 19 |
| 5.5.2 | System Tests | 19 |
| 6 | Quality Management | 20 |
| 6.1 | Quality Management Approach | 20 |
| 6.2 | Quality Control | 20 |
| 7 | Implementation | 21 |
| 7.1 | New Technologies | 21 |
| 7.1.1 | Serial Communication | 21 |
| 7.1.2 | Cellular Module Control: AT Commands | 21 |
| 7.2 | Challenges | 22 |
| 7.2.1 | Memory Management | 22 |
| 7.2.2 | AT Command Implementation | 22 |
| 7.2.3 | Task Timing | 24 |
| 7.2.4 | Certificate Installation | 26 |
| 7.2.5 | Certificate Usage | 26 |
| 7.3 | Result | 27 |
| 8 | Validation | 28 |
| 8.1 | Software Tests | 28 |
| 8.1.1 | Unit Tests | 28 |
| 8.1.2 | Integration Tests | 28 |
| 8.2 | Quality Validation | 29 |
| 8.3 | Requirement Comparison | 29 |
| 8.4 | Contradiction Between Results | 30 |
| 9 | Conclusion | 31 |
| 10 | Recommendations | 32 |
| | List of References | 33 |
| A | Project Plan | 35 |
| B | Stakeholder Analysis | 46 |
| C | Risk Analysis | 50 |
| D | Requirements Analysis | 56 |
| E | Quality Management Plan | 68 |
| F | Software Design Document | 79 |

| | |
|--------------------------------|------------|
| G Research Report | 96 |
| H Example JSON File | 107 |
| I Product Class Diagram | 108 |

List of Figures

| | | |
|------|---|-----|
| 4.1 | Stakeholder Power/ interest grid | 7 |
| 4.2 | Risk Matrix | 8 |
| 4.3 | Cost/ Value Graph of Project Requirements | 10 |
| 5.1 | Product Class Diagram: DataKeeping Module | 15 |
| 5.2 | Product Sequence Diagram: Encoding Data in JSON | 16 |
| 5.3 | Product Class Diagram: DataHandling Module | 17 |
| 5.4 | Product Sequence Diagram: Handling Data | 18 |
| 7.1 | Hardware/ Software Interactions on the Target Device | 24 |
| 7.2 | Thread Diagram | 25 |
| 8.1 | Calculated Score of Quality Metrics over Time | 29 |
| A.1 | Stakeholder Power/ interest grid | 41 |
| A.2 | Gantt Chart | 44 |
| B.1 | Stakeholder Power/ interest grid | 49 |
| C.1 | Risk Matrix | 54 |
| D.1 | Cost/ Value Graph of Project Requirements | 61 |
| F.1 | Product Domain Model | 83 |
| F.2 | Product Architecture | 84 |
| F.3 | Product Class Diagram: DataReading module | 86 |
| F.4 | Product Sequence Diagram: Process of reading data from sensors | 87 |
| F.5 | Product Class Diagram: DataKeeping module | 88 |
| F.6 | Product Sequence Diagram: Process of encoding data in a DataContainer to JSON | 88 |
| F.7 | Product Class Diagram: DataHandling module | 89 |
| F.8 | Product Sequence Diagram: Process of handling data contained in a DataContainer object. | 90 |
| F.9 | Product Class Diagram: ConnectionHandling module | 91 |
| F.10 | Product Class Diagram: Util module | 91 |
| F.11 | Product Class Diagram: Util module | 92 |
| F.12 | Product Class Diagram: Util module | 94 |
| I.1 | Product Sequence Diagram: Handling Data | 109 |

List of Tables

| | | |
|------|---|-----|
| 3.1 | Feasibility Core Question Results | 6 |
| 4.1 | Identified Risks | 8 |
| 4.2 | Project Core Requirements | 9 |
| 5.1 | Product Modules | 13 |
| A.1 | People relevant to the project | 42 |
| A.2 | Deliverables Deadlines | 43 |
| A.3 | Revision History | 45 |
| B.1 | Stakeholders | 48 |
| C.1 | Identified risks | 53 |
| C.2 | Risk Exposure | 54 |
| C.3 | Risk Handling | 55 |
| D.1 | Requirement F-Cellular | 61 |
| D.2 | Requirement F-OBd | 62 |
| D.3 | Requirement F-GPS | 62 |
| D.4 | Requirement F-JSON | 62 |
| D.5 | Requirement F-Time | 63 |
| D.6 | Requirement F-Configuration | 63 |
| D.7 | Requirement F-SD-Data | 63 |
| D.8 | Requirement F-SD-Data-Discard | 64 |
| D.9 | Requirement F-SD-Logging | 64 |
| D.10 | Requirement F-Bluetooth-Control | 64 |
| D.11 | Requirement NF-HTTPS | 65 |
| D.12 | Requirement NF-Performance | 65 |
| D.13 | Requirement NF-Extendibility | 65 |
| D.14 | Requirement NF-Flexibility | 66 |
| D.15 | Revision History | 67 |
| E.1 | Quality Target Metrics | 73 |
| E.2 | Quality Target Metrics Measurements: 2018-03-31 | 75 |
| E.3 | Quality Target Metrics Measurements: 2018-04-14 | 76 |
| E.4 | Quality Target Metrics Measurements: 2018-04-28 | 76 |
| E.5 | Quality Target Metrics Measurements: 2018-04-28 | 77 |
| E.6 | Revision History | 78 |
| G.1 | Data required to carry out live analysis | 100 |
| G.2 | Data required to carry out live analysis | 105 |

List of Abbreviations

| | |
|---------------|---|
| CAN | C ontroller A rea N etwork |
| GPS | G lobal P ositioning S ystem |
| HTTPS | H yper T ext T ransfer P rotocol S ecure |
| JSON | J ava S cript O bject N otation |
| NTP | N etwork T ime P rotocol |
| OBD-II | O n B oard D iagnostics V ersion 2 |
| RTOS | R eaL T ime O perating S ystem |
| SIM | S ubscriber I dentitY M odule |
| SSL | S ecure S ocket L ayer |
| TLS | T ransport L ayer S ecurity |

Glossary

| | |
|-----------------------------|---|
| Arduino | Open-source microcontroller , basis to many small electronics appliances |
| AT Commands | Also: Hayes Command Set. Command Language originally developed to control modem devices. Also used by the SIM5360 module to offer control to a user |
| CAN | Communications network to enable communication between microcontrollers without a central host computer |
| Cellular connection | For the context of this project, refers exclusively to an internet connection via mobile networks as used by cellphones. |
| Cellular module | In the context of this project, refers to an extension to the ESP32 chip with the capability of dialling into mobile networks to send and receive data from the internet (see SIM5360) |
| ESP32 | Arduino -like low cost microcontroller with integrated WLAN and Bluetooth connectors; although it is not actually related to Arduino chips, it is for the purposes of this project identical and may be referred to by this name |
| Freematics ONE+ | Freely programmable, open source OBD-II dongle based on the ESP32 microcontroller. Capable of gathering various data, equipped with cellular network module |
| FreeRTOS | An open-source RTOS designed for use on embedded devices such as microcontrollers like the ESP32 |
| GPS | A system to identify a device's current location on planet earth using triangulation of pings broadcast by satellites |
| HTTPS | Protocol to securely transfer data between a server and a client |
| JSON | A human- and machine-readable notation to encode arbitrary data |
| Micro-controller | A computer system based on a single chip, often equipped with various interfaces or I/O pins |
| NTP | Networking protocol to synchronize time between computer systems |
| OBD | Protocol and interface for communication between car and diagnostic hardware |
| RTOS | Operating System designed to handle data as it is created with little or no buffer in between. Used in time-critical applications such as safety measures in vehicles in order to ensure instant reaction |
| Serial Communication | Communication protocol in which data is sent bit by bit over a single wire/ connection as opposed to parallel communication which makes use of multiple wires/ connections to transmit several bits at once |
| SIM5360 | Cellular module attached to the ESP32 microcontroller at the core of the Freematics ONE+ . Offers connectivity to cellular networks for data transmission |

SSL / TLS

Set of cryptographic protocols to secure the communication between computer systems; used for encryption in **HTTPS**. Both terms are used interchangeably in this report

1 Introduction

This report was created during the project carried out as part of the bachelor thesis "Transmission and Analysis of Vehicle Telemetry Data Using OBD-II and Cellular Networks". It describes the results of both the software project and the accompanying research objective.

1.1 Context

In order to fully understand the project and its intended results, it is important to know the context in which it is being carried out: the company and the problems it is facing as well as a brief description of the tasks intended to solve this problem. This section provides that information, defines the graduation assignment and also briefly introduces the device that is a key part of the project.

1.1.1 The Company

The company IAV ("Ingenieurgesellschaft Auto und Verkehr") is a German automotive engineering company whose main focus is on the development and production of vehicle parts for several major car manufacturers and automotive component suppliers. The company was founded in 1983 in Berlin as a research institute attached to the Technical University of Berlin and has since grown large enough to have 6700 employees all over the world with a turnover of 734 million Euro in 2016 (see IAV, 2017).

1.1.2 The Problems

Because the testing of newly designed parts and software is a fundamental part of IAV's every-day business, it is important for them to streamline this process as much as possible. A large portion of testing involves the analysis of telemetry data generated by test vehicles in order to identify issues and their root causes as early as possible.

IAV is facing two problems in this context: The first is that considerable effort is required to set up and install the computer systems for this job. These systems gather data from the vehicle's onboard diagnostic (OBD) port and briefly process it before transmitting it to a remote server. Once data is received by this server it is analysed in more detail and stored for later use. The setup effort delays test drives which in turn causes development times to grow.

The second problem is that test drives are often defined by very detailed parameters, such as what type of roads to drive for what time and distance, what speed to drive, etc. A test driver can easily be overwhelmed by the number of parameters to keep an eye on so that they often do not notice when a test run becomes invalid. The continuation of invalid test runs does not yield usable results and is therefore an unnecessary expense that should be avoided.

1.1.3 Graduation Assignment

Based on the problems described in Section 1.1.2, the assignment for the duration of the graduation thesis is to optimise the process of carrying out test runs. Based on the assignment, two separate tasks were identified which will lead to the fulfilment of the project's aims:

First, the implementation of a software solution to run on a microcontroller-based telemetry device. The software's designated purpose is to read data from the OBD-II interface and global positioning system (GPS) sensor attached to the microcontroller and transmit it to a remote server for further analysis while encrypting all connections using Transport Layer Security (TLS) or its predecessor Secure Sockets Layer (SSL).

Second, based on the experience gathered and knowledge gained during the design and implementation of this software, a feasibility study should then answer the question whether the software tool can be extended to inform drivers about the status of a test drive's parameters.

1.1.4 Freematics ONE+

As described in Section 1.1.3, the project's primary aim is to implement software to run on a device capable of reading and transmitting telemetry data. This device comes in the form of the Freematics ONE+, a device based around the ESP32 microcontroller. It provides an OBD-II connector, a GPS sensor and a cellular module which uses a subscriber identity module (SIM) so that data can be read and transmitted. This device was selected by IAV before the start of the project over other candidates because it was the only candidate that is freely programmable and offers the desired functionality (Eden, 2018).

1.2 Report Structure

This report is divided into nine chapters including the introduction. The appendix contains documents which expand on some of the information summarised in the main content chapters.

- **Planning:** Summarises the project planning
- **Feasibility Study:** Describes work on and results of the feasibility study
- **Analysis:** Explains the results of the analysis phase by its individual products
- **Software Design:** Explains the process and decision making behind the creation of the software design based on the results of the analysis phase
- **Quality Management:** Summarises how the product's quality can be measured and kept at the desired level
- **Implementation:** Describes the technologies used during the implementation phase as well as the challenges that were faced and how they were overcome
- **Validation:** Summarises the results of all software and system tests as well as the final quality measurements
- **Conclusion:** Analyses the results of all phases, comparing them to the original task and drawing a conclusion
- **Recommendations:** Offers advice on how to work with the results of the project as well as how to improve or complete them.

2 Planning

Because of the project's limited duration, it was of vital importance to plan its execution in advance so that all of its objectives could be completed within the given time frame. For this reason, a project plan (see Appendix A Project Plan) was created in which the project was clearly defined and planned ahead of time as far as possible. The following subsections summarise the most important points made in that document.

2.1 Deliverables

Because the project was not defined in detail when it was started, the first step was to identify what deliverables the customer expected to see upon its conclusion. This enabled further, more detailed planning.

Within the project plan, two main deliverables were identified based on the customer's initial wishes:

The first deliverable was a piece of software to run on the Freematics ONE+ that should read telemetry data from a vehicle and transmit it to a remote analysis server. Attached to this product, a number of documents describing and defining it would be created, including this report:

- Project Plan
- Stakeholder Analysis
- Requirements Analysis
- Quality Management Plan
- Software Design Document
- Intermediate Progress Report
- Final Project report (in form of Thesis)

The second deliverable was a report on a feasibility study carried out on the topic of whether the same device could be used to avoid the continuation of invalid test runs by notifying drivers immediately when a test parameter can no longer be met.

Both deliverables are described in more detail in Appendix A.3.2.

2.2 Scope

In order to make sure that the project's requirements would be met to the farthest extend possible, it was important to clearly define its scope as early as possible to prevent bloating and feature-creep before they occurred. For this reason, the project plan clearly defined what

is included in and what is excluded from the project's scope as well as core scope definitions for the related research report (see Appendix G Research Report).

Most importantly, the scope included carrying out four of the five main software project phases (Analysis, Design, Implementation and Validation) while excluding the fifth (Maintenance) for the first core deliverable. It also defined that, while it could be considered if time constraints allowed it, an implementation of the software tool researched as part of the second core deliverable was not part of the scope.

2.3 Software Development Framework

With the intention of ensuring that a high quality product would be delivered once the project came to its conclusion, the selection of the proper framework for the project was of vital importance. The most important criteria were to ensure that the product would be delivered on time and that its core features (see Section 2.1) would be fully implemented upon completion. With regard to people involved the project, the most important factor was that the customer was an experienced software developer and project manager and as such could be expected to be familiar with the way requirements elicitation, software design and other mechanisms in the domain of software development work.

With these criteria in mind, the decision fell on using the traditional waterfall model instead of an agile approach: Because of the customer's experience, the assumption could be made that requirements and wishes would not change significantly during the execution. This made it possible to carry out analyses and make design decisions at the beginning of the project without having to revisit them after each sprint.

However, regular customer communication was still deemed very important. Therefore, a single feature was taken from agile methods: A weekly meeting to discuss progress and problems was established.

2.4 Time Planning

Based on the decisions made and knowledge gained during the clarification of scope (see Section 2.2) and the selection of the software development framework (see Section 2.3), work on all tasks was planned in advance at the start of the project. As described in the project plan (see Appendix A Project Plan), start and finish dates for each of the project's segments as well as deadlines for deliverables were defined. Furthermore, a Gantt chart (see Figure A.2) was created for to provide an easy overview over current and upcoming tasks.

3 Feasibility Study

This chapter explains the research carried out during the feasibility study as described in Section 1.1.3 by defining the core objective and questions as well as explaining the methodologies used and the results obtained. The full research report is available at Appendix G.

3.1 Purpose

As outlined in Section 1.1.3, the feasibility study is based on the second problem described in Section 1.1.2: Because of very specific and precise parameters that define product test drives, drivers are often unable to keep track of their performance and only find out that their tests were invalid once they are completed. This causes unnecessary expenses that should be avoided.

The general idea is that, using the same device that is already being used in the main project, the gathering and transmission of vehicle data, could be used to automatically keep track of parameters and warn the driver if a breach is imminent or completed. This would mean that test drives could be aborted early, effectively reducing costs. The feasibility study's aim is to research whether or not carrying out this process is possible on the Freematics ONE+ with respect to hardware capabilities as well as the telemetry product in terms of software compatibility.

3.2 Target System

After the system's general purpose was identified, the next step was to describe it in more detail by defining its core tasks as well as restrictions that would apply to it.

The following core tasks were identified:

- **Data Reading:** Gathering of data from various sources such as the device's GPS sensor, OBD interface and online sources
- **Data Tracking:** Calculating information from read values, keeping track of minimum/maximum values, etc.
- **Data Comparison:** Comparing gathered and calculated information to target values in order to identify when parameters are fulfilled or breached
- **User Interaction:** In case a parameter's value differs from target values, the test driver should be warned about this so that the test drive can be aborted

Additionally, the system's design would have to take two main restrictions into account: Firstly, because it's intended target environment is a moving vehicle, the end user will be unable to interact with the device frequently for safety reasons. Secondly, the device's hardware limitations have to be taken into account when answering any core questions.

3.3 Research Questions and Methodology

Based on the previously identified tasks and restrictions, the next step was to define the core questions to answer in order to decide whether or not implementing the system is feasible. The following core questions were identified:

1. Is it possible to gather all required data from either a sensor, an online API or another source?
2. Is the device capable of handling the acquired data in ways suitable to handle all tasks?
3. Is there a way in which a user can interact with the device?
4. Can the device carry out the required number of data reading / tracking actions?
5. Can the system be integrated into the data transmission project as a subsystem?

These questions were derived from customer interviews carried out previously as well as analysis of some usage examples given by the customer.

Since all questions are based on whether or not the device is capable of fulfilling a given functionality or providing some data, the research consists of comparing the device's capabilities according to its product page and specification sheets as well as experience gathered during the execution of the main project to what is required for the live analysis.

3.4 Results

Table 3.1 gives an overview over the core questions and their results. Some answers include remarks detailing the results.

TABLE 3.1: Feasibility Core Question Results

| Number | Answer | Remarks |
|--------|--------|---|
| 1 | Yes | Some data may take considerable effort to acquire |
| 2 | Yes | - |
| 3 | Yes | The user cannot directly communicate with the device, instead a bridge in form of e.g. a smartphone app is required |
| 4 | Yes | - |
| 5 | No | While the software implementation of adding the analysis to the main project is no issue, the device's hardware specifications are most likely insufficient |

This shows that all but one of the core questions could be answered affirmatively. The only answer that had to be answered with "No" was whether the proposed analysis system could be integrated into the existing telemetry transmission project's solution.

In conclusion this means that, although parts of the proposed system may take considerable effort to implement, it can be implemented as a stand-alone solution. The addition of this proposed system to the main project's software solution however is most likely infeasible due to performance limitations.

4 Analysis

After the initial project planning, the next step was to carry out the analyses that were defined as deliverables. The following chapters briefly summarise each of the documents describing the analysis results. In accordance with the pre-defined deliverables (see Section 2.1), the core milestones for this phase were the stakeholder analysis, the risk assessment and a requirements analysis.

4.1 Stakeholder Analysis

In order to identify and classify people and parties relevant to the project, a stakeholder analysis was carried out. This helped to identify and classify stakeholders, who have interest in and/ or power over the project and its outcomes so that it was clear how to handle each of them during the execution.

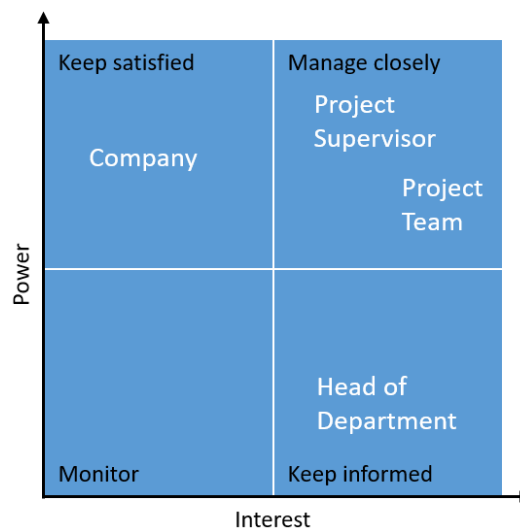


FIGURE 4.1: Stakeholder Power/ Interest Grid

After a number of techniques were used to identify stakeholders, their individual power over and interest in the project was plotted in a power/ interest grid. This made it easier to judge what the best course of action for handling each of the stakeholders would be over the course of the project. Figure 4.1 shows this grid. If the question of how a party should be handled during the project comes up, looking up their position in the grid gives a simple answer.

The full stakeholder analysis is described in Appendix B Stakeholder Analysis.

4.2 Risk Analysis

Because the project is of high importance, it was important that no major problems would impact it, threatening delays, suspensions or even a complete cancellation. In order to be as well-prepared as possible, a risk analysis was carried out early during the analysis phase. After risks were identified, each was individually analysed by estimating its likeliness of occurring as well as the strength of the impact it could possibly have on the project. From these two values, the total risk exposure was calculated. Table 4.1 describes the identified risks.

TABLE 4.1: Identified Risks

| # | Condition | Consequence |
|---|--|---|
| 1 | Insufficient time to finish project | Not all requirements can be re-alised |
| 2 | Device does not offer required functionality | Not all requirements can be re-alised |
| 3 | Additional requirements appear during project execution | More time is required to meet all requirements |
| 4 | A requirement takes longer to implement than anticipated | More time is required to meet all requirements |
| 5 | Hardware programming causes difficulties/ delays | Some core features may remain unfulfilled/ take longer so that other requirements cannot be worked on |

In order to identify large risks more easily, they were projected into a graph plotting their impact over their probability, see Figure 4.2. This way, highly dangerous risks could be identified at a glance. As the graph shows, all risk are located within the left column, meaning they are unlikely to occur. Although some risks could have a high impact on the project, none reside in to top-right quadrant with high risk and impact, meaning that the project is relatively safe.

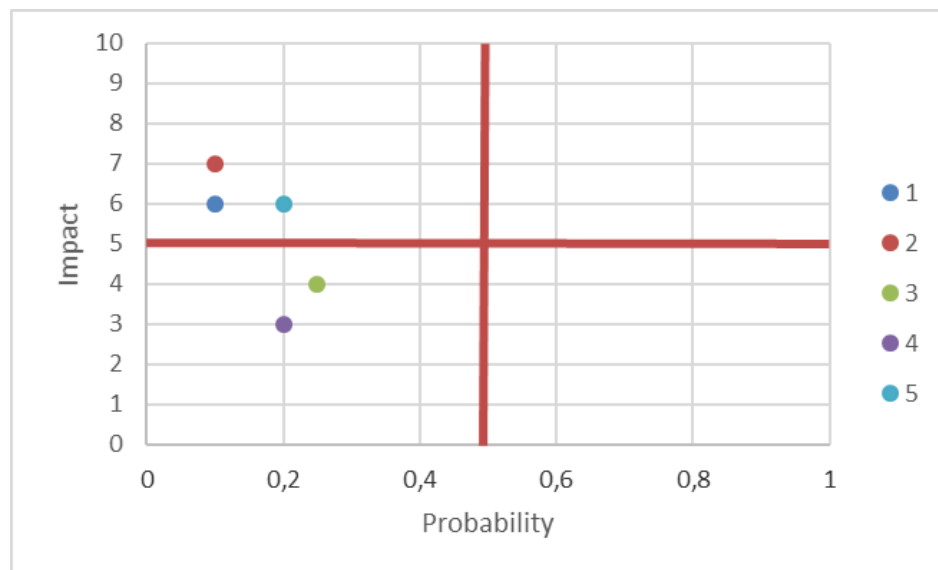


FIGURE 4.2: Risk Matrix

The final step was then to define mitigation actions, so that any potential impact would be less severe and contingency actions, so that the project can continue on its path without large

delays in case a risk does occur. Appendix C Risk Analysis describes the full risk analysis and its results.

4.3 Requirements Analysis

After stakeholders and risks were identified, the final analysis step was to identify and evaluate requirements. Because the basic premise of the project, the gathering and transmission of vehicle telemetry data, was already well-established, the requirements analysis focussed more on specific parameters to this task. This subsection briefly summarises the results of the full analysis requirements which can be read at Appendix D Requirements Analysis.

4.3.1 Requirements Elicitation

The first step of the requirements analysis was to identify requirements for further analysis. This was realised by combining a number of established techniques: At first, even though the project's basic intentions were well-known, they were confirmed by holding a short interview with the project spokesperson. After some time during which other project-related tasks such as the risk assessment were carried out, another interview session was held with the same spokesperson in order to identify more detailed requirements, such as how many times per second data should be gathered from the vehicle and sent to the remote server for analysis. Another example is the JavaScript Object Notation (JSON) specification that should be used for communication with the server. In Table 4.2, the project's four core requirements have been listed.

TABLE 4.2: Project Core Requirements

| Requirement ID | Description |
|----------------|---|
| F-Cellular | The device should use a mobile internet connection to transmit data to remote servers |
| F-OBd | The device should read all available data from the vehicle it is attached to |
| F-GPS | The device should gather GPS data and link it to other data |
| F-JSON | The device should encode all gathered data in JSON format, according to a specification |

In addition to these core requirements, several more functional and nonfunctional requirements were identified and described (see Appendix D.2).

4.3.2 Requirements Evaluation

After the requirements were identified, the next step was the evaluation of functional requirements based on value they would add to the project as well as the cost they would require to be realised. Because no objective numerical value could be identified for the requirements, their relative value was identified by the customer with the help of a value comparison matrix. This technique calculates the percentage of value each requirements adds to the project by calculating it from numerical statements such as "Requirement A is 5 times as important as Requirement B". Because of the project's small scale, the requirements' cost was expressed in the number of days of work they would require to be implemented.

In Figure 4.3, the results of this evaluation are visualised. The x-axis shows cost of implementation while the y-axis visualises added value for each requirement. By splitting the graph into three sections, requirements are grouped by high, medium and low priority.

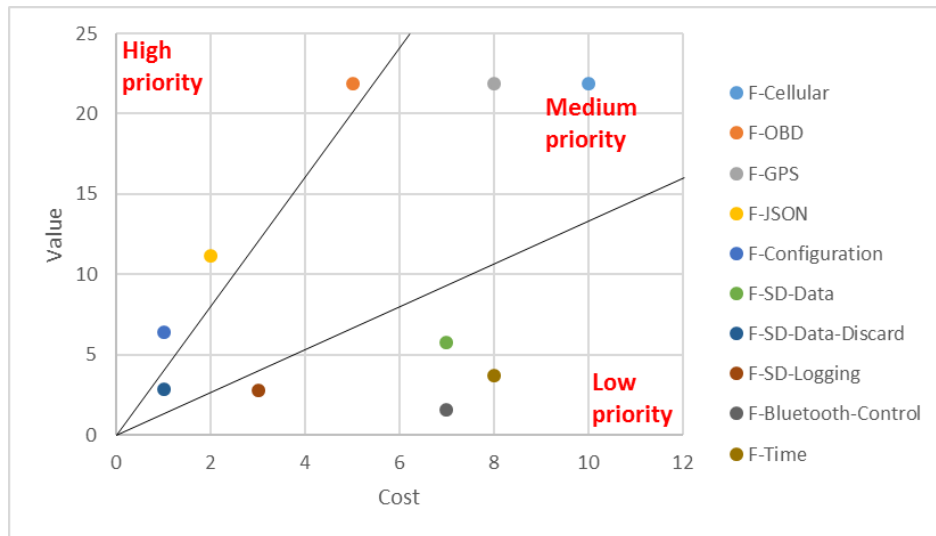


FIGURE 4.3: Cost/ Value Graph of Project Requirements

As explained in the requirements analysis document, the customer's decision was to implement the requirements F-Cellular, F-OBD, F-GPS and F-JSON before the others regardless of their priority group as they make up the project's core requirements. Furthermore, none of the other functional requirements were considered a core task, therefore the project would be considered as completed without them.

5 Software Design

The next step after completing the analysis steps described in the previous chapter was to design the software product based on the new information available. After gaining a rough understanding of the system's environment with the help of a domain model (see Figure F.1), this consisted of first deriving some core design aims from the non-functional requirements gathered as well as listing design constraints that would have to be taken into account. Afterwards, the actual software design was created in terms of basic structure of the solution: All functionality needed to be contained in classes and modules according to the design aims and constraints. Finally, a rough design concerning the distribution of tasks over the device's two processor cores was created. The following subsections explain these results and the reasoning behind them based on some core examples; the original document is available in Appendix F Software Design Document.

5.1 Design Parameters

As first part of creating a design for the software to be implemented, a number of core design aims were derived from the non-functional requirements identified during the requirements analysis. Additionally, design constraints were identified from hardware constraints. All design decisions then had to be made with these aims and constraints as a guideline.

5.1.1 Design Aims

- **Ease of implementation:** Where possible, design choices should be made that make the development and later maintenance of the software easier
- **Extendibility:** The software should be created with future expansions in mind. Specifically, additional sensors or methods of handling data could be added (see Table D.13)
- **Flexibility:** The software should be able to react to environment and system changes flexibly: For example, some data source may suddenly become unavailable or connection to cellular networks could be lost (see Table D.14)
- **Performance:** All functionality should be designed and implemented with the thought of achieving reading and transmission rates as high as possible (see Table D.12)

5.1.2 Design Constraints

While the design aims listed above were targets set for the design by the project team and its stakeholders, the design constraints described here are hard limits set by hardware limitations:

- **Available flash memory:** The ESP32 chip has 4 Megabytes of memory available for program space and 520 Kilobytes of RAM (see Freematics, No date(b))

- **Available processing power:** The ESP32 chip is based on a dual core processor that runs at a frequency of 240 MHz (see Freematics, No date(b))

5.2 Basic Structure

Based on the design aims and constraints identified in Section 5.1, the next step was to create a general overview over the product's basic structure. This subsection explains the reasoning behind the decisions made for this purpose.

5.2.1 Setup and Loops

Although similarly to desktop applications, the firmware running on the ESP32 chip is initialised with a `main()` function, the framework is designed so that user-created programs consist of two basic functions: The `setup()` function which is called once and the `loop()` function which is called in an infinite loop (see me-no-dev, 2017a). This is based on the design commonly used in Arduino systems (see Arduino, No date(b)) that makes it easy to separate setup tasks such as initialising serial communication or loading modules from the actual application logic.

This design however is based on the assumption that the program running on the chip only uses a single thread, most likely because the majority of Arduino devices do not offer multicore processing (see Arduino, No date(a)).

Because the ESP32 microcontroller this program will run on contains a dual core processor (see Espressif Systems, 2018), this basic structure needed to be adapted: Instead of running a single loop task, two functions (`readerLoop()` and `handlerLoop()`) will be running in infinite loops simultaneously on the two cores. This makes it possible to utilise all available computing power to the full extent, effectively increasing performance. The two tasks are then connected to each other by using a threadsafe queue as data buffer (see Figure I.1) into which the `readerLoop()` task can write data and the `handlerLoop()` task can extract data to work on from. This separation of tasks over two threads also had the advantage of preventing multiple accesses to the same resource.

5.2.2 Modules

With the intention of keeping in line with the design aims to keep the product easy to implement and maintain as well retaining the flexibility to react to environment or implementation changes, the decision was made to divide the system up into modules, each of which would have a clearly defined task to fulfill. An important aspect of this was that modules should be independent from each other so that changes in one would have as little effect on the others as possible.

This made sure that during development, unfinished modules could be simulated with dummy implementations implementing the same designed interface in order to test finished modules. Furthermore, implementation changes to any module will not affect other modules in future refactoring or maintenance work.

The decision was then made to create modules based on the basic tasks that need to be fulfilled in the system (see Table 5.1).

TABLE 5.1: Product Modules

| Module | Tasks |
|--------------------|--|
| DataReading | Gathering of data from OBD interface and GPS sensor |
| DataKeeping | Functionality to represent data and contain it in memory |
| DataHandling | Analysis and preparation of data for transmission |
| ConnectionHandling | Interfacing connection technologies (Cellular networks, WLAN, Bluetooth) as well as SD card writing |
| Logging | Centralized logging of notifications, warning and errors to various targets |
| Util | Contains configuration files, global constants, globally required functions as well as the functionality to keep system time updated |

5.3 Module Design

This subsection describes in more detail the choices made when designing the overall structure of the solution as well as two of the most important modules (**DataKeeping** and **DataHandling**). Design decisions behind the other modules can be read about in Appendix F Software Design Document ¹.

5.3.1 Inter-Module Communication Design

Since the basic premise of the design behind this product is that there are two tasks that take the role of producer and consumer (see subsection 5.2.1), it was necessary to outline a way in which these two tasks could communicate with each other to share data. A number of ways to implement this functionality were considered.

The first solution to this problem would have been to extend the **DataKeeping** module's functionality so that it would enable interthread communication, for example by means of synchronized collections which would contain data. In doing so, the **DataReading** and **DataHandling** modules would however be in danger of being affected by changes to the implementation of the **DataKeeping** module.

A second option would have been to design the **DataReading** module to create an interrupt to the microcontroller's processor upon which the **DataHandling** module would start its work.

Finally, the decision was made to use a global instance of a threadsafe queue provided by the real time operating system (RTOS) running on the chip to write data into and read data from as required. The modules would simply be instantiated with a reference to this queue; The **DataReading** module would then write into it while the **DataHandling** would periodically check for new data and handle it. This was considered the best solution because it would allow the two modules to operate completely separated from each other which guaranteed that neither would be influenced by changes to the other while also being the easiest to implement and maintain.

The class diagram in Figure I.1 visualises the connections between the modules via the RTOSQueue instance.

¹The (partial) class diagrams depicted on the following pages are not exhaustive; trivial members and relationships have been omitted in favour of readability. All UML diagrams were created in the style defined by Martin Fowler in his book "UML Distilled - Third Edition" (see Fowler, 2003). Colour filling is not according to UML standard but helps to improve readability.

5.3.2 DataKeeping Module

The **DataKeeping** module's purpose is to provide a framework to contain data in memory so that it can easily be shared between the two tasks.

Structural Design

Based on the aims and constraints identified earlier, the core focus behind the design of the **DataKeeping** module was to make sure that if any new sensors were added to the system at a later point, this could easily be done with minimal changes. Furthermore, the module should make sure that the system can continue working without impact if any data source is suddenly unavailable. Finally, based on requirement F-JSON (see Table 4.2), the module should also encode contained data to JSON format while keeping the design aim of simplicity in mind.

To begin with, a design had to be made which would mitigate issues originating from missing data.

The very first, and simplest, solution would have been to save all data in a single object in predetermined class members into which the **DataReading** module would write. By setting flags for the presence of each member or groups of members, the `getJSON()` method could have skipped those with no or invalid data, creating a valid JSON string. This idea was not implemented because it would have resulted in a single bloated class which would have made it difficult to maintain the module and adapt it to system changes.

The next iteration of the design was to implement a class for each type of data (GPS, OBD, etc.). The reading module would then create these and push them into the **RTOSQueue** for the **DataHandling** module to handle. This design had the advantage that maintenance was simplified in comparison to the first version: If a new data source was added, a new class could simply be added. Furthermore, if the relevant data source became unavailable, the object could simply not be pushed to the queue, eliminating the need to check presence and validity on the consumer side. However, it had one big flaw: Because the client task would be receiving objects one by one, it was forced to handle them individually. This meant that instead of only handling one object per reading cycle, it would handle as many objects as data sources were implemented. This was assumed to cause performance problems. Furthermore the object's type would be unknown which would go against C++ type safety rules.

Finally the decision was made to create a more complex solution to these problems:

By creating an **AbstractDataContainer** interface which classes like **OBDDataContainer** and **GPSPDataContainer** can then implement, the **DataContainer** class can aggregate references to them using their supertype instead of knowing about them by concrete references to their implementation. This way, when the `getJSON()` method is called, the **DataContainer** class can iterate over this list and combine the JSON strings obtained from each container to a full string. Because each JSON string is valid on its own, no issues arise if the containers for one or more data sources are not present at any given time.

This also means that a new data source can be represented in the **DataKeeping** module simply by implementing the corresponding **AbstractDataContainer** subtype.

Another design choice made for this module was to implement the Composite design pattern as described by the well-known "Gang of Four" (see Gamma et al., 1994): By making the **DataContainer** class implement the **AbstractDataContainer** interface and simultaneously

aggregate instances of the same interface's subtypes, it becomes the component object in the pattern while the other subtypes represent leaf nodes.

This has the advantage that if, for any reason, the handling of data should take longer than normally expected, data reading can continue by appending **DataContainer** objects to the queue described in subsection 5.3.1 while the **DataHandling** module recovers. Once it is back up to full speed, it can then read all available **DataContainer** objects from the queue at once and add them to a new encapsulating **DataContainer** object from which the JSON string for all contained data can then be obtained and handled at once.

Figure 5.1 visualises the final result of the design made for the **DataKeeping** module.

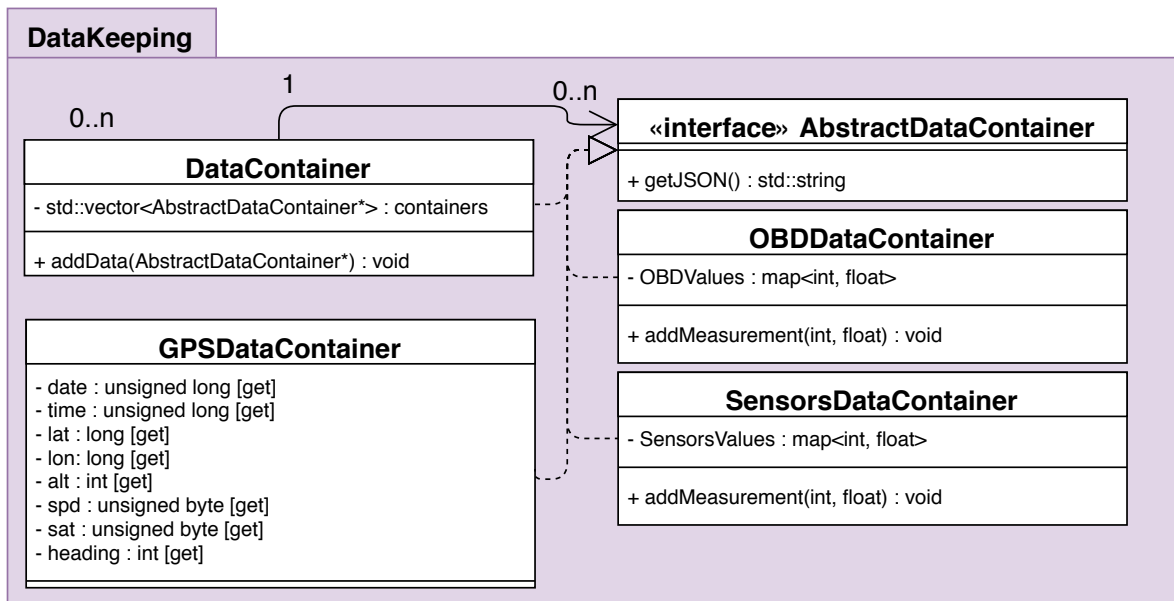


FIGURE 5.1: Product Class Diagram: DataKeeping Module

Behavioural Design

In order to fulfill requirement F-JSON, the system needs to be able to encode all gathered data into the JSON format. As described in subsection 5.3.2, this falls into the **DataKeeping** module's area of responsibility.

Upon calling of the **getJSON()** method on a **DataContainer** object, a new string which will contain the result is created. Afterwards the object iterates over its aggregation of **AbstractDataContainer** references: Because the module is implemented in the composite pattern (see subsection 5.3.2) each can either be another **DataContainer** object or an object containing concrete data such as an **OBDDataContainer**. By calling the **getJSON()** on either, a string is returned representing a data set. This is then appended to the result string. Finally, the result string is returned as visualised by Figure 5.2.

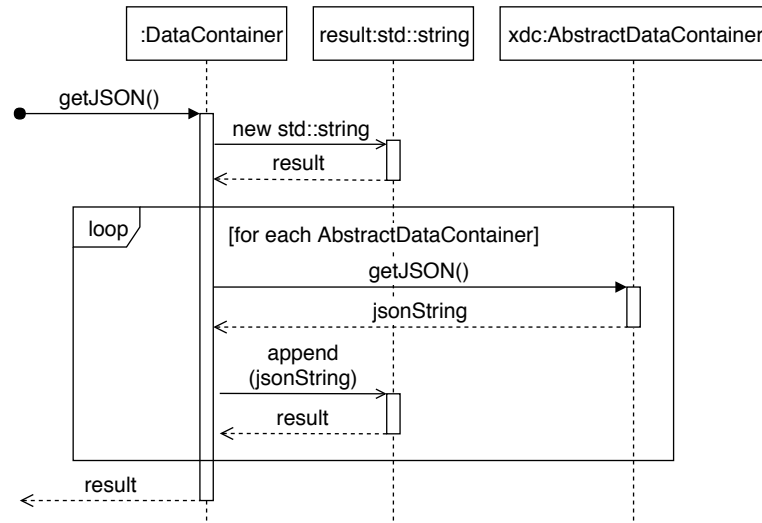


FIGURE 5.2: Product Sequence Diagram: Encoding Data in JSON

5.3.3 DataHandling Module

The **DataHandling** module's purpose is to read data from the **RTOSQueue** instance and operate on it in different ways, such as analysing it or transmitting it to a remote server.

Structural Design

For this module, the most important design aims were extendibility and flexibility as well as performance: New handling methods should be easy to add to the system, but the system should also be able to deal with the sudden unavailability of one or more handling methods if, for example, the connection to cellular networks was lost. All this should be possible while fulfilling requirement NF-Performance. Finally, the module should provide a simple interface for the `handlerLoop()` main function to call.

The first decision that had to be made based on these criteria was how to implement the process of actual data handling: While it would have been possible to implement everything in a single method in one class, creating only a very small memory footprint and a simple interface, maintenance in this design would have taken considerable effort.

Another possible solution that was considered was to implement individual classes for each type of handling so that a client could call them in sequence. However, this did not fulfill the design aim of providing a simple interface.

The final version of the design consisted of the **DataHandlerFacade** class which would be the only part of the module a client would interact with: Upon calling the `handledata()` method, the facade would iterate over its internal list of data handlers. By creating the **DataHandler** interface and making all concrete handlers implement it, the facade can iterate over a collection of these without knowing their concrete implementation and supply the **DataContainer** object it received to them. The data handlers can then use the data in a fashion most fitting to their purpose: For example, the **TransmissionDataHandler** only generates the JSON string representing the data and forwards it to the **CellularHandler** for transmission while the **AnalysisDataHandler** cares about the actual data itself and analyses it.

This design also made it possible to deal with unavailable handling targets: If one of the handlers was unable to work for any reason, the `DataReaderFacade` could simply skip over it.

Figure 5.3 visualises the `DataHandling` module’s final design.

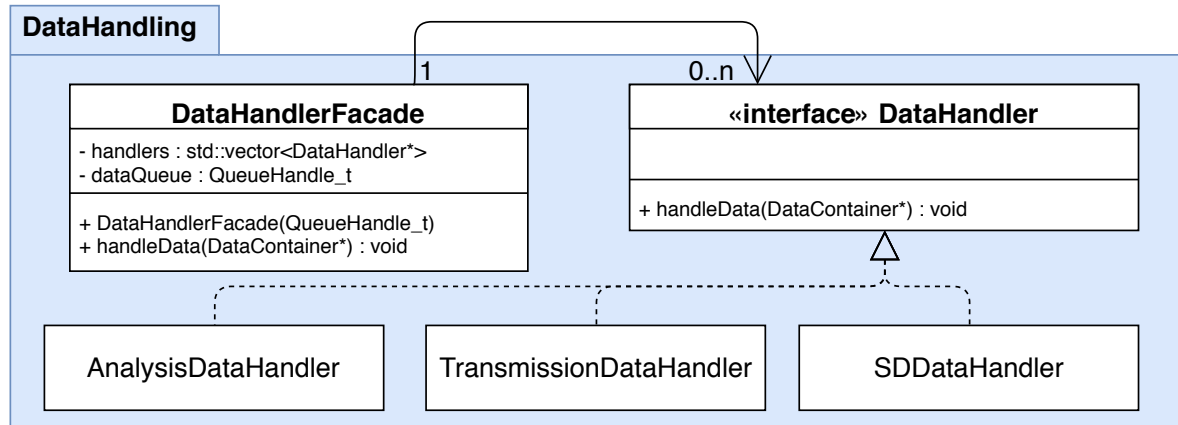


FIGURE 5.3: Product Class Diagram: `DataHandling` Module

Behavioural Design

The last step in the line of actions the system carries out is the handling of data. The handling tasks calls the `watchQueue()` method on the `DataHandlerFacade` object with every iteration. The object then checks whether any `DataContainer` objects are available in the `RTOSQueue` and receives them. It then iterates over its list of `DataHandler` subtypes, calling the `handleData()` method on them with the received `DataContainer` as parameter. The `TransmissionDataHandler` for example will then call the `handleJSON()` method on the `CellularHandler`. Finally, the `DataContainer` object is deleted and the process is finished.

Figure 5.4 shows the handling process using the `TransmissionDataHandler` as an example.

5.3.4 Time Keeping

Although it is not one of the core modules, the decisions behind the design of the time keeping functionality integrated into the system were vital to its success.

Because it is based on a microcontroller with very minimal equipment, the device the product will be deployed to does not have readily available functionality to get the current time and is also unable to keep this time up to date between power cycles. However, both the JSON format transmitted to the remote analysis server and for better logging, timestamps are essential. For this reason, the decision was made to add functionality to the system which should acquire current time at startup and keep track of it using the microprocessors clock cycle count.

At this point, the decision had to be made what time source should be used to acquire the initial startup time. The choices were either to use the built-in GPS antenna to acquire time over GPS which has an accuracy of around 100 nanoseconds (see Dana and Penrod, 1990) relative to UTC or to use the cellular module to acquire time over the network time protocol (NTP) which uses remote servers to get time information. Depending on network speeds and latency, it can be accurate to around 50 milliseconds (see Windl et al., 2006) to the time source server.

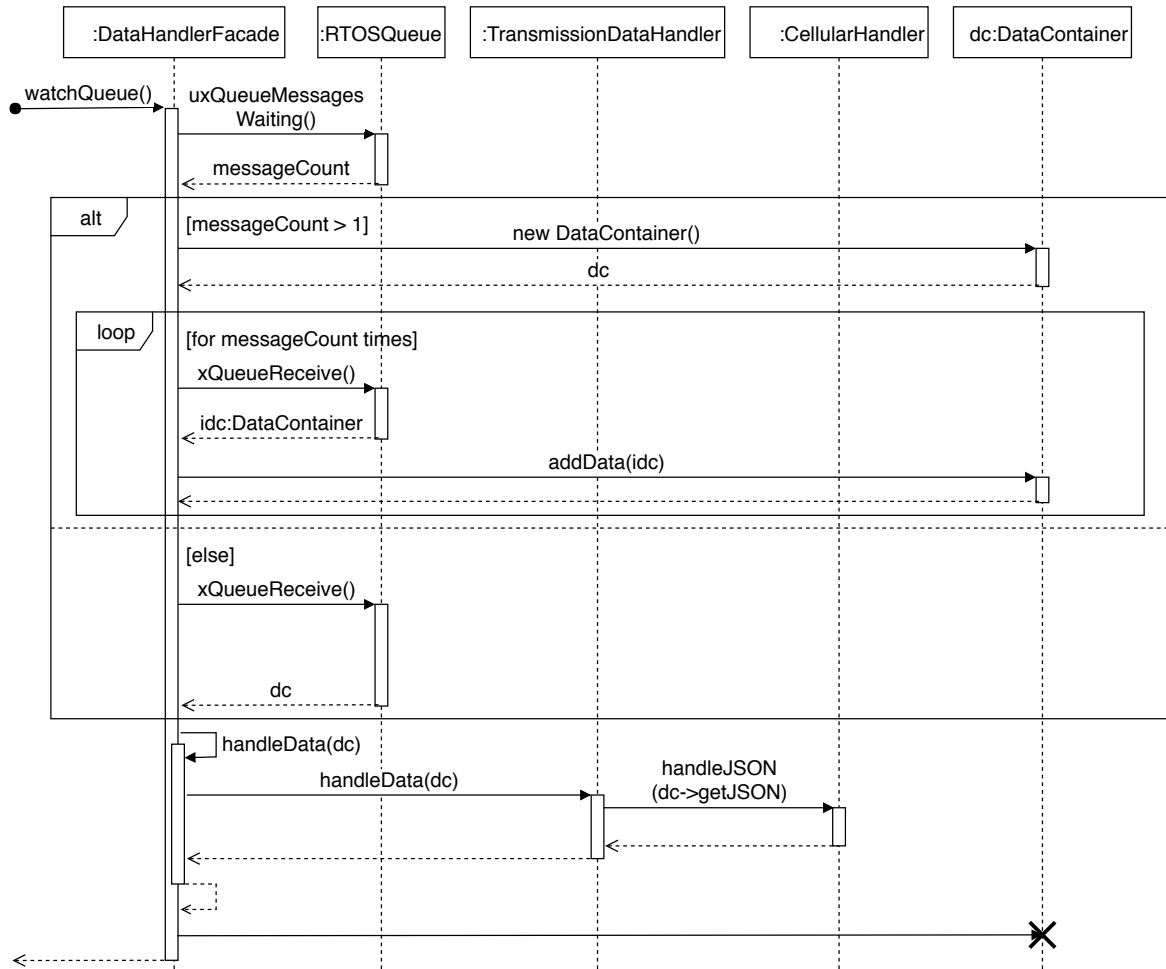


FIGURE 5.4: Product Sequence Diagram: Handling Data

Since an accuracy of 50 milliseconds is sufficient for this purpose and because the cellular module is more likely to be able to receive a signal at time of startup than the GPS antenna for a number of reasons², the decision was made to use NTP to acquire the initial time upon setup.

5.4 Multi Threading Design

As described in Section 5.2.1, the products main functionality of reading and transmitting data was designed to be separated over the microcontroller's two processing cores: This separation was decided upon in order to fully utilise all available computing power offered by the microcontroller.

The decision that had to be made was into how many threads each task should be divided. The choices were to either create only two tasks (one for reading data and one for handling data) or to also split up the handling of data into individual threads for each target: for

²Because the device is only equipped with a very weak GPS antenna, it needs good exposure to the sky in order to work properly (see Huang, 2017). While experiments show that placing it behind a car's windshield works fine, the customer expects people to forget about this causing issues when attempting to get current time.

example, one thread might be responsible for transmitting data over cellular networks while another would handle writing data to an SD card.

Since the basic structure of the program had already been decided on, the main considerations for this decision were the (positive or negative) impact on performance as well as the ease of implementation and extendibility. The final decision was to only use two tasks in this initial version of the product because the performance improvement to be gained from splitting up the handling task further was estimated to be negligible as still only one of these tasks could be executed simultaneously to the reading task. With that in mind, the decision was also made to keep a possible future change to this process in mind during the implementation phase and, where possible, facilitate the later refactoring.

5.5 Test Design

The final step of the design phase was to decide on a strategy to test implemented results. This included code tests with the intention of verifying that all code segments worked as intended as well as system tests that should verify that all code segments work together well.

5.5.1 Code Tests

The first decision that had to be made in this context was how individual code segments should be tested. An important aspect to the decision that was eventually made is the fact that large portions of the project's functionality depend on hardware interactions such as reading and transmitting data. While these functionalities could be mocked when testing other modules depending on them, they themselves would be difficult or even impossible to test in the form of unit tests.

For this reason, the decision was made that, where feasible within a reasonable time frame, all functionality should be tested with the help of unit tests that could be run whenever the implementation behind their targets was changed or updated. This also allowed for test-driven development to be used during the implementation phase: By creating tests that define the desired functionality of functions or methods before their real implementation, the target is more clearly defined and can more easily be worked towards.

All other parts of the product, that could not be tested with unit tests, such as reading data from the OBD interface, should be verified using manual tests such as comparing the output to information gathered from the device's environment (e.g. reading speed from the device and comparing it to the vehicle's tachometer).

5.5.2 System Tests

After code tests were decided upon, the next step was to design how the full system should be tested after its completion: This was important to make sure that no memory leaks or errors in the interaction between modules had been introduced. Based on time and resource constraints, no full testing suite could be obtained or implemented for this purpose, therefore the decision was made to carry out integration tests manually: In these, the device should be used as it would in its intended environment while logging errors and warnings. All logs could then later be analysed for problems.

6 Quality Management

This chapter describes the steps taken and actions defined in order to maintain high quality standards throughout the project. The full quality management document describing every part in more detail is available in Appendix E. The document was loosely based on the Standard for Software Quality Assurance Plans published by the Institute of Electrical and Electronics Engineers (see IEEE, 1998). A full implementation of the standard was not considered to result in a net benefit to the project with respect to time and quality standards due to the project's small scope and limited time frame. In order to achieve and maintain a high standard of quality in the project's results, several tasks had to be fulfilled. The sections below list these tasks and their results.

6.1 Quality Management Approach

First, standards and practices to follow during the implementation had to be defined. This consisted of standards to adhere to when writing source code (see Appendix E.3.1) and practices such as guidelines regarding the usage of version control and testing (see Appendix E.3.2).

Since the project's scope and time frame are limited, the selection criteria for all standards and practices were very simple: If a standard or practice already existed in the company, such as using the company-internal git repositories for version control, it was taken over. If none existed yet, the newly designed standards should be simple and easy to follow in order avoid unnecessary complications in development.

6.2 Quality Control

As a second step, some key metrics were to defined in order to make the quantification of quality possible:

- Time required for one iteration of reading data
- Time required for one iteration of handling data
- Time required by the system for startup and initialisation

For each metric, an acceptable value as well as a target value were defined. By calculating a normalised score from the acceptable and target values as well as measured values with the help of two formulas, an easy overview over the system's performance could be gained.

The decision was made to take a measurement of the system's performance bi-weekly during which improvements or declines in the quality score could be identified and analysed. Furthermore, the development of all scores over the course of the project should be visualised at the end of the project.

7 Implementation

This chapter summarises the work carried out during the implementation phase of the project. Since all implementation is based on the designs made earlier (see Chapter 5), most of the work in this phase was very straightforward. The following subsections give a brief introduction to the technologies used in the product and also describe challenges that were encountered and how they were overcome or worked around.

7.1 New Technologies

Although it was possible to apply known techniques during the analysis and design phases, for the implementation phase, a number of new and unfamiliar technologies and concepts had to be researched and understood. This section lists and explains them.

7.1.1 Serial Communication

Because microcontrollers are usually limited with respect to both computing power and available interfaces, the method of choice for communication with external devices is often a simple serial interface: While normal desktop applications could make use of e.g. network sockets to transmit data or commands to external clients, this method is unavailable on most microcontrollers and very impracticable in the specific case of the ESP32. For reasons of cost and simplicity, serial communication is often preferred for microcontrollers (see Jimb0, No date).

By transmitting bytes between the two connected devices, commands or data can be exchanged. In this project specifically, it was used to transmit status and debug messages from the ESP32 to any external client and to push commands to the cellular module.

7.1.2 Cellular Module Control: AT Commands

As one of the core requirements for the product was to make use of the cellular module attached to the microcontroller at the core of the telemetry device to transmit data, the control of this module was of high importance.

The control interface offered by the module are AT commands that are received over a serial interface. After submitting the character sequence `AT+` (short for "attention") to the module, it expects an incoming command and parameters¹. Afterwards, the actual command is sent (e.g. `CREG` for network registration). After the command itself has been transmitted, the execution is started with the transmission of a carriage return character. The client should then wait for a response from the module on whether or not the command was successful in order to decide upon how to proceed.

¹While commands with a different syntax exist, these are not relevant to this project and will therefore be disregarded.

In addition to these commands, small Lua scripts can be installed and run on the module. Although they offer the same functionality as AT commands², they also offer a fully fledged programming language instead of commands, making development easier. Because Lua scripts are independent from the microcontroller host, there is also no need to wait for answers between commands.

7.2 Challenges

Despite the fact that careful consideration went into the planning and design of the product, some issues were identified during the implementation phase. The following subsections describe the problem behind each of them and how they were solved or worked around.

7.2.1 Memory Management

Although it was clear from the beginning that hardware limitations would have an influence on the project's design and implementation (see Section 5.1), the full extent of limitations only became clear at a later point. With all libraries and code loaded, the free heap size at the start of setup was around 215 KB. After both tasks with a stack size of 15 KB each were created and all modules loaded, only around 170 KB remained available for data, although the decision was made to further restrict it to 150 KB in order to account for memory fragmentation and miscalculations.

Furthermore, experiments showed that with data for all 196 OBD IDs and a full set of GPS data contained in it, a **DataContainer** object takes up slightly more than 8.1 KB, meaning that a maximum of 18 could exist in the assigned memory block before the device would crash or run into undefined behaviour. Because the **RTOSQueue** responsible for facilitating the transmission of **DataContainer** objects from the reading task to the handling task does not contain the objects themselves but rather pointers to them and because the objects may vary in size, the required memory can not be allocated upon system startup.

With this information in mind, the decision was made to limit the number of **DataContainer** objects that could be stored in the **RTOSQueue** to 16: If for some reason the handling task cannot keep up with the reading task's performance, there is a small buffer to give it an opportunity to catch up, but the buffer is still small enough to be written to its maximum size without threatening to consume more than the allowed amount of system memory. In case the buffer was not enough, all newly read data would be discarded while logging the incident.

The number 16 was chosen because two objects should be able to exist outside of the queue: One that is currently being created and one that is currently being handled. After it has been handled, each object will be deleted, freeing up the memory for new data. In sum, the 18 objects fill the 150 KB limit almost perfectly.

7.2.2 AT Command Implementation

As described in Section 7.1.2, the cellular module attached to the ESP32 microchip at the core of the telemetry device can be controlled by AT commands. While this offers a very concise

²While this is not confirmed anywhere in the module's documentation, the similarities between parameters and return values of AT commands and Lua libraries seem to hint that the hardware functionalities offered to Lua scripts are simple wrappers to AT commands.

interface, implementing functionality with these commands turned out to be more difficult than initially expected for four main reasons:

The first reason is the lack of clear documentation and examples: Although an exhaustive documentation to the commands and their parameters exists (see SIMCom, 2017a), it is flawed in important ways: While all individual commands are explained in detail, the interaction between them is often not explained. For example, the command `AT+CREG` to register the device to a network must be called before `AT+CGREG` to register to a GPRS network, however this is not mentioned in the documentation.

Secondly, error handling with AT commands is nearly non-existent: While some of the commands provide basic information in case of an error (for example the `AT+CHTTPSEND` command, which sends an HTTPS request, informs about the type of error that has occurred (see SIMCom, 2017b), others, such as the `AT+CSSLOADCK` command to load certificate files only return the string "OK" or "ERROR" as feedback. This made troubleshooting and error finding very difficult.

The third reason is that there is little to no information available from people or companies that have worked through similar issues: Presumably this is because the hardware itself is not very commonly used and because it is predominantly used in corporate projects, whose owners are reluctant to share information or experience online. IAV has not worked with the technology in question before so that no help or guidance was available from company-internal sources either.

Finally, and most significantly, the use of AT commands requires a client to repeatedly wait for feedback from the module in order to verify the success of the command. No documentation exists for a recommended interval between commands, but experiments show that delays of less than 50 milliseconds between commands can cause issues with incompletely transmitted requests or responses³. Since establishing and maintaining connections for the transmission of data requires a significant number of commands in succession, this would result in a negative performance impact.

In order to work around these issues, the decision was made to make use of the module's ability to internally run Lua scripts for the transmission of data. By doing this, responsibility was transferred away from the microcontroller and onto the cellular module itself.

More specifically, the Lua script's responsibility is to check for files in the module's file system that contain the data to transmit. The content of these files is then read and transmitted. The microcontroller's responsibility is now only to gather data and write it into the module's file system. This has the advantage that instead of at minimum three commands (opening a network session, transmitting data and checking for a response), only one has to be called by the microcontroller for the transmission of data to the cellular module's file system. Furthermore the module's flash storage now acts as a second, larger data buffer. Figure 7.1 visualises these interactions between the device components.

³The reason for these issues is unknown. With the transmission rate of 115200 baud (see SIMCom, 2017c) and approximately 50 bytes per command, around 4 ms should theoretically be enough ($400 \text{ bytes} / 115200 \text{ baud} \approx 0.0034 \text{ seconds}$) to transmit commands with no issues.

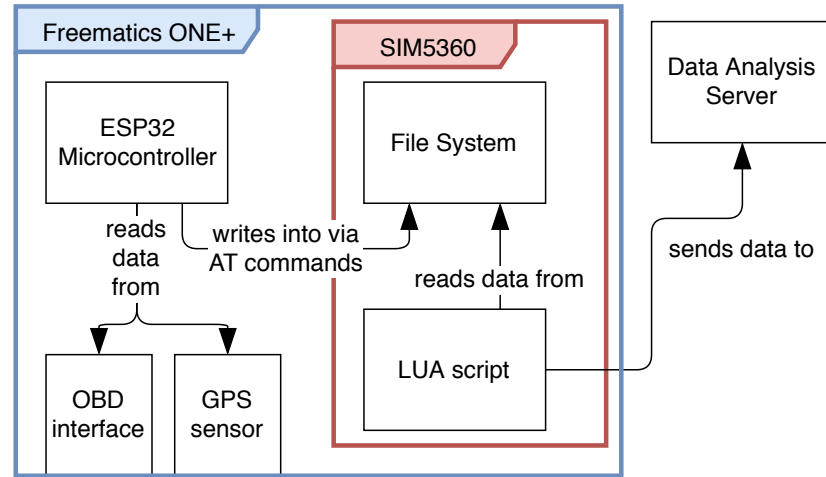


FIGURE 7.1: Hardware/ Software Interactions on the Target Device

7.2.3 Task Timing

As described in Section 5.4, the system's two core tasks are designed to be split over the device's two processing cores: While one core handles the reading of data, the second checks for the presence of data in the RTOS Queue (see Section 5.3.1) and handles it.

Because of insufficient knowledge at the time, two major issues related to tasks were introduced to the system by this design.

The first issue is that by deleting the task carrying out the `loop()` function and replacing it with a custom task each for the `readerLoop()` and `handlerLoop()` functions (see Section 5.2.1), an important functionality was also deleted:

In order to track the time in microseconds that has elapsed since the system was powered up, the `micros()` function divides the number of completed processor clock cycles by the clock's frequency. In order to prevent premature overflow of this value caused by the processor's relatively high clock speed, the `micros()` function needs to be called approximately every 17 seconds⁴ in order to account for the number of times the count has overflowed (see me-no-dev, 2017b). This is normally done after every call of the `loop()` function by the main task (see me-no-dev, 2017a). By deleting this task however, the `micros()` function is also no longer being called regularly. In order to account for this, the function had to be included into the custom tasks. Because the reading task was assumed to be less likely to stall for an extended period of time, it was placed in the `readerLoop()` function.

The second issue is caused by running both tasks in infinite loops simultaneously: Because neither task leaves the "Running" state, no other task of lower priority can ever be executed. This also affects the idle task created by the system framework on startup, whose responsibility is to clean up kernel resources whenever a task is deleted (see FreeRTOS, 2017a). Although currently, no tasks are being deleted in the system, the design aim of extendibility still counts during the implementation phase. As mentioned in Section 5.4, a possible future change to the system could be the introduction of individual tasks for each data handling type. In order to facilitate this, the decision was made to prevent the starvation of the idle task.

This was realised by limiting the number of times each task could be executed per second and waiting for the next allocated time chunk if it exited earlier. This way, both tasks would

⁴(Unsigned integer maximum value [4.294.967.296] / clock speed in Hz [240.000.000] \approx 17.89)

7.2.4 Certificate Installation

In order to establish a TLS-encrypted connection to the remote analysis server, certificates and key files had to be installed onto the cellular module. Because the software provided by the manufacturer to access the module's file system refused to work in combination with the microcontroller over which all communication between PC and module had to be bridged, a custom solution had to be implemented. This was realised in the form of a python script that reads the selected files and transmits them to the module via the serial connection. By calculating a checksum from the original file and the transmitted file, a successful transmission can be ensured.

7.2.5 Certificate Usage

Finally, a problem was encountered when implementing the transmission of data from the device to the remote analysis server: Attempting to load and use installed TLS certificates results in an error. This subsection describes the problem and all attempted solutions in detail.

Problem Description

In order to fulfill the requirement NF-HTTPS, an encrypted connection has to be established to IAV's remote analysis server using the cellular module's HTTPS functionality. For reasons of data security and integrity, a two-way authenticated connection is used, meaning that not only the server's certificate needs to be verified but also a client certificate needs to be transmitted to the server for verification.

The first step to realise this functionality was the installation of certificate files to the cellular module's internal flash storage (see Section 7.2.4). After this was completed, the next step was to initiate an encrypted connection to the server using these certificate files. After selecting the certificate files to use, they then need to be loaded using the `AT+CSSLOADCK` command. At this point, the module returns the status string "ERROR" without additional information.

Problem Analysis

In an attempt to fix the problem, it was analysed thoroughly, yielding a number of facts about the problem that could possibly help to find a solution:

1. The used client and root certificates are valid. Testing them on various other systems confirmed that they are accepted by the server and verify its certificate.
2. The error occurs when loading certificates, before they are being used. An unencrypted connection can still be established. This means that the problem exists unrelated to the server and is not caused by an authentication error.

Fix Attempts

Based on the results of the problem analysis, various approaches to find a solution were identified and executed.

Firstly, the possibility of using invalid or incorrectly transmitted certificate files was eliminated by checking the validity and contents of the certificate files.

Secondly, by following all product documentation precisely (see SIMCom, 2013), the possible cause that commands were used incorrectly was accounted for. This was further confirmed by re-implementing the same functionality in a Lua script which yielded no further error information.

Because the module's documentation states that certificate files can be installed in two different formats (.der / encoded in binary and .pem / encoded in Base64) (see SIMCom, 2017d), each of the files was installed in both formats and using them was attempted in all possible combinations. In the case of the .pem files, this included installing four versions of each certificate: using both carriage return (CR) and line feed (LF) for new lines, using CR and LF individually and completely omitting new lines.

Furthermore, a number of smaller changes were made to the implementation such as using the exact file names used in the documentation or attempting the process on a completely reset module in order to account for full storage or changed settings.

Finally, a hardware issue was considered, but both the fact that all other commands work without a problem and the result of testing the code on another device of the same type show that this is not the cause of the error.

During all attempts, various tries at receiving outside help were made, such as discussing the problem with developers from other departments of the company as well as tutors, none of which resulted in a solution. The module's manufacturer was neither able to identify the error nor provide usable troubleshooting guidance when contacted.

Eventually, the decision was made to work around the problem by implementing unencrypted connections first and concentrating on realising the project's other requirements.

Current Status

Up until the end of the project, no solution could be found to the certificate usage problem. For this reason, the transmission of data was implemented over an unencrypted HTTP connection instead. While this is in clear violation of the requirement NF-HTTPS, it allows for testing and validation of other project parts. Because the command structure to establish connections is identical whether or not encryption is used in both AT commands (see SIMCom, 2017e) and Lua functions (see SIMCom, 2015), all implementation can easily be switched to HTTPS if the problem was solved in future.

7.3 Result

Despite all efforts to implement the product based on the previously made software designs and the customer's requirements, it could not be realised up to full specifications: As described in Section 7.2.5, no encrypted connection to the remote server could be established due to an error in the process of loading certificate files. Apart from two minor requirements (F-SD-Data-Discard and F-Bluetooth-Control), the product is otherwise feature complete.

8 Validation

As described in Section 2.2, part of the project’s task scope was the validation of all products. This consists of examining the results of unit and integration tests as well as analysing quality measurements and verifying that all initially identified requirements are fulfilled. The following sections describe these tasks and their results.

8.1 Software Tests

This section describes the results of the software tests carried out as decided upon during the design phase of the project (see Section 5.5).

8.1.1 Unit Tests

Due to the decision to make use of test-driven development, the execution of tests was a vital part of the development process: By implementing functionality to make tests pass, functionality of methods and functions was decided upon before any real implementation was made. This resulted in 100 per cent successful tests at the end of the project.

However parts of the system could not be tested using unit tests, such as the reading or transmission of data, both of which rely heavily upon hardware calls and whose results vary too greatly to verify them with predefined expected results. In order to test these parts, manual tests were carried out whenever implementations were changed in order to verify no functionality was broken or changed. At the end of the project, all desired functionality such as reading data from the device’s OBD interface and GPS sensor or handling data via mobile network transmission and SD card writing was verified to work as expected.

8.1.2 Integration Tests

The next step after testing the product’s code base in its individual parts was to verify that all parts work together flawlessly. This was realised by carrying out manual integration tests. At the beginning of the project, when not many results were available yet, these were carried out by using dummy modules where implementations were not complete yet.

At later stages, the product was deployed to its target device and tested in its target environment by attaching the device to a vehicle and carrying out test drives.

These test showed that all functionality apart from the encrypted transmission of telemetry data worked flawlessly. Section 7.2.5 explains in depth the problems encountered with the transmission of data.

8.2 Quality Validation

After verifying that all parts of the software product were working as intended, the next step was to take a final measurement of all quality metrics in order to defined the overall quality reached by the product.

Based on the results of quality measurements that were to be taken bi-weekly (see Appendix E.3.3), Figure 8.1 visualises the change of measured quality scores over time by plotting the calculated score of each measurement. Since lower score indicate higher quality, the graph is inverted.

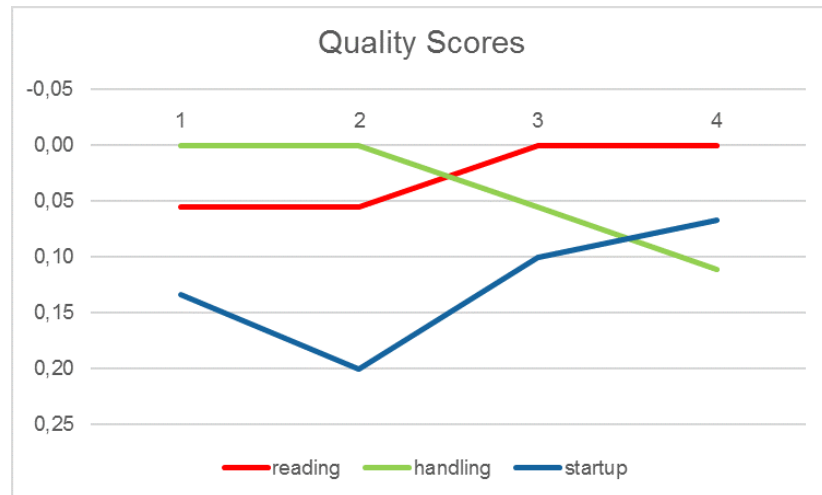


FIGURE 8.1: Calculated Score of Quality Metrics over Time

This shows that none of the quality metrics left the range of acceptable scores over the course of the implementation. Furthermore the final quality scores are visible: While the highest score was reached by the data reading metric, the handling metric leaves room for improvements.

Due to the fact that some implementation changes were made at the very end of the project however, such as switching the transmission of data from being controlled by AT commands to implementing a Lua script for this purpose, the final measurement does not represent the final product's performance. Apart from an improvement in the handling score, no significant changes are estimated to be shown however.

This leads to the conclusion that, with respect to the quality metrics, the project was a success.

8.3 Requirement Comparison

The final step to verify the project's completeness and success was to make sure that all requirements had been met.

As explained in Section 4.3.2, the projects requirements were split into core and auxiliary requirements: While the core requirements were vital to the project's success, all auxiliary requirements simply added functionality.

Of all requirements listed in Appendix D.2.3 and Appendix D.2.4, only F-SD-Data-Discard, F-Bluetooth-Control and NF-HTTPS remained unfulfilled. While F-SD-Data-Discard and

F-Bluetooth-Control remained unfulfilled because of time constraints, NF-HTTPS could not be fulfilled due to errors in system calls. Section 7.2.5 analyses this problem in detail.

8.4 Contradiction Between Results

The stark contrast between the almost perfect score reached by the product for its core quality measurements and the fact that one of its core requirements was not implemented correctly at all raises the question whether the selected quality requirements were chosen correctly.

By having a closer look at the metrics and their intended purpose however, it becomes obvious that they are less concerned with the system's functionality and more so with the performance numbers shown by the product. This means that while additional metrics could have been chosen to portray the product's functionality more accurately, these would have relayed no additional information that is not already conveyed in the requirements check.

9 Conclusion

The project's original aim was to improve the process of carrying out test drives for newly designed vehicles and vehicle parts by creating an easy-to-use software tool that gathers data from the vehicle and transmits it to a remote server along with location data. Additionally, a feasibility study concerning the real-time analysis of the same data had to be carried out.

In order to fulfill the first task, the implementation of the telemetry data transmission tool, four of the five software development phases were carried out (analysis, design, implementation, validation). After all requirements and other parameters were identified, the product was designed based on this information. Implementing the tool based on the designs and verifying that all aims and targets were fulfilled were the next steps.

Unfortunately, the end result of this task did not fulfill all of its requirements. Most importantly, the encryption of all data transmission could not be realised due to some functionality not working. Despite significant effort, this error could not be fixed and persists in the product's final version. This means that the company cannot use the product in real-world settings.

The second task, carrying out a feasibility study on the topic of real-time data analysis on the telemetry device, was completed without problems. Its results shows that, while implementing real-time analysis of telemetry data on the device is not a problem on its own, some data may be more difficult to obtain than anticipated. Furthermore, combining the transmission and analysis of telemetry data will most likely lead to undesirable performance issues.

Although the project was not in itself successful in creating improvements to the company's test drive processes, its results and the knowledge gained during the execution can easily be used to create such improvements. If, for example, the problem blocking the use of encrypted connections for data transmission was solved, only two configuration items need to be changed to make the product fully viable for its intended purpose.

10 Recommendations

Although the product in its current state is not viable for real-world application, it is considered mostly feature-complete with only minor changes required to fix all known flaws. For this reason, two recommendations can be made with the intent of improving the project in various ways.

The first recommendation is concerned with working around the HTTPS problem and making the product viable for real-world application. By implementing a relatively simple protocol to send data that was originally intended to be transmitted over mobile networks over the device's serial connection, an external device such as a smartphone or a mini computer like the Raspberry Pi could take over the responsibility of transmitting data. Although the additional device would interfere with the product's "plug and play" feature, this would mean that the intended functionality could still be fulfilled.

The second recommendation is to replace or refactor the device libraries supplied by the manufacturer: Because they incorporate code not only for the specific device but also for older versions with different hardware, they require unnecessarily large amounts of memory to load. By removing all unused code, performance could be improved or additional features could be implemented. Furthermore, implementation and maintenance could be simplified if instead of importing all hardware functionality at once, clients could choose to import parts of it, such as SD card handling, on their own.

List of References

- alronzo, pseudonym (No date). *GPS Basics - learn.sparkfun.com*. URL: <https://learn.sparkfun.com/tutorials/gps-basics> (visited on May 3, 2018).
- Arduino (No date[a]). *Arduino - Products*. URL: <https://github.com/espressif/arduino-esp32/blob/master/cores/esp32/main.cpp> (visited on Apr. 18, 2018).
- Arduino (No date[b]). *Arduino Sketch*. URL: <https://www.arduino.cc/en/Tutorial/Sketch> (visited on Feb. 20, 2018).
- Barry, R. (Dec. 2016). *Mastering the FreeRTOSTM Real Time Kernel*. Real Time Engineers Ltd., pp. 243–258. URL: https://www.freertos.org/Documentation/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf.
- Dana, P. H. and B. M. Penrod (1990). *The Role of GPS in Precise Time and Frequency Dissemination*, 5. URL: http://www.pdana.com/PHDWWW_files/gpsrole.pdf (visited on Apr. 19, 2018).
- Eden, A. (Feb. 2018). *Initial Project Interview, Translated from German*.
- Espressif Systems (Jan. 2018). *ESP32 Datasheet*. Espressif Systems. URL: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.
- Fowler, M. (2003). *UML Distilled Third Edition*.
- Freemetrics (No date[a]). “Freemetrics ONE+ Developers Guide”. In: URL: <https://freemetrics.com/pages/products/freemetrics-one-plus/guide/> (visited on Feb. 19, 2018).
- Freemetrics (No date[b]). “Freemetrics ONE+ Product Page”. In: URL: <https://freemetrics.com/pages/products/freemetrics-one-plus/> (visited on Feb. 19, 2018).
- FreeRTOS (2017a). *The FreeRTOSTM Reference Manual - API functions and configuration options*. Amazon Web Services, p. 76. URL: https://www.freertos.org/Documentation/FreeRTOS_Reference_Manual_V10.0.0.pdf.
- FreeRTOS (2017b). *The FreeRTOSTM Reference Manual - API functions and configuration options*. Amazon Web Services, 157 ff. URL: https://www.freertos.org/Documentation/FreeRTOS_Reference_Manual_V10.0.0.pdf.
- FreeRTOS (2017c). *The FreeRTOSTM Reference Manual - API functions and configuration options*. Amazon Web Services, pp. 48–52. URL: https://www.freertos.org/Documentation/FreeRTOS_Reference_Manual_V10.0.0.pdf.
- Gamma, E., R. Helm, R. Johnson, and J. Vlissides (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 156 ff.
- Google LLC (May 2018). *Overview | Geocoding API | Google Developers*. URL: <https://developers.google.com/maps/documentation/geocoding/intro#ReverseGeocoding> (visited on May 3, 2018).
- Huang, S. (Oct. 2017). *GPS problems - Freemetrics Forum*. URL: <https://freemetrics.com/forum/viewtopic.php?f=14&t=2593#p5689> (visited on Apr. 18, 2018).
- IAV (Jan. 2017). “IAV in Figures”. In: URL: <https://www.iav.com/en/about-iav/iav-figures>.
- IEEE (June 1998). *IEEE Standard for Software Quality Assurance Plans*. Software Engineering Standards Committee of the IEEE Computer Society. URL: <http://users.csc.calpoly.edu/~jdalbey/308/Resources/IEEE7301989.pdf> (visited on May 2, 2018).

- Jimb0, pseudonym (No date). *Serial Communication - learn.sparkfun.com*. URL: <https://learn.sparkfun.com/tutorials/serial-communication> (visited on Apr. 25, 2018).
- Kanda Admin, pseudonym (Nov. 2012). *CAN Bus and OBD-II*. (Visited on May 4, 2018).
- me-no-dev, pseudonym (Aug. 2017a). *arduino-esp32/main.cpp at master · espressif/arduino-esp32 · GitHub*. URL: <https://github.com/espressif/arduino-esp32/blob/master/cores/esp32/main.cpp>.
- me-no-dev, pseudonym (Mar. 2017b). *micros() overflow fix by jpmeijers · Pull Request 267 · espressif/arduino-esp32 · GitHub*. URL: <https://github.com/espressif/arduino-esp32/pull/267>.
- SIMCom (Dec. 2013). *SSL Application Note*. SIMCom, pp. 15–17, 21–23. URL: http://simcom.ee/documents/SIM5360/SIM5360_SSL_Application_Note_V0.01.pdf.
- SIMCom (July 2015). *SIM5360 LUA Application Note V0.05*. SIMCom, pp. 131–135. URL: http://simcom.ee/documents/SIM5360/SIM5360_LUA_Application_Note_V0.05.pdf.
- SIMCom (Mar. 2017a). *AT Command Set SIM5360*. SIMCom. URL: http://simcom.ee/documents/SIM5360/SIMCOM_SIM5360_ATC_EN_V0.24.pdf.
- SIMCom (Mar. 2017b). *AT Command Set SIM5360*. SIMCom, pp. 469, 470. URL: http://simcom.ee/documents/SIM5360/SIMCOM_SIM5360_ATC_EN_V0.24.pdf.
- SIMCom (Mar. 2017c). *AT Command Set SIM5360*. SIMCom, p. 25. URL: http://simcom.ee/documents/SIM5360/SIMCOM_SIM5360_ATC_EN_V0.24.pdf.
- SIMCom (Mar. 2017d). *AT Command Set SIM5360*. SIMCom, p. 517. URL: http://simcom.ee/documents/SIM5360/SIMCOM_SIM5360_ATC_EN_V0.24.pdf.
- SIMCom (Mar. 2017e). *AT Command Set SIM5360*. SIMCom, pp. 467, 468. URL: http://simcom.ee/documents/SIM5360/SIMCOM_SIM5360_ATC_EN_V0.24.pdf.
- Sparkfun Electronics (No date). *SparkFun Atmospheric Sensor Breakout - BME280 - SEN-13676 - SparkFun Electronics*. URL: <https://www.sparkfun.com/products/13676> (visited on May 3, 2018).
- Wikipedia (No date). *OBD-II PIDs - Wikipedia*. URL: https://en.wikipedia.org/wiki/OBD-II_PIDs (visited on Apr. 20, 2018).
- Windl, U., D. Dalton, M. Martinec, and D. R. Worley (Nov. 2006). *The NTP FAQ and HOWTO: Understanding and using the Network Time Protocol*. URL: <http://www.ntp.org/ntpfaq/NTP-s-algo.htm#Q-ACCURATE-CLOCK> (visited on Apr. 18, 2018).

A Project Plan

PROJECT PLAN

for the project

*Transmission and Analysis of Vehicle
Telemetry Data using OBD-II
and Cellular Networks*

IAV GmbH

by Nicholas Walter

Gifhorn, March 6, 2018

Table of Contents

| | |
|-------------------------------------|-----------|
| A.1 Context | 37 |
| A.1.1 The company | 37 |
| A.1.2 The department | 37 |
| A.2 The Problem | 38 |
| A.2.1 Problem description | 38 |
| A.2.2 Solution draft | 38 |
| A.2.3 Expected results | 38 |
| A.3 The Project | 39 |
| A.3.1 Assignment | 39 |
| A.3.2 Deliverables | 39 |
| A.3.3 Scope | 40 |
| A.3.4 Organisation | 41 |
| A.3.5 Testing and Quality Assurance | 42 |
| A.4 Planning | 43 |
| A.4.1 Initial planning | 43 |
| A.4.2 Deadlines | 43 |
| A.4.3 Communication | 43 |
| A.5 Revision History | 45 |

A.1 Context

A.1.1 The company

IAV ("Ingenieurgesellschaft Auto und Verkehr" / "Engineer Society Automobile and Traffic") originated as a research institute as part of the Technische Universität Berlin in 1983. Today, they employ more than 6,000 people (see IAV, 2017) and the company has become a world-wide supplier of automotive parts and designs.

A.1.2 The department

The project described in this document will be carried out in the department "Vehicle IT", which is responsible for the development of software in, around, and related to automobiles for IAV's customers. Part of their every-day job is the testing and validation of new features, parts or designs in so-called "Test vehicles" ("Versuchsfahrzeuge").

A.2 The Problem

The following subchapters will define the problem the project intends to solve, a rough solution draft taken as basis for future definitions and plannings as well as results the stakeholders expect to see when the project is completed.

A.2.1 Problem description

As described in subsection A.1.2, a big part of the department's responsibility is the validation of designs and parts created for customers. Reading and analysing information from these vehicles is a vital part of these tests. At the moment, the department uses standalone devices that can be connected to both the vehicle and a cellular connection in order to retrieve data and send it to a remote server where it can be analysed, however these are bulky, expensive and take a not insignificant amount of time to install before tests can actually be started.

A.2.2 Solution draft

The project described in this document offers a solution to the problems outlined in subsection A.2.1 in form of an Arduino-based OBD-II dongle. The "ONE+" produced by the company Freematics uses the well-known Espressif ESP32 as main controller and is therefore fully programmable. Additionally, it houses a variety of sensors (Global Positioning System (GPS), Motion/ Orientation/ G-force, On Board Diagnostics (OBD) and is capable of transmitting data in a number of ways (SD-storage, Bluetooth, Cellular connections, WiFi) (see Freematics, No date(b)). Using these functionalities, the basic premise of the project is to program the device to read data from the diagnostics port of the vehicle it is attached to, briefly analyse this data and then to transfer it to a remote web-server for more in-depth analysis using the previously mentioned data transmission possibilities. A more exhaustive description of the project will be made once the required analyses have been completed and the project has been defined in more detail.

A.2.3 Expected results

At the end of the project, the device should be capable of reading and transmitting data as described in subsection A.2.2. It is expected to make testing of vehicles and vehicle parts easier than it is now, saving time and therefore money.

A.3 The Project

The following sections explain the process to solve the problem outlined in subsection A.2.1. They describe the project assignment in detail, define deliverables and scope as well as other project parameters.

A.3.1 Assignment

The project is divided into two core objectives: The implementation of the solution for the problem described in subsection A.2.1 and a research report. The following paragraphs briefly describe both of them.

As described in subsection A.2.2, the basic premise of the solution is to program an Arduino-based OBD-II dongle so that it reads and briefly analyses data coming from the vehicle it is attached to.

Simultaneous to the process of gathering and analysing data, the device should also transmit data to a pre-existing remote server for further analysis. This transmission needs to be secure and should be realised using the cellular connection that can be established using the built-in 3G module.

The specific data to read and analyse as well as detailed information regarding the transmission of this data will be defined in a requirements analysis at a later stage of the project.

Simultaneous to the development of the product itself, a research report on the topic of whether or not the device can be used for a specific real-time data analysis task should be created (see Appendix A.3.2).

A.3.2 Deliverables

The following subsections describe the two core objectives to be carried out during the execution of the project. These will be more clearly and granularly defined once the project analysis in general and the requirements analysis specifically have been concluded.

Software

The project consists of two main deliverables: As described in Appendix A.3.1, the first deliverable is a piece of software to run on the afore-mentioned OBD-II dongle with the aim of reading, analysing and transmitting data from any OBD-compatible vehicle it is attached to.

Attached to this software, a number of documents defining and describing the project and its parts should be created:

- Project Plan
- Stakeholder Analysis
- Requirements Analysis
- Quality Management Plan
- Software Design Document

- Intermediate Progress Report
- Project report (in form of Thesis)

Deadlines for each of these items can be found in Appendix A.4.2.

Research

The second deliverable is a report detailing the results of the research concerning the feasibility of analysing vehicle data in real time in order to identify invalid test runs:

Because modern cars consist of thousands of individual parts, each of which must be tested to find flaws and to prove it is working correctly, so called test runs are part of every vehicle design company's every-day business. These consist of very strictly pre-defined parameters (such as what type of road to drive on what for what time and distance, what speed to drive at, times the vehicle must be in idle, etc.), which often makes it difficult for test drivers to recognise when the test drive they are currently working on became invalid because a parameter cannot be fulfilled anymore. Because these tests are not cheap to carry out, errors like this can cause significant expenses that need to be avoided.

The research should clarify whether it is feasible to create a software that provides instance feedback to drivers on test runs, notifying them of invalid runs so that they can be aborted before they are completed.

A.3.3 Scope

In order to be able to fulfill the project's requirements to their fullest within the given time frame, its scope must be clearly defined so that feature-creep and bloating do not threaten to sabotage the project. The following subchapters clearly define what is included in and what is excluded from the project scope for the two core objectives.

Software

Included

- Analysis of stakeholders, risks and requirements
- Design of software solution
- Implementation of software solution
- Validation of software solution against requirements and predefined quality goals
- Creation and Maintenance of the documents described in section A.3.2

Excluded

- Maintenance of software solution
- External components that interface with the product (e.g. mobile applications to realise bluetooth controls)

Research

Included

- Analysis of posed research question

Excluded

- Implementation of real time data analysis tool (may optionally be considered if time constraints allow it)

A.3.4 Organisation

In order to fully understand the project in its details, it is also important to know how it is organised. The following subchapters briefly summarise the results of the stakeholder analysis (see Appendix B) and briefly introduce the project team.

Stakeholders

In order to identify and classify people and parties relevant to the project, a stakeholder analysis was carried out. The graphic below (Figure A.1) shows the result of their classification, assigning each of the identified entities a group defining the optimal way to handle them during the project's execution.

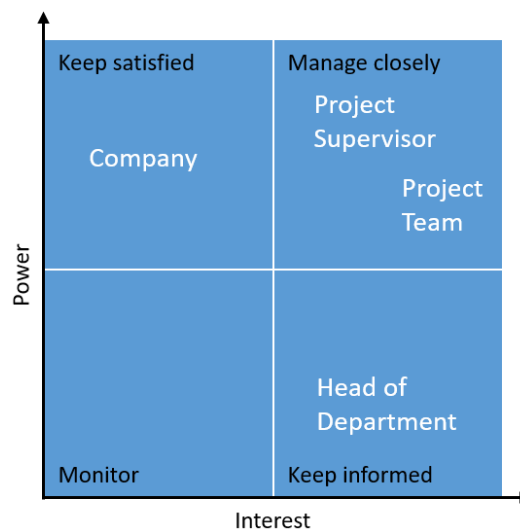


FIGURE A.1: Stakeholder Power/ interest grid

This shows that the most important party to consider is the project supervisor who has the most interest in and the most power over the project.

People

Due to the project's nature as a graduation thesis, only very few people are relevant to the project, see Table A.1. The table lists them by name and defines their roles and responsibilities.

TABLE A.1: People relevant to the project

| Name | Role | Responsibilities |
|-------------------------|--------------------|---|
| Dr. Ing. Sebastian Bode | Project Owner | - |
| Dr. Ing. Arnd Eden | Project Supervisor | Supervision of project |
| Nicholas Walter | Project Developer | Analysis, Design, Implementation and Validation of product, Research |

A.3.5 Testing and Quality Assurance

Because the project's aim is to create a product that will handle sensitive data in possibly a production environment, it is important that it is implemented very closely to its specification, especially with respect to data security. In order to ensure this, a quality management plan that will define testing and validation measures will be created within the starting phase of the project. The deadline for this document is defined in subsection A.4.2.

A.4 Planning

The following sections describe the initial planning made for the project's execution, which consists of roughly defined start and finish dates for each core segment, and a number of deadlines set for important documents and deliverables.

A.4.1 Initial planning

Because the project is very limited with regard to its duration, it is important to plan its execution in detail in order to ensure that all of its goals can be completed within the given time frame. For this reason, all of the most important tasks to complete were gathered and listed with their respective deadlines. A chart to visualise this data was created to make it easier to recognise upcoming due dates and track progress, see Figure A.2.

The given time frame was divided to accommodate both main deliverables (see subsection A.3.2). Furthermore the project section was subdivided into its individual phases.

Because the project is still in its early days at the time this document is produced, all data is preliminary and subject to change, for example once the requirements analysis is concluded.

A.4.2 Deadlines

Having defined rough guidelines for the project's time planning, the next step was to define deadlines for the deliverables described in subsection A.3.2, these are listed in Table A.2.

TABLE A.2: Deliverables Deadlines

| Deliverable | Deadline |
|------------------------------------|------------|
| Project Plan | 2018-03-06 |
| Stakeholder Analysis | 2018-03-06 |
| Requirements Analysis | 2018-03-16 |
| Quality Management Plan | 2018-03-26 |
| Software Design Document | 2018-03-26 |
| Intermediate Progress Report | 2018-03-27 |
| Software solution | 2018-06-12 |
| Research report | 2018-06-12 |
| Project report (in form of Thesis) | 2018-06-12 |

A.4.3 Communication

To ensure that a high quality product can be delivered at the end of the project, communication between the project team and the customer is of vital importance, since requirements or other parameters could change quickly. Because of the project's small scale however, and because it proved to be difficult to schedule weekly meetings, a full-scale implementation of an established communication pattern such as Scrum was impractical. For this reason, a very reduced version will be implemented by establishing a bi-weekly meeting with the intention of discussing results of the previous time slice and plans for the upcoming time slice. Based on these discussions, the project plan will be updated.

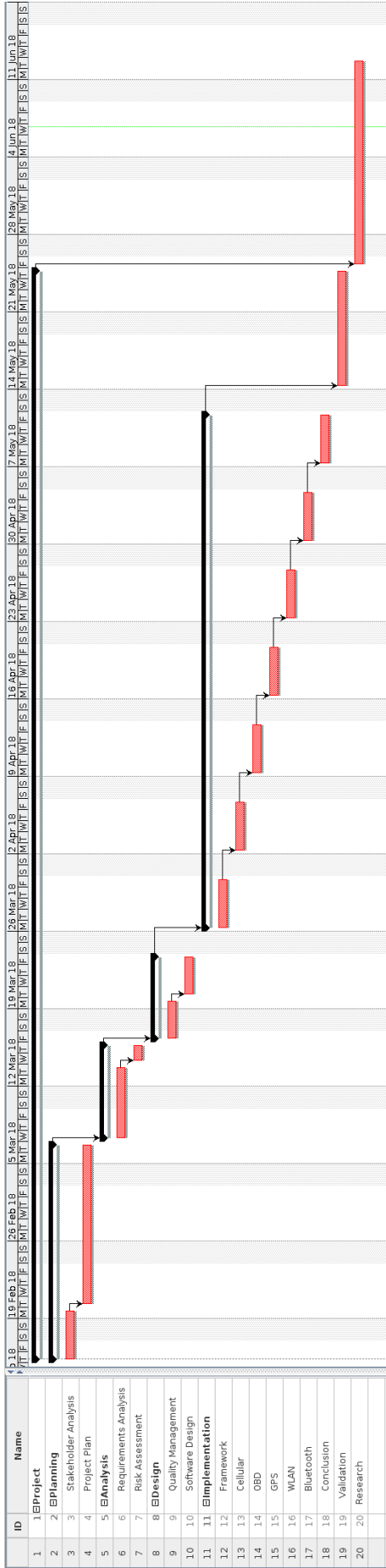


FIGURE A.2: Gantt Chart

A.5 Revision History

TABLE A.3: Revision History

| Date | Changes |
|------------|------------------------------|
| 2018-03-20 | Defined scope in more detail |
| 2018-03-19 | Added communication chapter |
| 2018-03-06 | Original Document |

B Stakeholder Analysis

STAKEHOLDER ANALYSIS

for the project

*Transmission and Analysis of Vehicle
Telemetry Data using OBD-II
and Cellular Networks*

IAV GmbH

by Nicholas Walter

Gifhorn, March 6, 2018

Table of Contents

| | |
|-------------------------------------|-----------|
| B.1 Identification | 48 |
| B.2 Prioritisation | 49 |

B.1 Identification

In order to gather requirements, all people and parties with power over the project and all entities that would likely be affected by the project and its results in the presence or in the future would need to be considered. To make this easier, a list of these entities was compiled based on informal background research regarding the project and its context as well as stakeholder nomination, where already identified key stakeholders were asked to list other possible stakeholders.

The resulting list of identified stakeholders is depicted in Table B.1.

TABLE B.1: Stakeholders

| Stakeholder | Description and Relevance |
|-------------------------|---|
| Company IAV | Interested in using time and resources efficiently |
| Department "Vehicle IT" | Interested in using new technologies to make their work easier and more efficient |
| Project Supervisor | Will make final decisions |
| Project Team | Will create the product |

B.2 Prioritisation

With the intention of creating an easier overview over the stakeholders identified in section B.1, they were categorised based on their power over and interest in the project. The resulting Power / Interest grid is depicted in Figure B.1

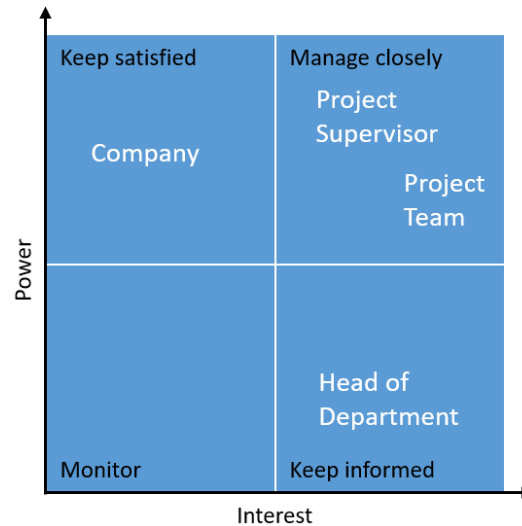


FIGURE B.1: Stakeholder Power/ interest grid

Using this visualisation, it is easy to determine how each of the stakeholders should be handled during the project execution:

The lower left corner holds parties that are of very little importance during the project. Their wishes and ideas for the project should however be taken into consideration during the analysis and design phase. None of the stakeholders identified for this project fit this description.

Parties in the upper left corner mostly have no direct interest in the project itself, however their influence over it could be very big if they desired. However unlikely, any input from these parties should be taken into close consideration. For this project, only a single entity is sorted into this corner: The company itself as represented by its executives. Given that IAV is a company of considerable size, it is unlikely that any information regarding this project specifically is conveyed to them or that they would take any action to influence it.

The lower right corner contains parties that can only have very little influence over the project's execution but who have a large interest in it nonetheless. Specifically, it holds the head of the department "Vehicle IT". In their role as supervisor to all projects that are being carried out in the department, this person's interest in the project is considerable, however their direct influence over it is lower than that of other parties.

Finally, the upper right corner holds people with large influence and a lot of interest in the project: Their wishes and needs should be satisfied at all times in order to keep the project going. In this instance, the project supervisor as well as the project team reside in this corner.

C Risk Analysis

RISK ANALYSIS

for the project

*Transmission and Analysis of Vehicle
Telemetry Data using OBD-II
and Cellular Networks*

IAV GmbH

by Nicholas Walter

Gifhorn, March 7, 2018

Table of Contents

| | | |
|------------|--------------------------------------|-----------|
| C.1 | Introduction | 52 |
| C.2 | Risk Identification | 53 |
| C.3 | Risk Evaluation | 54 |
| C.4 | Risk Handling | 55 |

C.1 Introduction

This document describes the identification and evaluation of risks to the project "Vehicle Telemetry using OBD-II and cellular networks". It highlights what risks exist during the execution of the project, attempts to estimate their likelihood of occurring and their impact, and defines mitigation and contingency measures.

C.2 Risk Identification

Since the project's goal is not to change an existing workflow in the company or make changes to a pre-existing product, there are no risks concerning the loss of functionality or causing delays in the context of other work or projects. Furthermore the project's small team size means that there are no risks regarding a sudden loss or change of team members by default. All other risks that could be identified have been listed in Table C.1, describing their conditions and consequences.

TABLE C.1: Identified risks

| # | Condition | Consequence |
|---|--|---|
| 1 | Insufficient time to finish project | Not all requirements can be realised |
| 2 | Device does not offer required functionality | Not all requirements can be realised |
| 3 | Additional requirements appear during project execution | More time is required to meet all requirements |
| 4 | A requirement takes longer to implement than anticipated | More time is required to meet all requirements |
| 5 | Hardware programming causes difficulties/ delays | Some core features may remain unfulfilled/ take longer so that other requirements cannot be worked on |

C.3 Risk Evaluation

After risks were identified, their individual exposure was calculated from their probability of occurring (0 to 1) and the impact they would have on the project (0 to 10). The higher the exposure, the more dangerous a risk would be to the project. Table C.2 lists the risks ordered by their exposure.

TABLE C.2: Risk Exposure

| # | Probability | Impact | Exposure |
|---|-------------|--------|----------|
| 5 | 0.2 | 6 | 1.2 |
| 3 | 0.25 | 4 | 1 |
| 2 | 0.1 | 7 | 0.7 |
| 4 | 0.2 | 3 | 0.6 |
| 1 | 0.1 | 6 | 0.6 |

For better readability, this data was then projected into a graph plotting impact over probability (see Figure C.1), split into four quadrants: Bottom left for risks that are unlikely to have significant influence over the project, top right for those that pose a major threat and top left / bottom right for those in between.

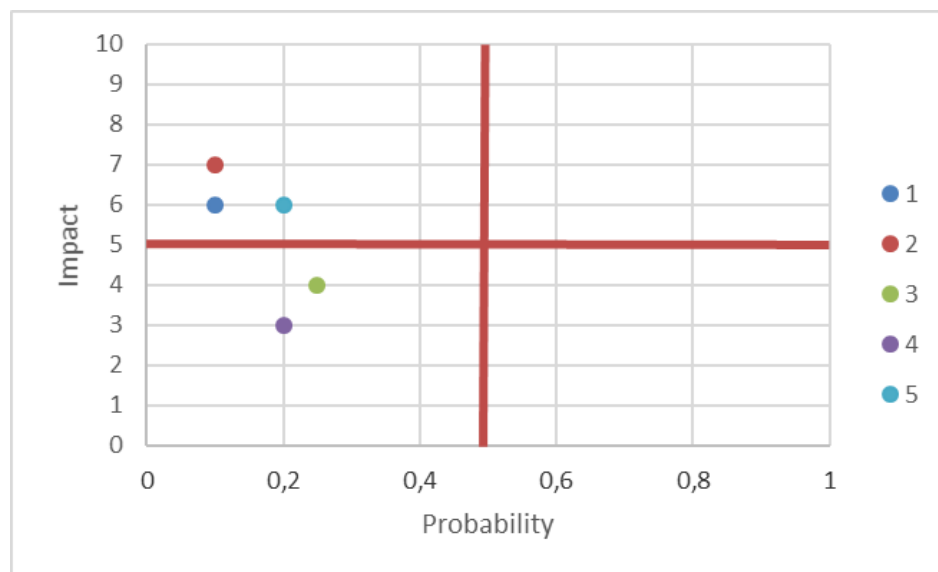


FIGURE C.1: Risk Matrix

C.4 Risk Handling

In order to keep risks to the project on as low a level as possible, the final step of the risk analysis was to define mitigation and contingency actions for each risk. Table C.3 lists both for each risk.

TABLE C.3: Risk Handling

| # | Mitigation | Contingency |
|---|--|--|
| 1 | Create detailed and exhaustive project planning in advance | Implement most important requirements first |
| 2 | Disallow requirements in need of non-existent functionality | Attempt to replace missing functionality, otherwise cancel requirement |
| 3 | Carry out exhaustive requirements analysis before implementation start | Add new requirement to list of low-priority requirements |
| 4 | Create detailed and exhaustive project planning in advance | Work on most important requirements first to ensure they are completed |
| 5 | Learn as much as possible about hardware programming and applicable hardware | Implement requirement as far as possible, use workarounds where required |

D Requirements Analysis

REQUIREMENTS ANALYSIS

for the project

*Transmission and Analysis of Vehicle
Telemetry Data using OBD-II
and Cellular Networks*

IAV GmbH

by Nicholas Walter

Gifhorn, June 8, 2018

Table of Contents

| | |
|---|-----------|
| D.1 Introduction | 58 |
| D.1.1 Purpose of the System | 58 |
| D.1.2 Objectives | 58 |
| D.1.3 Scope | 58 |
| D.2 Proposed System | 60 |
| D.2.1 Overview | 60 |
| D.2.2 Requirement Analysis | 60 |
| D.2.3 Functional requirements | 61 |
| D.2.4 Non-Functional requirements | 65 |
| D.2.5 Requirements Prioritisation | 66 |
| D.3 Revision History | 67 |

D.1 Introduction

This document describes the requirements analysis carried out for the project "Transmission and Analysis of Vehicle Telemetry Data Using OBD-II and Cellular Networks". It briefly describes the project's goals and the system to replace. Afterwards, it lists functional and non-functional requirements and plots them in a value / cost prioritisation.

D.1.1 Purpose of the System

The system's purpose is to transmit telemetry data from an arbitrary OBD-compatible vehicle to a remote server. This allows for real-time analysis of this data, enabling designers and developers to identify flaws and failures as well as successes immediately.

D.1.2 Objectives

The project's objective is to program an Arduino-based OBD-II dongle to read data from the vehicle it is connected to and transmit it to the previously mentioned remote server using the device's ability to create a cellular connection to the internet.

D.1.3 Scope

In order to be able to fulfill the project's requirements to their fullest within the given time frame, its scope must be clearly defined so that feature-creep and bloating do not threaten to sabotage the project. The following subchapters clearly define what is included in and what is excluded from the project scope for the two core objectives.

Software

Included

- Analysis of stakeholders, risks and requirements
- Design of software solution
- Implementation of software solution
- Validation of software solution against requirements and predefined quality goals
- Creation and Maintenance of documents describing results and progress

Excluded

- Maintenance of software solution

Research

Included

- Analysis of posed research question

Excluded

- Implementation of real time data analysis tool (may optionally be considered if time constraints allow it)

D.2 Proposed System

This chapter describes the system that is intended to be implemented.

D.2.1 Overview

The system acts as middle-point between two components that need to be connected: On one side, a vehicle generating telemetry data that needs to be analysed and on the other side a web server that is capable of receiving and handling this data.

D.2.2 Requirement Analysis

The requirement analysis for this project consisted of two phases: Firstly, the requirement elicitation, during which requirements were identified and described and secondly, the evaluation phase, during which requirements were assigned to a priority group based on the value they bring to the project and the cost they come at. The following subsections describe these phases. Appendix D.2.3 and Appendix D.2.4 show the final results of all phases.

Requirement Elicitation

In order to identify the requirements describing this project and its deliverables, a number of well-known techniques were used for requirement elicitation. At the very beginning, the project's goals were gathered from the customer by holding an interview to describe the project's main goals. The interviews results were supported by analysis of an exemplary scenario in which the product could be used.

After some time had been spent on other analyses of the project, another interview was held to confirm the found requirements and identify additional goals.

Appendix D.2.3 and Appendix D.2.4 list the identified requirements.

Requirement Evaluation

After requirements were identified, the next step was to evaluate them for the value they would bring to the project as well as the costs they would require to meet. Since there was no measurable value to be found in any of the requirements, a comparison matrix was used to identify the relative value of requirements compared to each other. Furthermore, because of the projects small scale and limited time frame, instead of cost, the number of days the implementation would take was estimated. Based on this information, a cost/ value graph was created, plotting requirements' value over their cost and assigning them them to high, medium and low priority groups. Figure D.1 shows the result of this evaluation: the x-axis shows the requirements' cost in days of work, the y-axis visualises the value added to the project in percent.

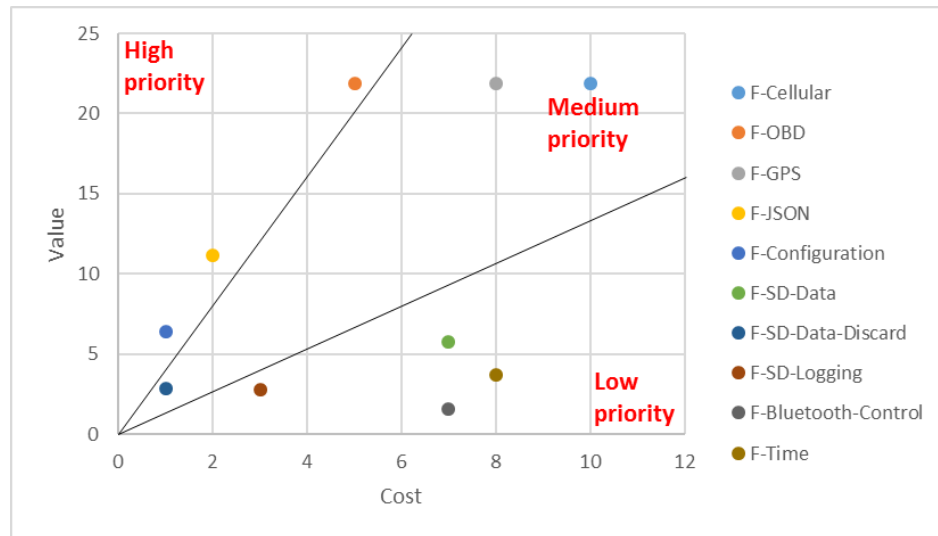


FIGURE D.1: Cost/ Value Graph of Project Requirements

D.2.3 Functional requirements

The following subsections list the project's functional requirements. Each requirement was assigned a unique identifier and a priority group. A description and a definition how the success of the requirement can be measured have been added as well. Finally, the date each requirement added to the list as well as the source it has been obtained from were identified.

Requirement F-Cellular

TABLE D.1: Requirement F-Cellular

| Requirement ID | F-Cellular | Type: | Functional |
|----------------|--|---------|---------------------|
| Description | The product will use cellular connections to transmit collected data to a defined remote server | | |
| Rationale | All analyses of vehicle data collected happen on this remote server, by using cellular connections, no external device needs to be set up to start data transmission | | |
| Measurement | - | | |
| Priority | Medium | | |
| Date created | 2018-03-05 | Origin: | Interview Arnd Eden |

Requirement F-OBD

TABLE D.2: Requirement F-OBD

| | | | |
|----------------|--|---------|---------------------|
| Requirement ID | F-OBD | Type: | Functional |
| Description | The product should read all available data from the vehicle's OBD-II port | | |
| Rationale | Because the analysis server is set up to receive arbitrary data, future-proofing by sending all available data saves future development time | | |
| Measurement | A remote server will be set up to receive data. All incoming data will be checked for completeness | | |
| Priority | High | | |
| Date created | 2018-03-05 | Origin: | Interview Arnd Eden |

Requirement F-GPS

TABLE D.3: Requirement F-GPS

| | | | |
|----------------|--|---------|---------------------|
| Requirement ID | F-GPS | Type: | Functional |
| Description | The product should gather and transmit GPS data, linking it to data gathered in fulfillment of requirement F-OBD | | |
| Rationale | GPS allows for analysis of data based on time and location | | |
| Measurement | A remote server will be set up to receive data. All incoming data will be checked for completeness | | |
| Priority | Medium | | |
| Date created | 2018-03-05 | Origin: | Interview Arnd Eden |

Requirement F-JSON

TABLE D.4: Requirement F-JSON

| | | | |
|----------------|---|---------|---------------------|
| Requirement ID | F-JSON | Type: | Functional |
| Description | The product should encode the gathered information using JSON in a predefined manner (see Appendix H for an exact formatting example) | | |
| Rationale | Because the remote server's location may change at any point in time, it should not be a large effort to reconfigure this setting in the product. | | |
| Measurement | Two remote servers will be set up to test this. Only one file should may be changed to configure the device to reach either of them | | |
| Priority | High | | |
| Date created | 2018-03-13 | Origin: | Interview Arnd Eden |

Requirement F-Time

TABLE D.5: Requirement F-Time

| Requirement ID | F-Time | Type: | Functional |
|----------------|--|---------|---------------------|
| Description | The product should make use of an internet connection to get the current world time via any protocol at system startup, then save it and update it based on its internal microsecond counter | | |
| Rationale | Because the product will be run on a microcontroller that has no functionality to get the current time or save it between power cycles, current time needs to be got on system startup | | |
| Measurement | - | | |
| Priority | Low | | |
| Date created | 2018-03-13 | Origin: | Interview Arnd Eden |

Requirement F-Configuration

TABLE D.6: Requirement F-Configuration

| Requirement ID | F-Configuration | Type: | Functional |
|----------------|---|---------|---------------------|
| Description | The analysis server location should be easily configurable | | |
| Rationale | Because the remote server's location may change at any point in time, it should not be a large effort to reconfigure this setting in the product. | | |
| Measurement | Two remote servers will be set up to test this. Only one file should may be changed to configure the device to reach either of them | | |
| Priority | High | | |
| Date created | 2018-03-13 | Origin: | Interview Arnd Eden |

Requirement F-SD-Data

TABLE D.7: Requirement F-SD-Data

| Requirement ID | F-SD-Data | Type: | Functional |
|----------------|--|---------|-------------------|
| Description | The product should log all data to an SD-card for safe-keeping | | |
| Rationale | In case of signal loss, more data can be saved to an SD-card than to internal memory. Upon signal regain, this data can be transmitted | | |
| Measurement | SD-card logging will be verified by examining the contents of the SD-card after a test run | | |
| Priority | Low | | |
| Date created | 2018-03-13 | Origin: | Scenario Analysis |

Requirement F-SD-Data-Discard

TABLE D.8: Requirement F-SD-Data-Discard

| | | | |
|----------------|---|---------|-------------------|
| Requirement ID | F-SD-Data-Discard | Type: | Functional |
| Description | The product should discard the oldest data on the SD-card when no more space is available | | |
| Rationale | If signal remains unavailable for an extended amount of time, the SD-card will not be able to hold infinite data. Therefore the oldest data should be deleted to make room for new data | | |
| Measurement | SD-card logging will be verified by examining the contents of the SD-card after a test run | | |
| Priority | Medium | | |
| Date created | 2018-03-13 | Origin: | Scenario Analysis |

Requirement F-SD-Logging

TABLE D.9: Requirement F-SD-Logging

| | | | |
|----------------|--|---------|-------------------|
| Requirement ID | F-SD-Logging | Type: | Functional |
| Description | The product should store system logs on the SD-card | | |
| Rationale | For both debugging of the product and vehicle error identification, it is useful to know at what time system events such as signal loss, signal regain etc. occurred | | |
| Measurement | SD-card logging will be verified by examining the contents of the SD-card after a test run | | |
| Priority | Low | | |
| Date created | 2018-03-13 | Origin: | Scenario Analysis |

Requirement F-Bluetooth-Control

TABLE D.10: Requirement F-Bluetooth-Control

| | | | |
|----------------|--|---------|-------------------|
| Requirement ID | F-Bluetooth-Control | Type: | Functional |
| Description | The product will be controllable using Bluetooth connections from e.g. a smartphone | | |
| Rationale | Bluetooth controls would allow users to turn data logging off or put the device into low-power mode without physical interaction | | |
| Measurement | - | | |
| Priority | Low | | |
| Date created | 2018-03-13 | Origin: | Scenario Analysis |

D.2.4 Non-Functional requirements

The following subsections list non-functional requirements. Because they do not add a specific value to the product, they have not been considered in the cost/ value analysis and therefore have not been assigned to a priority group. However, it is important to keep them in mind when implementing functional requirements.

Requirement NF-HTTPS

TABLE D.11: Requirement NF-HTTPS

| | | | |
|----------------|--|---------|---------------------------|
| Requirement ID | NF-HTTPS | Type: | Non-Functional / Security |
| Description | The product will use HTTPS to encrypt the transmission of vehicle data | | |
| Rationale | Sensitive Vehicle Telemetry needs to be protected from attackers & spies | | |
| Measurement | - | | |
| Date created | 2018-03-05 | Origin: | Interview Arnd Eden |

Requirement NF-Performance

TABLE D.12: Requirement NF-Performance

| | | | |
|----------------|--|---------|------------------------------|
| Requirement ID | NF-Performance | Type: | Non-Functional / Performance |
| Description | The product should read and transmit data at the highest rate possible, but at least once per second | | |
| Rationale | The product's purpose is to enable real-time monitoring, therefore a delay of more than one second is not acceptable | | |
| Measurement | Performance measurement will be added to the product's source code | | |
| Date created | 2018-03-13 | Origin: | Interview Arnd Eden |

Requirement NF-Extendibility

TABLE D.13: Requirement NF-Extendibility

| | | | |
|----------------|---|---------|-------------------------------|
| Requirement ID | NF-Extendibility | Type: | Non-Functional / Code Quality |
| Description | The product should be easily extendable by adding more sensors or more data handling procedures | | |
| Rationale | The project will not result in a completely finished product but instead in a prototype which may require changes and additions | | |
| Measurement | - | | |
| Date created | 2018-03-13 | Origin: | Interview Arnd Eden |

Requirement NF-Flexibility

TABLE D.14: Requirement NF-Flexibility

| Requirement ID | NF-Flexibility | Type: | Non-Functional / Code Quality |
|----------------|---|---------|-------------------------------|
| Description | The product should be able to continue working if environment parameters change, such as loss of cellular connection or unavailability of GPS signal | | |
| Rationale | The availability of data sources and handling targets cannot be guaranteed for the entire run time duration, therefore the product must be able to cope if any number of these are expectedly or unexpectedly unavailable | | |
| Measurement | - | | |
| Date created | 2018-03-13 | Origin: | Interview Arnd Eden |

D.2.5 Requirements Prioritisation

Although each requirement's priority was identified earlier, by the customers wishes the requirements F-Cellular, F-OBD and F-GPS will be implemented first regardless of the priority they were assigned. Furthermore these requirements will suffice to consider the project as finished while all other requirements will be considered as additional features of the product.

D.3 Revision History

TABLE D.15: Revision History

| Date | Changes |
|------------|-------------------|
| 2018-03-16 | Original Document |

E Quality Management Plan

QUALITY MANAGEMENT PLAN

for the project

*Transmission and Analysis of Vehicle
Telemetry Data using OBD-II
and Cellular Networks*

IAV GmbH

by Nicholas Walter

Gifhorn, June 8, 2018

Table of Contents

| | | |
|------------|---|-----------|
| E.1 | Introduction | 70 |
| E.1.1 | Referenced Documents | 70 |
| E.1.2 | Management | 70 |
| E.2 | Documentation | 71 |
| E.3 | Standards, Practices and Quality Control | 72 |
| E.3.1 | Standards | 72 |
| E.3.2 | Practices | 72 |
| E.3.3 | Quality Control | 72 |
| E.4 | Quality Control Measurements | 75 |
| E.4.1 | Measurements: 2018-03-31 | 75 |
| E.4.2 | Measurements: 2018-04-14 | 75 |
| E.4.3 | Measurements: 2018-04-28 | 76 |
| E.4.4 | Measurements: 2018-05-12 | 77 |
| E.5 | Revision History | 78 |

E.1 Introduction

This document is part of the project "Transmission and Analysis of Vehicle Telemetry Data Using OBD-II and Cellular Networks". It describes quality standards the project shall adhere to and how they will be met. It defines activities to assure and control quality throughout the project. Furthermore it will list comparisons between actual measurements and quality targets for each point.

E.1.1 Referenced Documents

This document is loosely based on the IEEE Standard for Software Quality Assurance Plans (see IEEE, 1998). Due to the minimal scope and duration of the project, a full implementation of the document was deemed unnecessary and unlikely to have a net positive influence on product quality and implementation time. An old version of this standard was used as newer versions are not freely available.

E.1.2 Management

This section describes the organisation in which the project will be carried out as well as the tasks that need to be carried out in order to achieve a high quality product. Due to the project's team size of a single person, responsibilities do not need to be divided up.

Organisation

The project will be carried out in the company IAV, more specifically in the department Vehicle IT, whose responsibility is the development of software solutions in and around cars.

Tasks

In order to maintain a high standard of quality throughout the project, a number of tasks are defined that need to be carried out:

- Definition of Key Quality Metrics
- Initial measurement of metrics
- Repeated measurement of metrics to track progress
- Analysis of progress based on measurements

While the first two tasks, the definition and initial measurement, only need to be carried out once at the beginning of the implementation phase, the other two tasks are intended to be carried out at the end of every second week: This way, quality changes can be identified and analysed in order to facilitate improvements to the product.

E.2 Documentation

This chapter describes the sources from which the quality metrics can be drawn.

The following documents are of relevance to the quality management:

- Project Plan (see Appendix A)
- Requirements Analysis (see Appendix D)
- Software Design Document (see Appendix F)

The project plan defines the project's scope while the requirements analysis lists the requirements on which all quality metrics need to be based. The Software Design Document describes how the goals set as requirements are realised.

E.3 Standards, Practices and Quality Control

This chapter describes the standards and practices that should be adhered to during the project execution as well as the metrics based on which quality will be measured during and after the project execution.

E.3.1 Standards

This section defines the standards of quality the product shall adhere to in terms of standard procedures:

- All source code shall be documented with a short info text about parameters, functionality and return values for each function/ class method
- Since there is no global standard C++ style guideline and no company-wide equivalent, none shall be given for this project. Instead, all code should be internally consistent in its layout and style choices

E.3.2 Practices

This section defines practices to adhere to that help to maintain high quality standards:

- All implemented code should be version controlled using the project's git repository
- Changes to code should be committed and pushed to the repository on predefined triggers:

Completed implementation of a chunk of code such as a class, a function or a module

Completed refactoring step of any code chunk

- Commit messages should be meaningful and explain the changes made
- Code chunks that cannot be tested with unit tests, such as hardware interaction, should be tested manually in regular intervals and upon any change

E.3.3 Quality Control

This section defines quality metrics the product should adhere to as well as the interval in which measurements should be taken. It also gives a guideline on how to improve quality of the product based on the measurements taken and how they relate to the target metrics.

Metrics

Because the project's aim is to provide a tool that is suitable for the real-time measurement and analysis of data, the most important metrics identified in this document are concerned with performance and stability. Additionally, usability and configurability play a big role.

In Table E.1, the project's key performance metrics are defined by an ID and a description as well as the target performance indicator and, where applicable, the source of the metric.

TABLE E.1: Quality Target Metrics

| Metric ID | Description | Target Value | Perfect Value | Source |
|--------------------------|--|--------------|---------------|----------------------------|
| Performance-Reading | The time in milliseconds one iteration of data reading takes (rounded to next 50) | 1000 | 100 | Requirement NF-Performance |
| Performance-Transmission | The amount of time in milliseconds required to go through one iteration of handling gathered data (rounded to next 50) | 1000 | 100 | Requirement NF-Performance |
| Usability-Startuptime | The maximum amount of time in seconds the product may require to be fully started up and ready for operation | 180 | 30 | - |

Quality Measurement Interval

In order to track the improvement or degradation of quality based on the measurement of key metrics, these measurements need to be taken in a regular interval. Because of the project's small scale a weekly measurement is assumed to yield insufficient new data, therefore the interval of two weeks, starting on 2018-03-31, was chosen.

Quality Analysis

In order to identify the origin of quality improvements and reductions, all changes that were made to the implementation between measurements should be summarised after each measurement cycle. Furthermore, a normalise quality score should be calculated from the measured value and the two target values using the two formulas below where S represents the score, V is the measured value, Vt the target value and Vp the perfect value.

If V is smaller (meaning better) than Vt: $S = (V - Vt)/(Vp - Vt)$

If V is larger (meaning worse) than Vt: $S = V/Vt$

These formulas would result in a number between 0 and 1 if the measured value was smaller (meaning better) than the target value and larger than 1 if the measured value was larger (meaning worse) than the target value.

In terms of quality, this means that all scores that are smaller than 1 fulfill the quality goal. If a score was smaller than 0, this would mean that the measured value exceeds even the perfect target.

Quality Improvement

In order to make use of the quality measurements taken, the next step will be to analyse their results both in absolute perspective and in comparison to the prior intervals' results so that

changes in the codebase can more easily be linked to increases or decreases in quality. Based on this link, a decision can then be made whether a change should persist or be rolled back.

Additionally, at the end of the project, all measurements should be combined into a graph to easily identify the changes in quality over time.

E.4 Quality Control Measurements

This chapter uses the quality metrics defined in Appendix E.3.3 and compares them to measures taken from the actual product. Each section describes one set of results taken according to the schedule defined in Appendix E.3.3.

E.4.1 Measurements: 2018-03-31

This section describes and analyses quality measurements taken on 2018-03-31.

Measurements

Table E.2 compares the target and measured values of all quality metrics and gives a short analysis of the relation between the values.

TABLE E.2: Quality Target Metrics Measurements: 2018-03-31

| Metric ID | Target Value | Perfect Value | Measured Value | Score | Analysis |
|--------------------------|--------------|---------------|----------------|-------|--|
| Performance-Reading | 1000 | 100 | 150 | 0.06 | Very good score, but some data has yet to be read, therefore this number may still worsen |
| Performance-Transmission | 1000 | 100 | 100 | 0 | Perfect score, but cellular transmission is not enabled yet, this is deemed to be the most impactful handling step |
| Usability-Startuptime | 180 | 30 | 50 | 0.13 | Good score, several debugging functions still enabled that could have an impact |

Analysis

Based on the measured values for the key performance indicators, the project is on a good track towards success: All values are within the target range, one even reached the perfect score. However since a good amount of functionality is still missing or incomplete, changes to these measurements need to be expected.

E.4.2 Measurements: 2018-04-14

This section describes and analyses quality measurements taken on 2018-04-14.

Measurements

Table E.3 compares the target and measured values of all quality metrics and gives a short analysis of the relation between the values.

TABLE E.3: Quality Target Metrics Measurements: 2018-04-14

| Metric ID | Target Value | Perfect Value | Measured Value | Score | Analysis |
|--------------------------|--------------|---------------|----------------|-------|---|
| Performance-Reading | 1000 | 100 | 150 | 0.06 | Very good score, but not all data has yet to be read, therefore this number may still worsen; performance was improved by reading GPS data in chunks instead of one at a time |
| Performance-Transmission | 1000 | 100 | 100 | 0 | No change to previous implementation |
| Usability-Startuptime | 180 | 30 | 60 | 0.2 | Additional logging functionality had an impact on startup time: SD card handler has to be initialised, mutexes declared |

Analysis

Slight improvements to quality measurements were made by streamlining reading and startup procedures. Upon removal of debug functionality, startup time is expected to decrease; implementing the final missing functionality is expected to decrease the reading rate.

E.4.3 Measurements: 2018-04-28

This section describes and analyses quality measurements taken on 2018-04-28.

Measurements

Table E.4 compares the target and measured values of all quality metrics and gives a short analysis of the relation between the values.

TABLE E.4: Quality Target Metrics Measurements: 2018-04-28

| Metric ID | Target Value | Perfect Value | Measured Value | Score | Analysis |
|--------------------------|--------------|---------------|----------------|-------|---|
| Performance-Reading | 1000 | 100 | 100 | 0 | Implementation was not changed but libraries were updated |
| Performance-Transmission | 1000 | 100 | 150 | 0.06 | now opening file on SD card on demand instead of using file opened on setup |
| Usability-Startuptime | 180 | 30 | 45 | 0.1 | removed SD initialisation; SD libraries do not allow for multiple files to be opened → now opening them on demand |

Analysis

Because the SD library does not allow for multiple files to be opened and written into simultaneously, they will now be opened on demand. This improved the startup time but had a negative impact on handling times. If a work around can be found, the best of both worlds can maybe be combined.

E.4.4 Measurements: 2018-05-12

This section describes and analyses quality measurements taken on 2018-05-12. As per the project planning, this will be the last intermediate measurement taken before the project is concluded.

Measurements

Table E.5 compares the target and measured values of all quality metrics and gives a short analysis of the relation between the values.

TABLE E.5: Quality Target Metrics Measurements: 2018-04-28

| Metric ID | Target Value | Perfect Value | Measured Value | Score | Analysis |
|--------------------------|--------------|---------------|----------------|-------|--|
| Performance-Reading | 1000 | 100 | 100 | 0 | No changes |
| Performance-Transmission | 1000 | 100 | 200 | 0.11 | Implemented transmission of data to remote server (though unencrypted) |
| Usability-Startuptime | 180 | 30 | 40 | 0.07 | Disabled all debugging helps in the code |

Analysis

Between the previous and this last intermediate measurement, the transmission of data from the device to a remote server was implemented, although HTTPS cannot be used, therefore the transmission is unencrypted. This is not estimated to make a difference in terms of performance as encryption happens on the cellular module. This reduced the number of times per second data can be handled though the value is still very much acceptable. Startup time was, as expected, improved slightly by disabling debugging functionality which contains a number of somewhat time-expensive system status requests to the cellular module.

E.5 Revision History

TABLE E.6: Revision History

| Date | Changes |
|------------|--------------------|
| 2018-05-12 | Added measurements |
| 2018-04-28 | Added measurements |
| 2018-04-14 | Added measurements |
| 2018-03-31 | Added measurements |
| 2018-03-26 | Original Document |

F Software Design Document

SOFTWARE DESIGN DOCUMENT

for the project

*Transmission and Analysis of Vehicle
Telemetry Data using OBD-II
and Cellular Networks*

IAV GmbH

by Nicholas Walter

Gifhorn, June 8, 2018

Table of Contents

| | | |
|------------|---|-----------|
| F.1 | Introduction | 81 |
| F.2 | System Overview | 82 |
| F.2.1 | Project Purpose | 82 |
| F.2.2 | Project Design Aims | 82 |
| F.2.3 | System Environment and Components | 82 |
| F.2.4 | Freematics ONE+ | 83 |
| F.3 | System Architecture | 85 |
| F.3.1 | System modules | 85 |
| F.3.2 | Module Design | 85 |
| F.3.3 | Task Design | 92 |
| F.4 | Test Design | 95 |
| F.4.1 | Development Approach | 95 |
| F.4.2 | Test Design | 95 |

F.1 Introduction

This document belongs to the project "Transmission and Analysis of Vehicle Telemetry Data Using OBD-II and Cellular Networks" and defines it with focus on the product's software design. Chapter F.2 gives a rough overview over the project's purpose and its basic design. Chapter F.3 explains the product's architecture first from a high-level perspective and then in more detail by looking at the individual modules while section F.4 defines and explains the project's test strategies.

F.2 System Overview

F.2.1 Project Purpose

The project's aim is to create a software product to run on an arduino-based OBD-II dongle in order to read telemetry data from a vehicle and transmit it to a webserver for analysis. More detailed information about this can be gathered from the project plan created for this project.

F.2.2 Project Design Aims

Based on the requirements identified in Appendix D, a list of design aims and constraints was compiled:

Design Aims

- **Ease of implementation:** Where possible, design choices should be made that make the development and later maintenance of the software easier
- **Extensibility:** The software should be created with future expansions in mind. Specifically, additional sensors or methods of handling data could be added
- **Flexibility:** The software should be able to react to environment and system changes flexibly: For example, some data source may suddenly become unavailable or connection to cellular networks could be lost
- **Performance:** All functionality should be designed and implemented with the thought of achieving reading and transmission rates as high as possible (see Appendix D for details)

Design Constraints

- **Available flash memory:** The ESP32 chip has 4 Megabytes of memory available for program space and 520 Kilobytes of RAM (see Freemantics, No date(b))
- **Available processing power:** The ESP32 chip is based on a dual core processor that runs at a frequency of 240 MHz (see Freemantics, No date(b))

All design decisions were made with these aims and constraints as a basis.

F.2.3 System Environment and Components

As part of the design process, the system was broken down into individual relationships between components in order to identify components of the system interacting with each other as well as outside actors:

- The driver drives the vehicle
- The test driver connects the device.
- The device reads data from the vehicle.

- The device transmits data to the remote analysis server.
- The remote analysis server analyses data items.
- The device briefly analyses data items.

By analysing these relationships for their noun phrases, a domain model was created. Figure F.1 shows the domain model.

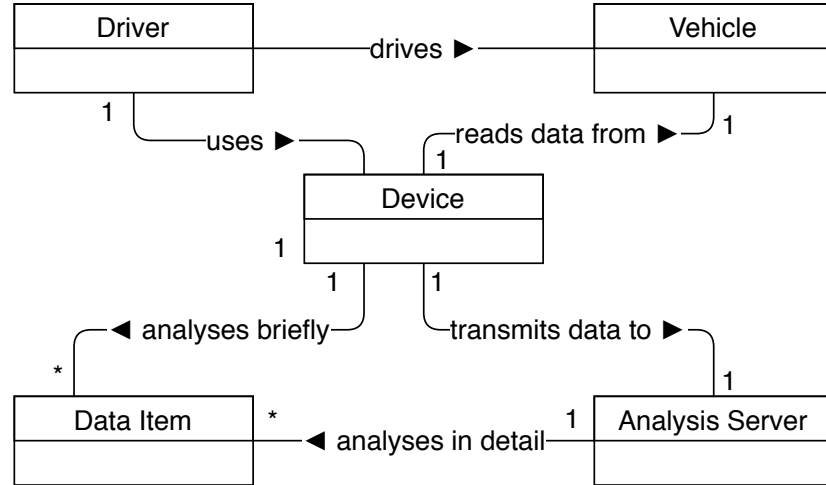


FIGURE F.1: Product Domain Model

The domain model helped to understand the relationships between all components and actors in the system so that later design decisions could be made more easily.

F.2.4 Freematics ONE+

The Freematics ONE+ is based on an ESP32 microcontroller, giving it WiFi and bluetooth connectivity and a dual-core processor so that two tasks (such as reading and handling data) can be executed simultaneously (see Espressif Systems, 2018). It is also equipped with a SIM module, allowing for data transfer over cellular networks and a GPS-module to acquire location and movement data (see Freematics, No date(b)). Figure F.2 visualises the device's components, their rough tasks and interactions.

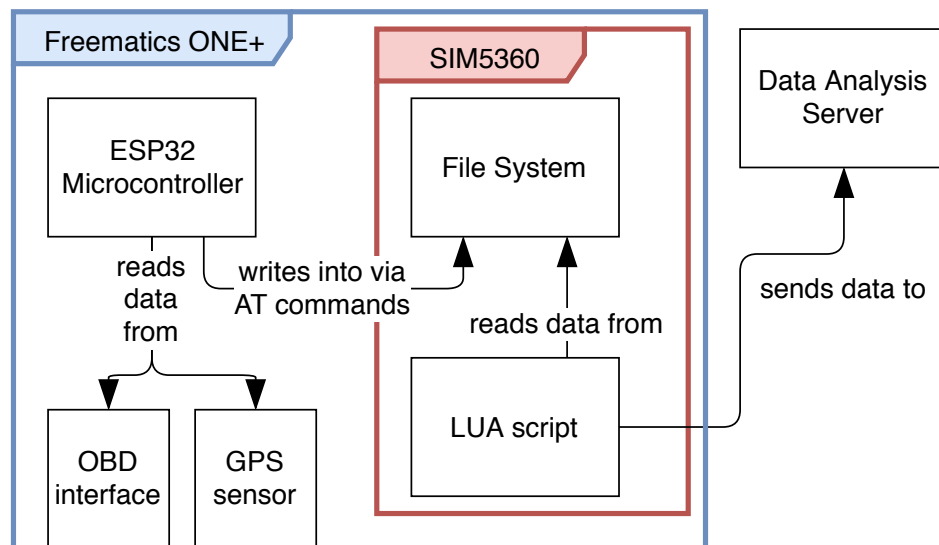


FIGURE F.2: Product Architecture

F.3 System Architecture

This chapter shall describe the product's system architecture in detail by going over all decisions that were made regarding the design of the system overall and of individual modules as well as the factors that went into the choices made and their results.

F.3.1 System modules

At the very beginning of the design process, before designing individual functionalities, the system's overall structure had to be decided upon.

Although it would have been possible to implement the desired functionality using a very simple design such as reading and encoding data in a single function, then transmitting it in another, effectively only creating a single source code file, this approach was abandoned because it was not in line with the previously defined design goals: Extending and maintaining this code would be difficult and require considerable effort.

Instead, the decision was made to split the system up into modules, each of which would fulfill a certain task without necessarily knowing about the others' implementations.

The program created will consist of three major modules:

- DataReading, which is responsible for gathering the desired data (initially vehicle telemetry and GPS data) from the sensors attached to the device.
- DataKeeping, containing functionality to store data for a short time
- DataHandling, where classes handle incoming data and prepare it for transmission, analysis or other methods

Additionally, three support modules were designed:

- ConnectionHandling, which contains functionality of transmitting data over various interfaces such as cellular networks, WLAN, Bluetooth; also houses SD card writing functionality
- Logging, a module which centralizes all system logging: debugging and status messages as well as warning and error notifications
- Util, where all functionality is housed that does not fit in any of the other modules: distribution of constant variables, time keeping, etc.

Figure I.1 shows the class diagram resulting from this module design. Appendix F.3.2 explains the decisions behind the individual modules' design.

F.3.2 Module Design

This section describes the decision making process behind the design choices for the product's individual software modules ¹.

As Figure I.1 visualises and as described in Appendix F.3.1, the system is divided into six major components, each of which contains a set of tasks related to each other.

¹In order to reduce the figures' size and improve readability, the display of trivial members, functions and connections has been omitted.

Inter-Module Communication

Although the design decision made regarding modules in Appendix F.3.1 included that all modules should, where possible, be separate from each other, an important aspect of the product would be the communication between some of them, especially between the **DataReading** and **DataHandling** module, each of which only fulfills part of the system's tasks. For this reason, a link between these modules has to be established.

FreeRTOS, the operating system kernel used on the chip the system will be deployed to, offers threadsafe queues that enable communication between two tasks. For this project, a queue is used by the **DataReading** module to push data into so that the **DataHandling** module can access it and deal with it as required.

DataReading Module

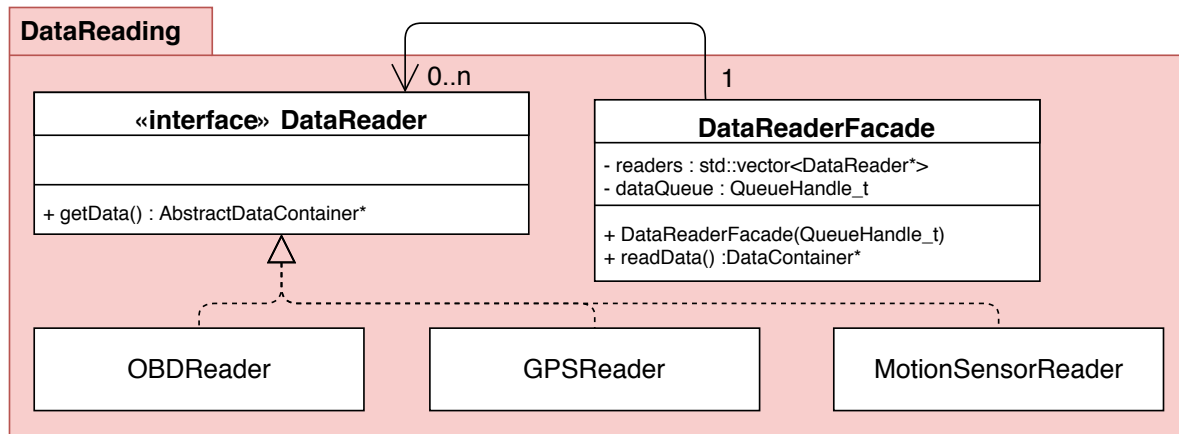


FIGURE F.3: Product Class Diagram: DataReading module

The **DataReading** module is responsible for the generation of data from the various sensors and interfaces available on the device.

Each of the classes responsible for reading one of the sensors extends the **DataReader** interface to allow the **DataReaderFacade** class to use polymorphism to call each sensor without being aware of the specific implementation. Using the Facade pattern in the **DataReaderFacade** class allows clients to interface with the module using only a single method. By passing a **QueueHandle** object, essentially a reference used to deal with FreeRTOS data queues, to the **DataReaderFacade** in its constructor, it is ensured that it can write all results into the queue.

Figure F.4 visualises the process of reading data from sensors and combining it into a **DataContainer** object (see Appendix F.3.2).

DataKeeping Module

The **DataKeeping** module contains data structures to store data for a short period of time. This is necessary because the immediate handling, and especially transmission, of data gathered from sensors and interfaces at desired speeds is assumed to be too work-intensive for the processing cores. Furthermore, this design offers a number of utilities that would otherwise have to be implemented in other places, such as the encoding of data into JSON strings.

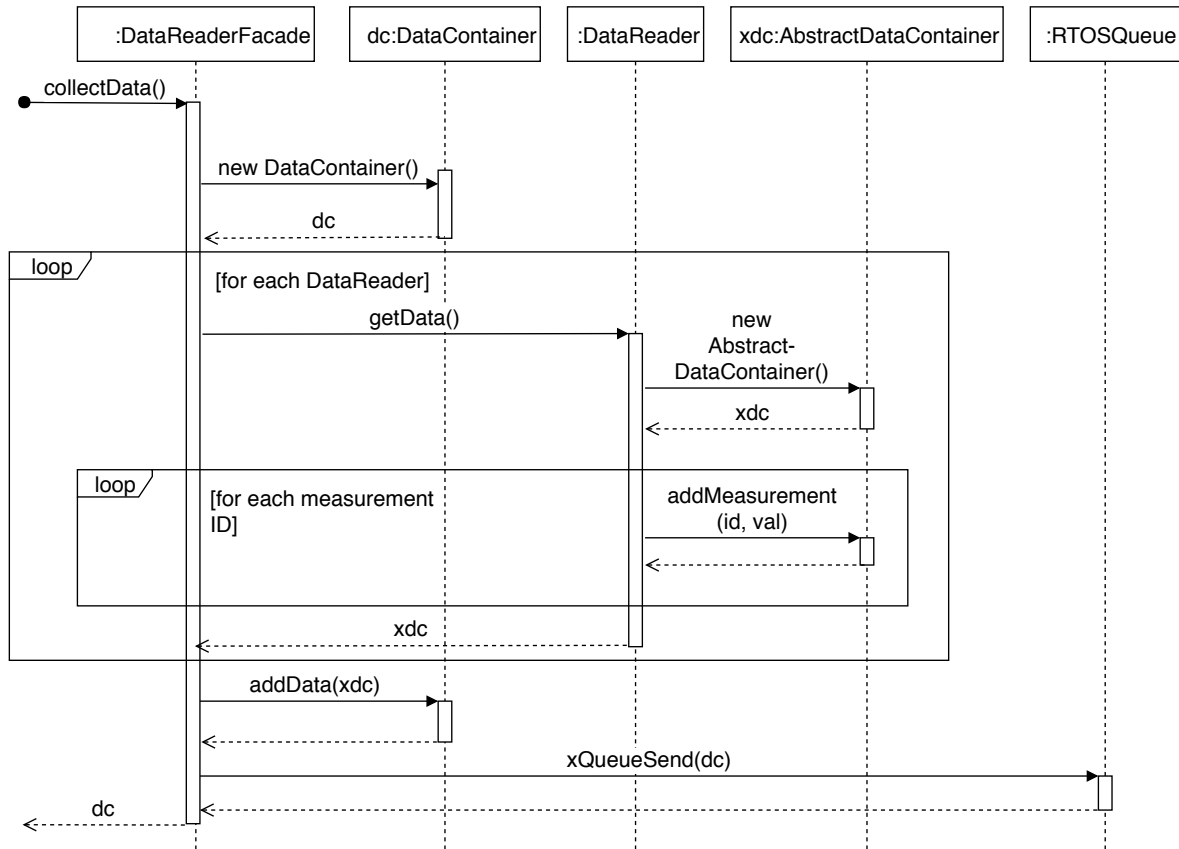


FIGURE F.4: Product Sequence Diagram: Process of reading data from sensors

The basic premise behind the module is that for each sensor reader in the **DataReading** module (see Appendix F.3.2), an appropriate **AbstractDataContainer** subtype exists in the **DataKeeping** module. When reading, each **DataReader** creates its own corresponding **AbstractDataContainer** subtype and fills it with the acquired data. The **DataReaderFacade** class then adds each of them to the base **DataContainer**.

By applying the Composite pattern to this module, it is assured that clients can use any subtype of the **AbstractDataContainer** interface, including the **DataContainer** class, without checking whether all leaf nodes are present. Furthermore it is possible for users of the data to combine multiple **DataContainer** objects into one so that e.g. the **getJSON()** method only needs to be called once instead of one time for each **DataContainer**.

Implementing the **AbstractDataContainer** interface also means that each subtype is able to produce a JSON-representation of itself. In the case of the **DataContainer** class, this means to combine the results gathered from all leaf nodes to a single JSON string.

Figure F.6 visualises the process of encoding all data contained in a **DataContainer** object to a JSON string.

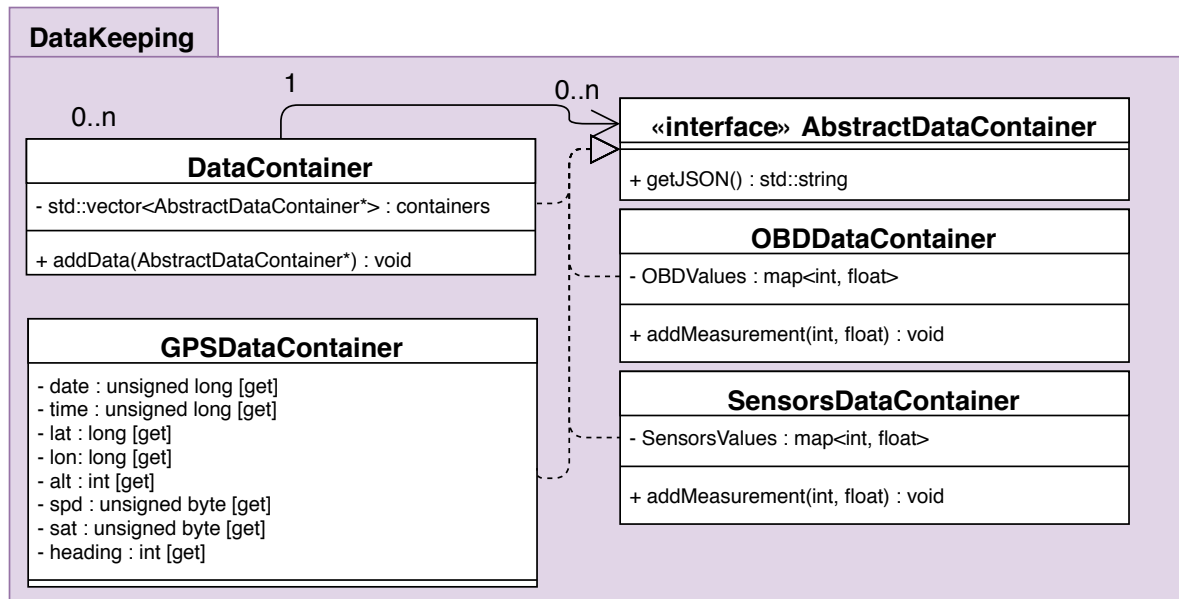


FIGURE F.5: Product Class Diagram: DataKeeping module

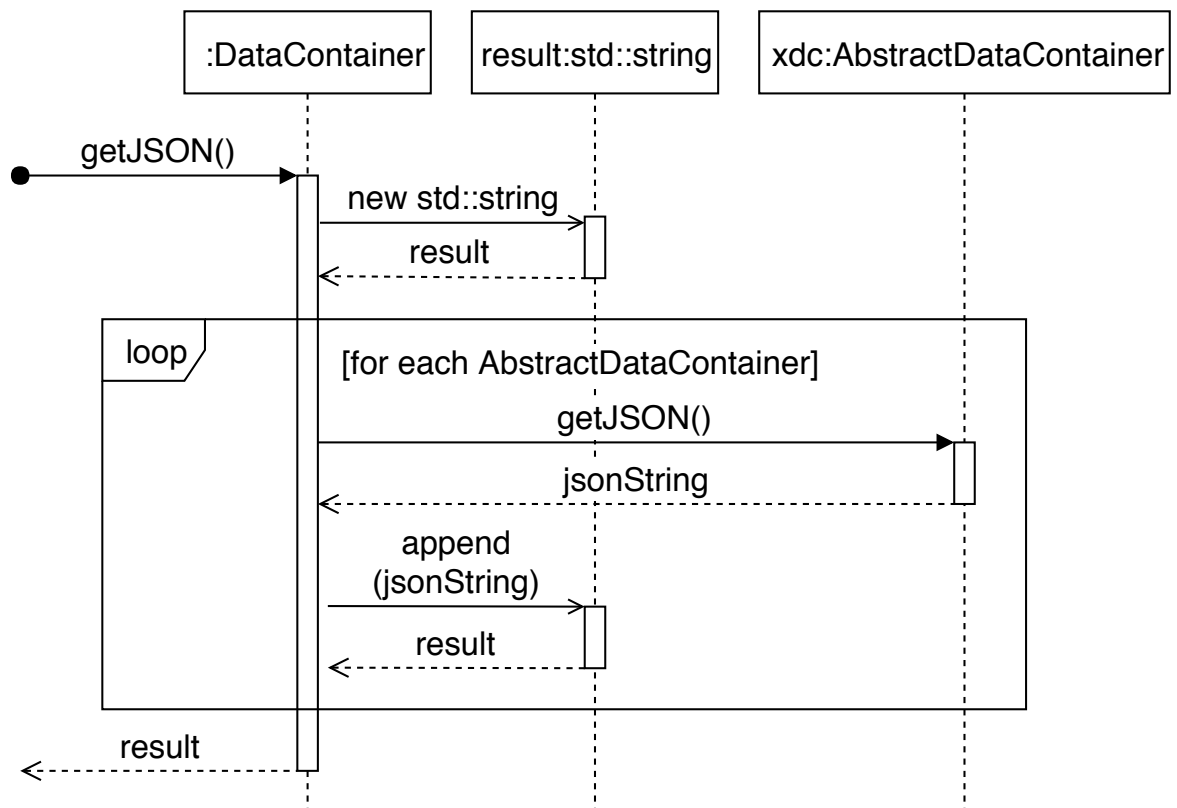


FIGURE F.6: Product Sequence Diagram: Process of encoding data in a DataContainer to JSON

DataHandling Module

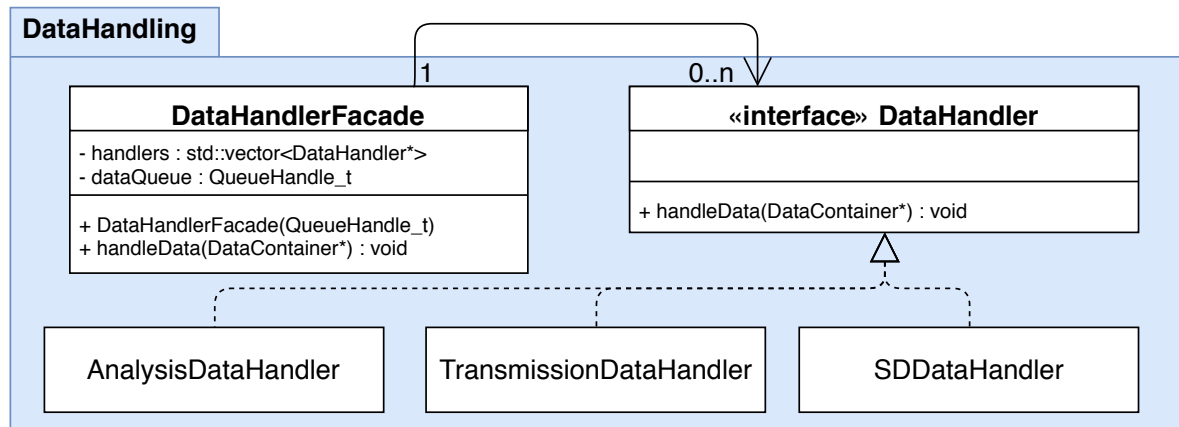


FIGURE F.7: Product Class Diagram: DataHandling module

The **DataHandling** module is structured very similar to its counterpart, the **DataReading** module. Using the Facade pattern makes it very easy for clients to supply data to an arbitrary number of handlers. Because all handlers implement the **DataHandler** interface, the **DataHandlerFacade** class can simply iterate over a list of **DataHandler** objects and call their **handleData()** method instead of knowing about and precisely calling each individual handler. This makes the entire module very easy to extend if future additions to the system require it.

When called, the **DataHandlerFacade** proceeds to supply the given data to each of the **DataHandler** subtypes sequentially for them to handle. With the intention of simplifying memory deallocation, all data handling happens in the same thread.

Each of the handlers uses the data for its own purpose: The **AnalysisDataHandler** carries out a quick analysis and may trigger actions based on its results, the **TransmissionDataHandler** is responsible for the transmission of data to remote servers and devices outside of the product's scope and the **SDDDataHandler** handles the logging of data to local storage.

Figure F.8 visualises the process of handling data contained in a **DataContainer** object (see Appendix F.3.2) on the example of the **TransmissionDataHandler**.

At first, the number of messages (**DataContainer** objects) waiting in the **RTOSQueue** is checked. If several are waiting, a new **DataContainer** object is created into which the others are inserted so that they can all be handled at once. If only one message is waiting, the object is used as is. This is made possible by the application of the Composite pattern in the **DataKeeping** module (see Appendix F.3.2).

Afterwards, the final object is passed to various **DataHandler** subtypes to handle the data. Figure F.8 shows this on the example of the **TransmissionDataHandler**, which passes the JSON string representing the data on to the **CellularHandler** class. Other **DataHandlers** handle data differently.

Finally, the **DataContainer** object is deleted, freeing up memory so that the system can continue its cycle.

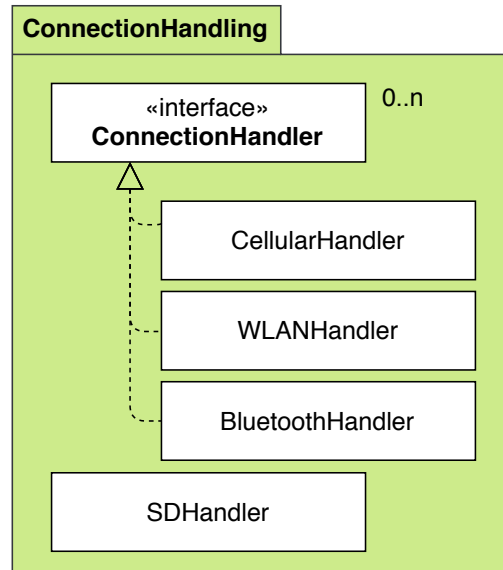


FIGURE F.9: Product Class Diagram: ConnectionHandling module

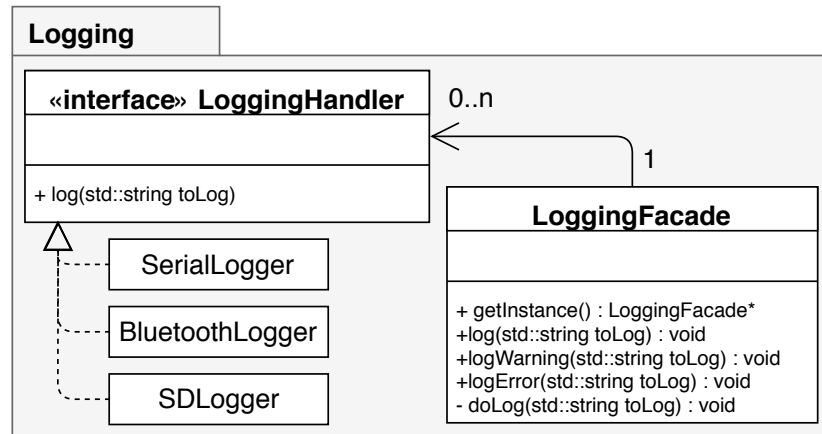


FIGURE F.10: Product Class Diagram: Util module

offers a single interface to write logs to different targets, such as the serial interface, Bluetooth or an SD card.

Util Module

In addition to the previously described main modules, the **Util** module offers functionality that is frequently used elsewhere but does not fit in the domain of any single module: the `config.h` and `constants.h` files contain information that is often required as parameter and does not need to be changed during runtime. This also makes it easier to configure the program at compiletime, since all configurations can be changed in one central point. This has the advantage that in order to change how the system works, no deeper knowledge of the underlying code is required. It also contains functions such as a `toString()` helper and other functionality that can not be associated with only one class or module.

Finally, it also contains the **TimeKeeper** class, which houses the functionality of keeping the system time up to date. Because the microcontroller the device is based on is unable to keep its system time up to date between power cycles, the current time is read from an NTP

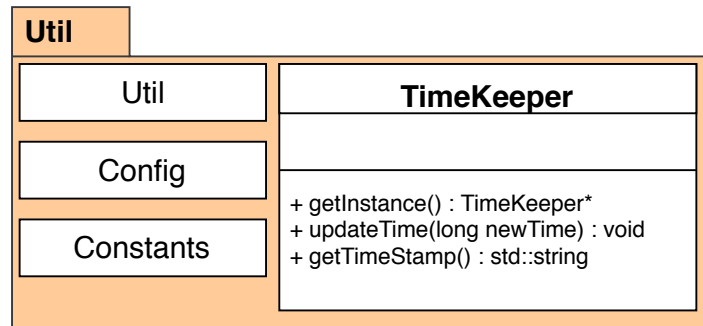


FIGURE F.11: Product Class Diagram: Util module

server upon startup and saved there. Whenever a client calls the `getTimeStamp()` method, the current time is updated based on the milliseconds elapsed since the last update and saved.

F.3.3 Task Design

Because the ONE+’s computing power is very limited, it is important to use what is available efficiently. This section explains the basic structure behind the code that will be deployed to the device and define the individual tasks that will be created in an effort to effectively employ the available resources. Furthermore, a rough attempt at designing threads in terms of priority and execution order will be made.

Task control functionality

By default, the ONE+ comes with FreeRTOS, a real time operating system, pre-installed. FreeRTOS offers functionalities to create threads that can be run independently from each other and simultaneously. It also supports blocking tasks so that those with higher priority can execute while lower-priority tasks yield.

In addition to general task control, it also offers utility functionality such as threadsafe queues that can be used for communication between two tasks (see FreeRTOS, 2017b) or resource access management (see Barry, 2016) as well as functions that delay execution of a task by a number of either milliseconds or processor ticks, depending on configuration, while other tasks are executed (see FreeRTOS, 2017c).

Program structure

Since the project’s intention is to develop an application to run on the ESP32 microcontroller, its structure at the very core consists of the `setup()` and `loop()` functions typical for arduino-like devices (see Arduino, No date(b)). Upon system startup, the `setup()` function is called once: Its responsibility is the set up of all system-relevant modules and components, such as instantiating classes, powering up attached devices etc. Once this function concludes, the `loop()` function will be called in an infinite loop until the device is powered down (see me-no-dev, 2017a).

Because the ESP32, untypically for small-scale microcontrollers, comes with two processing cores, two tasks can run simultaneously. By taking advantage of this fact and dividing the program into two main tasks, performance can be increased. However this comes at the cost of a larger design effort.

For the purpose of the system to be designed and implemented in this project, two main tasks have been identified: the reading of data from sensors and interfaces attached to the device and the handling of this data in several forms. These two tasks have been reduced to the calling of two functions in the `main.cpp` file: `readerLoop()` and `handlerLoop()`. Because the default `loop()` task is also responsible for a number of maintenance functions, such as preventing integer overflow in the timing function `micros()` (see me-no-dev, 2017b), that must be called regularly, these functions had to be extracted and moved to one of the two main tasks. Because the `readerLoop()` is deemed to be less likely to fail or stall during execution, the decision was made to use this instead of the `handlerLoop()`.

The two separated tasks are then connected to each other by using a threadsafe queue as data buffer (see Figure I.1), into which the `readerLoop()` task can write data and the `handlerLoop()` task can extract data from to work on it.

Thread Design

As mentioned in Appendix F.3.3, the products main functionality (reading and handling data) is separated over two threads: While one is responsible for the gathering of data from various sources, the other uses the same data and handles it with various methods. This separation was created in order to fully utilise the performance power offered by the micro-controller's two processing cores.

Furthermore, the decision was made to divide processor time up into chunks. If the task completes before its allocated time slot expires, the task will idle up until the next chunk starts. This way, there is no danger that the reading tasks will run significantly more often than the handling tasks, causing the data queue to fill up and data to get lost.

Figure F.12 visualises how the two tasks are split over the processing cores: After the system has been set up completely, the two tasks are started. While one core starts working on the data reading tasks (green) immediately, the second core idles (white) for slightly more than half of a time chunk so that data can be read and put into the queue. Afterwards, it starts working on the data handling process (red). After they finished their work, both cores wait for the time chunk to be completed before repeating. Since both tasks wait for the same length of time chunk, this procedure means that the data handling core always lags behind the reading core for slightly more than half a chunk. Theoretically, this means that the data queue should never contain more than one `DataContainer` object, given that handling always works and never requires more time than allocated. If, for some reason, the handling task should take longer than anticipated, the reading task will still write into the queue as it normally would: The handling tasks will check on its beginning how many messages are waiting in the queue and, if necessary, combine them to handle them at once (see Appendix F.3.2).

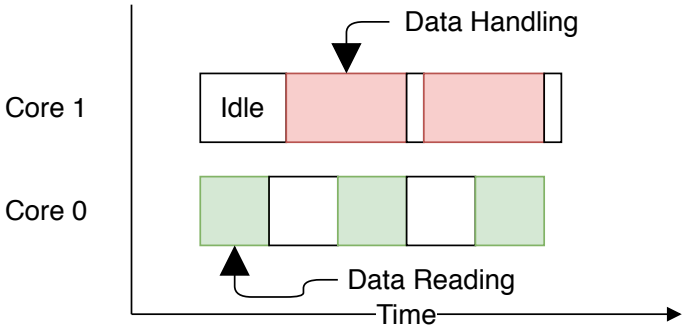


FIGURE F.12: Product Class Diagram: Util module

F.4 Test Design

Because of long deployment times and because debugging functionality is limited to `print()` statements instead of breakpoints and variable inspection at runtime, verifying that code is working as intended on an embedded system such as the one this project is being carried out with takes considerably more effort than it would when developing a desktop application. For this reason, it is especially important to properly design and carry out software tests throughout the implementation and validation phase. This chapter defines the testing approach that will be used during this project.

F.4.1 Development Approach

Given the system's modular design (see section F.3), the decision was made to make use of test-driven development throughout the implementation phase of the project. The basic premise of test-driven development is that, before any actual production code is written, tests for each module and tasks are implemented, which describe a fully functional system. Based on these tests, the system will then be implemented, constantly in an effort to reach 100 per cent success across all tests. However, because many features of the product rely on working closely to the target hardware, it is not possible to test these in an automated fashion. Instead, those features will be tested manually during development.

This has the advantage that results and procedures are clearly defined and development will not be hindered by making important decision on the fly. Additionally, given that tests are run automatically on each new build, any breaking code changes can be identified immediately.

F.4.2 Test Design

Based on the decision to make use of test-driven development, the next step is to define what types of test to use during the implementation and validation phase of the project.

When applying test-driven development, the use of unit tests de facto comes by default, therefore they were applied in this project as well: Each module and each of the contained classes will be described by a number of unit tests so that development can start by implementing the methods and functions they test.

Additionally, the decision was made to carry out real-world tests in a semi-regular schedule: By plugging the target device into a test vehicle and driving for a reasonable amount of time, performance and usability can be tested in a production-like environment.

G Research Report

SOFTWARE FEASIBILITY STUDY REPORT

for the project

*Transmission and Analysis of Vehicle
Telemetry Data using OBD-II
and Cellular Networks*

IAV GmbH

by Nicholas Walter

Gifhorn, June 8, 2018

Table of Contents

| | |
|-------------------------------|------------|
| G.1 Introduction | 98 |
| G.1.1 Context | 98 |
| G.1.2 The Device | 98 |
| G.1.3 Environment | 99 |
| G.2 Target System | 99 |
| G.2.1 Tasks | 99 |
| G.2.2 Restrictions | 100 |
| G.3 Research Execution | 101 |
| G.3.1 Core Questions | 101 |
| G.3.2 Methodology | 102 |
| G.3.3 Results | 102 |
| G.4 Conclusion | 105 |

G.1 Introduction

This document describes the feasibility study carried out as part of the project "Transmission and Analysis of Vehicle Telemetry Data Using OBD-II and Cellular Networks". The aim behind the feasibility study is to determine whether it is possible to use the device used for the gathering and transmission of data during the main project for real-time analysis of the same data in order to identify invalid test runs.

This report should answer the question whether the analysis described above can be implemented as part of the main project or alternatively whether it can be implemented as an additional project. In addition, some general hints towards the implementation should be given, such as where data can be obtained from or how things can be implemented.

G.1.1 Context

The system in question would be used as part of the testing routine for newly developed cars and car parts at IAV. Because new parts need to be tested thoroughly, for both validation and trouble shooting, a considerable amount of effort goes into carrying out test drives with tightly regulated parameters. These parameters can include but are not limited to the distance that should be driven, the speed at which the car should drive, what type of roads etc. Because the number and nature of these parameters can be very complex, it is often difficult for drivers to keep track of their performance: Sometimes completed test drives turn out to be invalid when data is checked. In this case, all effort was for nothing and the test drive needs to be redone.

In order to prevent this, the system proposed should keep track of the parameters and inform drivers about invalidation so that test drives can be aborted early.

G.1.2 The Device

In order to judge whether the proposed addition to the project is feasible, it is important to understand the target system and its capabilities. This section describes the hardware and its capabilities in detail.

The device in question is the Freematics ONE+, an open-source microcontroller-based interface to a vehicles on-board diagnostics (OBD) port. It is based on the ESP32 microcontroller and offers interfaces to cellular network connections and the global positioning system (GPS) in addition to the OBD interface (see Freematics, No date(b)).

Hardware Performance

The microcontroller the device is based on offers two processing cores with a tact frequency of 240MHz as well as 520 kilobyte of RAM (see Freematics, No date(b)).

Hardware Interfaces

The ONE+ is equipped with Bluetooth and WiFi capabilities offered by the ESP32 chip as well as a cellular module and two external I/O ports that are used to connect the GPS antenna

but can be repurposed. Additionally, a micro-SD card can be written to or read from (see Freematics, No date(b)).

G.1.3 Environment

This section describes the environment in which the proposed system would be used.

Vehicle

Since its main purpose will be to analyse vehicle data, the most important aspect of the system's environment is the vehicle it will be deployed in. In all cases relevant to this report, the vehicle will be either a recently developed model or one that is still currently in development. Therefore it can be assumed that the OBD-II protocol is supported.

Available Data

Given that the device is powered over the OBD-II connector, the available data always includes all data that can be read over this interface. This includes basic data like vehicle speed, engine RPM and throttle position but also more obscure data (see Wikipedia, No date).

Additionally, the device's GPS sensor can be used to obtain location information while the cellular module can be used to read basically arbitrary information from various online APIs.

G.2 Target System

This chapter specifies the system for which this feasibility study is being carried out by its desired features, the tasks it will have to fulfill and some restrictions that apply.

G.2.1 Tasks

The basic task the system should carry out is the real-time analysis of incoming vehicle and GPS data in order to identify invalid test runs. In order for this to work, the system should read and keep track of said data, comparing it to target values. The following subsections describe each of the subtasks in more detail.

Reading Data

In order to fulfill its tasks, the first step for the device is to read data. Specifically, the tasks require the availability of position data as well as vehicle data such as speed, engine load, currently selected gear etc. Table G.1 lists all required data.

Information about what data is required was gathered from interviews and inferred from usage examples.

TABLE G.1: Data required to carry out live analysis

| ID | Name | Description |
|----|-------------------------|---|
| 1 | Current Time | The current world time accurate to milliseconds |
| 2 | Engine RPM | Number of revolutions the vehicle's engine is making per minute at this moment |
| 3 | Vehicle Speed | The speed the vehicle is driving at this moment |
| 4 | Engine Load | The amount of power the vehicle's engine is currently putting out towards the wheels |
| 5 | Currently selected gear | The gear selected in the vehicles gearbox at this moment |
| 6 | Location | The global position the vehicle is currently at |
| 7 | Road Type | The type of road the vehicle is currently driving on |
| 8 | External conditions | The atmospheric conditions outside the vehicle at this moment (temperature, barometric pressure, humidity) |
| 9 | Other OBD Data | In addition to engine RPM, vehicle speed and engine load, the OBD interface offers various types of data to connected readers |

Tracking Data

After data has been read from the sensors, the system's task is to keep track of it. This consists not only of storing read values but also calculating others from it. For example, one of the core tasks would be to calculate distance driven from either GPS position markers or vehicle speed and time.

Comparing Data To Target Values

Finally, the tracked data needs to be compared to the target values set at the beginning of the test drive. As soon as one of the parameters set becomes invalid, the system should give notice to the driver.

User Interaction

Because the actual device's only way to directly communicate with a human driver is a small status LED, all user interaction needs to be channelled through an external device with which the system needs to communicate in real time. This external device could then inform the user about the status and (in)validation of parameters. Furthermore it could be used to change configurations or set parameters at the beginning of the test drive.

G.2.2 Restrictions

Of course, there are some restrictions in how the technologies available can be used to implement the set tasks. Firstly, user interaction must be restricted to an absolute minimum as the standard use case is that the user is driving a vehicle which severely restricts how much attention can be put on watching readouts or other information obtained from the system.

Secondly, the device the system would be deployed to is very limited in terms of computing power:

- **Available flash memory:** The ESP32 chip has 4 Megabytes of memory available for program space and 520 Kilobytes of RAM (see Freematics, No date(b))
- **Available processing power:** The ESP32 chip is based on a dual core processor that runs at a frequency of 240 MHz (see Freematics, No date(b))

These hardware restrictions need to be taken into consideration when developing the real-time analysis tool.

G.3 Research Execution

This chapter explains the execution and results of the feasibility study described in the previous chapters by defining the core questions to be answered, the methods used and explaining the final results.

G.3.1 Core Questions

Based on the tasks the system would have to fulfill as defined in Appendix G.2, this section defines the core questions to be answered as part of the feasibility study. Based on whether the questions can be answered affirmatively or some features are not possible to implement, the result will define whether the system can be implemented in this configuration. The following subsections each define one of the core questions and describe it in more detail.

Data Availability

Is it possible to gather all required data from either a sensor, an online API or another source? The data in question is listed in Table G.1.

Data Handling

Is the device capable of handling the acquired data in ways suitable to handle all tasks?

User Interaction

Is there a way in which a user can interact with the device?

Performance

Can the device carry out the required number of data reading / tracking actions?

Integration

Can the system be integrated into the data transmission project as a subsystem?

G.3.2 Methodology

This section describes the methodologies that were used in order to answer the core questions set in Appendix G.3.1.

Since all questions are based on whether or not the device is capable of fulfilling a given functionality or providing some data, the research consists of comparing the device's capabilities according to its product page and specification sheets as well as experience gathered during the execution of the main project to what is required for the live analysis.

The questions themselves were derived from customer interviews carried out previously as well as analysis of some usage examples given by the customer.

G.3.3 Results

The following subsections answer the questions designed in Appendix G.3.1 in detail, giving exemplary solutions or recommendations where applicable.

Data Availability

This subsection is concerned with defining the availability of the data types defined in Table G.1.

Current Time While the device itself, given that it is based on a very basic microcontroller, does not have functionality to get the current time, there are three ways to acquire it at runtime in the user code running on the device.

Firstly, if the GPS antenna is attached to the device, it is possible to extract the time sent along with the GPS signal (see [alronzo](#), No date) and save it for later use.

Secondly, given that a connection to cellular networks can be established via the device's cellular module, the current time can be obtained from a network time protocol (NTP) server as it was implemented in the main project (see Appendix F.3.2).

Finally, an external device could be connected to the microcontroller via either WLAN, Bluetooth or its serial port and supply the time either once upon startup or repeatedly.

The main difficulty regarding time keeping on the device is the fact that there is no such thing as an internal clock that could keep track of time independently from the user code running on the device. For this reason, keeping the time up to date based on the device's processor cycle count needs to be taken into account when implementing the software.

This means that the current time (data ID 1) is available in the right device configurations, given that some effort is put into the implementation.

OBD Data Data IDs 2, 3, 4 and 9 (see Table G.1) require the availability of vehicle OBD data on the interface attached to the device. Given that the device is powered over the OBD interface (see [Freematics](#), No date(a)), the presence of the interface and therefore the availability of data can be assumed.

Selected Gear Given that the OBD interface's purpose is to expose mainly engine and emission data, other vehicle-related data such as the currently selected gear is not directly available over the OBD interface (see Wikipedia, No date). However, because the OBD interface design includes the high and low cables of the vehicle controller area network (CAN) bus, it is theoretically possible to extract more data, although it would require additional effort (see Kanda Admin, 2012). Whether or not the information that can be extracted from the CAN bus includes the currently selected gear depends on the manufacturer's implementation.

In summary, this means that there is no single solution to obtain the currently selected gear from the vehicle that would work in all cases.

Location As explained in Appendix G.1.2, the Freematics ONE+ device can be equipped with a GPS receiver. This means that, if the antenna is present, the device can be programmed to read the current location in latitude and longitude coordinates from this sensor.

However the location is limited to pure coordinates, any meta information on these coordinates, such as what country they belong to, has to be read from an external source.

Road Type As described in Appendix G.3.3, the device is capable of obtaining the current location's coordinates, but not the relevant meta information for this location. This means that the device has to rely on external sources, such as Google's Geocoding API (see Google LLC, 2018) or similar services, that are able to match geographical coordinates with the appropriate meta information.

This in turn means that theoretically it is possible to obtain the required information, however it is connected to considerable effort and most likely financial resources which may outweigh the possible benefits of the proposed system.

External Conditions Because a test drive's parameters may include external conditions (temperature, humidity, barometric pressure), the device should be able to read them from a source. Although sensors that supply this type of data exist (see Sparkfun Electronics, No date), the Freematics ONE+ device does not offer a sufficient number of I/O pins to attach them. For this reason, obtaining atmospheric data directly is not possible.

Similar to how time can be obtained though, it is possible to program external devices to collect the data and relay it to the telemetry device via WLAN, Bluetooth or the serial port.

Summary In summary, the question "Is it possible to gather all required data from either a sensor, an online API or another source?" can be answered with "Yes", although obtaining some specific data, such as atmospheric conditions and the currently selected gear as well as the type of road the car is driving on, may require more effort than can be justified for the project.

Data Handling

The proposed system's tasks include the handling of data after it has been read. More specifically, parameters may take four distinct forms:

- **Maximum Value:** A maximum value for a value that may not be overstepped (e.g. "The car must never exceed a speed of 100 km/h")

- **Minimum Value:** A minimum value for a value that may not be understepped (e.g. "The car's engine must never run at lower than 700 RPM")
- **Average Value:** A value's average must stay at (or close to, as defined by the parameter) this target value (e.g. "The car should use 7.1 litres of fuel per 100km on average")
- **Range:** A value must stay within a range defined by upper and lower limit for the value (e.g. "The test drive must incorporate between 10 and 50 kilometers of driving on highways")
- **Sum:** The cumulated sum of a value must be at (or close to, as defined by the parameter) this value (e.g. "The test drive must consist of 100km in length")

Given that the device is based around the ESP32, a fully programmable microcontroller, the implementation of these parameters is possible without effort. Depending on the number and type of calculations however, the calculations may put a strain on the processing cores and the system RAM. Without experiments however, it is for all practical purposes impossible to determine how strongly performance will be impacted.

This means that the question "Is the device capable of handling the acquired data in ways suitable to handle all tasks?" can be answered with "Yes".

User Interaction

Given that the system's main purpose is to communicate the imminent or fulfilled breach of a parameter to the user, it is important that the device can interact with the user adequately. Furthermore, before the system is actually started, a user should be able to add/ remove/ change parameters for the system to watch. This way, the device can be used for a new test drive without requiring a firmware flash.

Unfortunately the device does not offer much in terms of human interface devices per se: Only a single LED can be controlled by the user code and there is no way to for a user to directly interact with the device such as buttons or dials. As mentioned in Appendix G.3.3, there is also no way to add these because of the devices insufficient number of I/O ports. Although two exist (see Freematics, No date(b)), they are blocked by the GPS antenna.

For this reason, all user communication must take place via an external device. This could for example be connected via either the device's serial port, Bluetooth or WLAN. This means that the question "Is there a way in which a user can interact with the device?" can be answered with "Yes", although it requires the development of an external communication device or software (e.g. a smartphone app) to realise.

Performance

While for all practical purposes it is impossible to calculate this accurately and proof it beyond doubt, the experience gathered during the implementation of the main project, the gathering and transmission of telemetry data, implies that the device's performance is sufficient for these tasks. Given the device's reasonable hardware specifications (see Appendix G.1.2), no performance issues should arise.

Although the customer interviews and example analyses carried out as part of this feasibility study did not include specific performance measures such as the rate at which data should be

gathered, the project team believes that a "reasonable" reading and analysis performance is a non-issue.

This means that the question "Can the device carry out the required number of data reading / tracking actions?" can be answered with "Yes".

Integration

The final question to be answered during this feasibility study was "Can the system be integrated into the data transmission project as a subsystem?".

Due to the fact that one of the project's core design aims was to make the final product easily extendable, the implementation of analysing the already available data and reading some more data from different sources is most definitely feasible without unreasonable effort. In combination with the already resource-heavy main project however, the device's hardware specifications may prove to be insufficient.

Although, as mentioned in Appendix G.3.3, the analysis of data alone is unlikely to be an issue in terms of computing power, the main project already is designed to make use of all available resources. With the additional tasks to fulfill, it is possible that the device's computing power is insufficient.

In summary, this means that, although the addition of the analysis to the main project is no problem in terms of software implementation, the hardware may not support it and the question "Can the system be integrated into the data transmission project as a subsystem?" must be answered with "No".

G.4 Conclusion

This chapter summarises the results of the research execution described in Appendix G.3 and combines them in order to answer the question whether or not the proposed project is feasible. Table G.2 gives an overview over the answers and remarks, if any.

TABLE G.2: Data required to carry out live analysis

| Number | Answer | Remarks |
|--------|--------|--|
| 1 | Yes | Some data may take considerable effort to acquire |
| 2 | Yes | - |
| 3 | Yes | The user cannot directly communicate with the device, instead a bridge in form of e.g. a smart-phone app is required |
| 4 | Yes | - |
| 5 | No | While the software implementation of adding the analysis to the main project is no issue, the device's hardware specs are most likely insufficient |

This shows that all but one of the core questions could be answered affirmatively. The only answer that had to be answered with "No" was whether the proposed analysis system could be integrated into the existing telemetry transmission project's solution.

In conclusion this means that, although parts of the proposed system may take considerable effort to implement, it can be implemented as a stand-alone solution. The addition of this proposed system to the main project's software solution however is most likely infeasible due to performance limitations.

H Example JSON File

```
1  [
2    {
3      "time":1472110107000,
4      "configurationName":"MyConfigName_v42.ncf",
5      "data":
6        {
7          "some_attribute":2,
8          "other_attribute":42
9        }
10   },
11   {
12     "time":1472110209000,
13     "configurationName":"MyConfigName_v42.ncf",
14     "data":
15       {
16         "some_attribute":5,
17         "other_attribute":105
18       }
19   }
20  ]
```

I Product Class Diagram

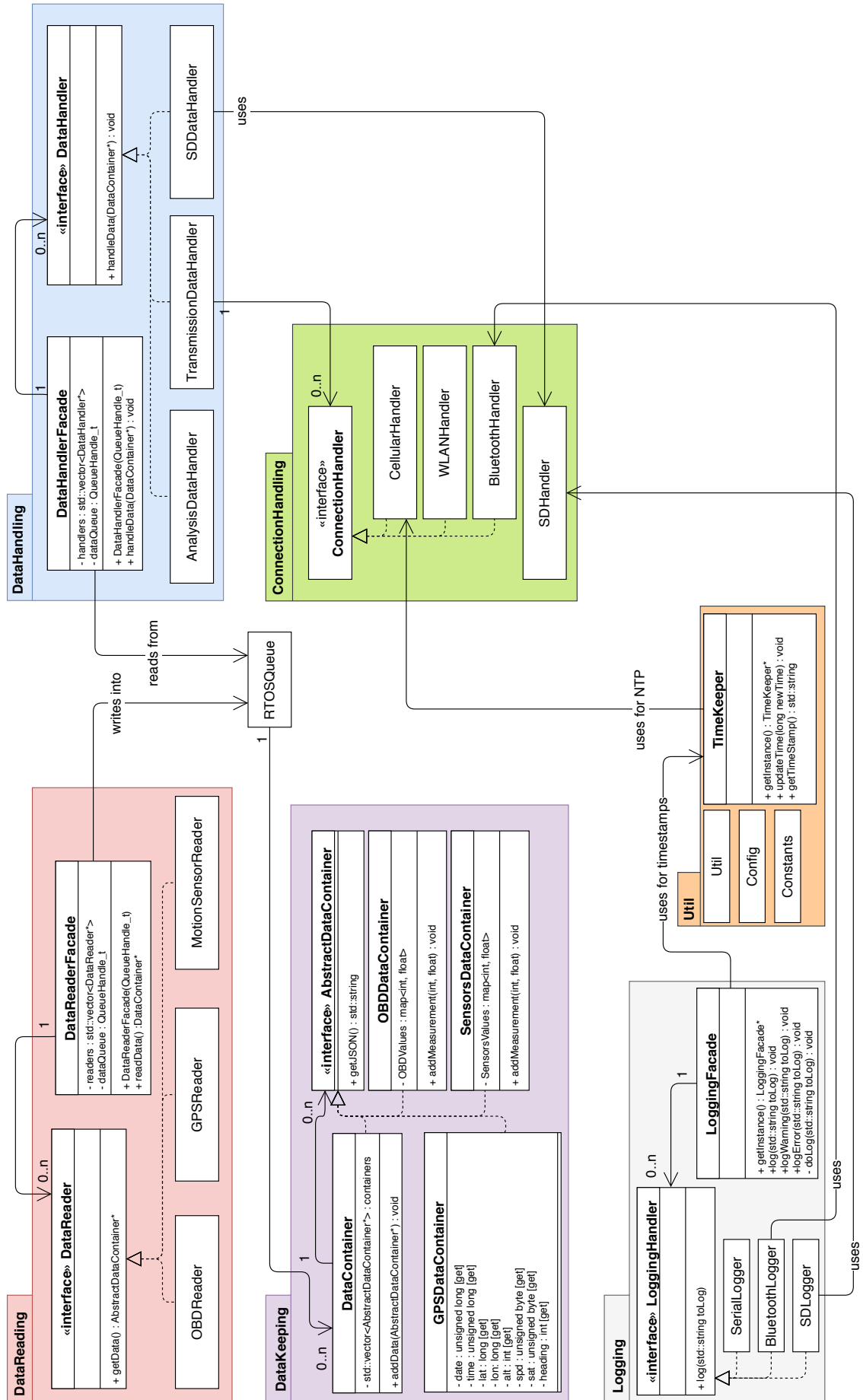


FIGURE I.1: Product Sequence Diagram: Handling Data