NTNU – Trondheim
Norwegian University of
Science and Technology

Revolve Analyzer

Development of racing data analysis software

Lauritz Møllersen
Per Øyvind Stadheim

Master of Science in Computer Science
Submission date: June 2012
Supervisor: Alf Inge Wang, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

Car racing has become a more and more technological sport. Not only are the cars themselves pioneering within technology, but also the aspects outside of the actual racing. One of these aspects are racing data analysis. In our thesis we will look at racing data analysis and the software used for it, we will identify missing functionality and try to find ways to make our own analysis software based on this experience.

Our project presents a software called Revolve Analyzer. The main parts of our thesis is:

- Research on how to build an efficient, modifiable and user-friendly racing data analysis software.
- A detailed software architecture. The design and implementation of the architecture is covered in detail.
- Seven different plugins that fit this architecture, all solving different problems within racing data analysis.
- An evaluation of the software from the end-users.

Preface

This report have been submitted to the Norwegian University of Science and Technology as fulfillment of the requirements for the degree Master of Science in Technology.

The report is divided in the following parts:

Part I gives a general understanding of the project. It contains a quick briefing, including project background, motivation, development process and schedule.

Part II contains the findings from the prestudy. It is used as a reference in the research and development work.

Part III contains the requirement specification document, architectural description of the system and an overview of the results from the implementation phase.

Part IV contains an evaluation of the efforts conducted during the project and the conclusion.

Contents

Preface iii

I INTRODUCTION 1

1 Project background 3
1.1 Project context . . . 3
1.2 Assignment choice and motivation . . . 3
1.2.1 Assignment description . . . 4
1.2.2 Changes . . . 4
1.3 Revolve . . . 5
1.3.1 Racing data analysis . . . 5
1.4 Project goals . . . 6
1.5 Stakeholders . . . 7
1.5.1 Project group . . . 7
1.5.2 Revolve . . . 7
1.5.3 Course staff . . . 7

2 Development 9
2.1 Development processes . . . 9
2.1.1 Modified waterfall model . . . 10
2.1.2 Scrum . . . 10
2.2 Development tools . . . 11
2.2.1 Documentation . . . 11
2.2.2 Implementation . . . 12
2.2.3 Communication . . . 12
2.3 Schedule . . . 12

3 Research 15
3.1 Introduction . . . 15
3.2 Questions . . . 16
3.3 Methods . . . 16
3.3.1 The empirical method . . . 16
3.3.2 The engineering method . . . 17
3.3.3 Literature search . . . 17

II PRESTUDY 19

4 Prestudy introduction 21
4.1 Motivation and goal . . . 21
4.2 Sources . . . 22
4.3 Racing terminology . . . 22

5 Race car data acquisition 25
5.1 What is it? . . . 25
5.2 How? . . . 26

6 Race car data analysis 27
6.1 Data channels . . . 27
6.1.1 Speed . . . 27
6.1.2 Engine RPM . . . 28
6.1.3 Lateral g-force . . . 28
6.1.4 Steering . . . 28
6.1.5 Throttle . . . 29
6.1.6 Math . . . 29
6.2 Displaying data . . . 29
6.2.1 Strip chart . . . 29
6.2.2 XY graphs . . . 30
6.2.3 Track map . . . 30
6.2.4 Time slip . . . 32
6.2.5 Sector and split times . . . 33
6.2.6 Channel reports . . . 34
6.2.7 Histograms . . . 34
6.2.8 Video . . . 35
6.3 Combining different types of visualization . . . 36
6.3.1 Engine health . . . 37
6.3.2 Driver inputs . . . 37
6.3.3 G-forces . . . 37
6.3.4 Brakes . . . 37
6.4 Further analysis examples . . . 38

7 Existing software 41
7.1 Products examined . . . 41
7.1.1 i2 Pro . . . 41
7.1.2 Race Studio 2.0 . . . 42
7.1.3 Race Technology's Data Analysis Software . . . 43
7.2 Summary . . . 43

III OWN CONTRIBUTION 45

8 Requirements specification 47
8.1 Requirements analysis process . . . 47
8.2 Requirements overview . . . 47
8.3 Specific requirements . . . 48
8.3.1 Functional requirements . . . 48

8.4 Changes . . . 52
8.5 Scenario and use cases . . . 52
8.5.1 Scenario . . . 52
8.5.2 Basic use cases . . . 53

9 Design phase introduction **57**
9.1 Process . . . 57
9.2 Architectural viewpoints . . . 58
9.3 Design issues and quality focus . . . 58
9.3.1 System lifetime and time-to market . . . 59
9.3.2 Usability . . . 59
9.3.3 Modifiability . . . 59
9.3.4 Modularity . . . 59
9.3.5 Performance and scalability . . . 60
9.3.6 Portability and platform independence . . . 60

10 System design **61**
10.1 System environment . . . 61
10.2 User interface and conceptual model . . . 62
10.2.1 User interface . . . 62
10.2.2 Conceptual system model . . . 64

11 High level architecture **65**
11.1 Modules overview . . . 65
11.2 Modules rationale . . . 66
11.2.1 Specific plugins . . . 67

12 Low level architecture **69**
12.1 Plugin architecture . . . 69
12.2 Modules . . . 71
12.2.1 Project and support modules . . . 71
12.2.2 Main GUI . . . 75
12.2.3 Plugin module . . . 75
12.3 View module . . . 77
12.4 Sequence diagrams . . . 78
12.5 Choice of programming language . . . 83

13 Architecture summary **85**

14 Implementation **87**
14.1 Introduction . . . 87
14.1.1 Approach . . . 87
14.1.2 Testing . . . 87
14.1.3 Open Source . . . 88
14.1.4 Test data . . . 89
14.2 Sprint 1 . . . 89
14.2.1 Goals . . . 89
14.2.2 The project module . . . 90
14.2.3 Parser and channel data structure . . . 91
14.2.4 Calibrator and math channels . . . 92

14.2.5 Segmentation . . . 93
14.2.6 Error module . . . 94
14.2.7 Serializer . . . 95
14.2.8 Plugin framework . . . 95
14.2.9 View interface . . . 96
14.2.10 Sprint wrap-up . . . 97
14.3 Sprint 2 . . . 98
14.3.1 Goals . . . 98
14.3.2 MainGUI . . . 98
14.3.3 The BasicView component . . . 99
14.3.4 Sprint wrap-up . . . 100
14.4 Sprint 3 . . . 101
14.4.1 Goals . . . 101
14.4.2 Linechart, XY-chart and TimeSlip plugins . . . 101
14.4.3 Map plugin . . . 104
14.4.4 Report Plugin . . . 105
14.4.5 Video Plugin . . . 105
14.4.6 Notepad Plugin . . . 107
14.4.7 Playback . . . 107
14.4.8 Sprint wrap-up . . . 108
14.5 Completeness . . . 110
14.6 Deployment . . . 113

15 User guide **115**
15.1 Dependencies . . . 115
15.2 Create project and load data . . . 115
15.3 Create a new view . . . 116
15.4 Calibrating channels . . . 116
15.5 Analyzing data using graphs . . . 117
15.6 Generating a track map from the data . . . 118
15.7 Using video to analyze data . . . 118

16 Plugin development guide **121**
16.1 The interface . . . 121
16.1.1 Required methods . . . 121
16.1.2 Other methods . . . 122
16.2 Deployment . . . 123

IV EVALUATION AND CONCLUSIONS **125**

17 User evaluation **127**
17.1 First impressions usertest . . . 127
17.1.1 Participants . . . 127
17.1.2 Feedback . . . 128
17.1.3 Conclusion . . . 129
17.2 SUS user test . . . 129
17.2.1 Alternatives to SUS . . . 129
17.2.2 SUS results . . . 129

17.2.3 Feedback . . . 130
17.2.4 Conclusion . . . 131

18 Self evaluation **133**
18.1 Planning . . . 133
18.2 Research . . . 133
18.3 Development . . . 134
18.4 Report . . . 134
18.5 Individual thoughts . . . 134

19 Conclusion **135**

20 Further work **137**
20.1 GUI . . . 137
20.2 Plugins . . . 137
20.2.1 Linechart and XYChart . . . 137
20.2.2 Map . . . 137
20.2.3 Report . . . 138
20.2.4 Video . . . 138
20.2.5 Notepad . . . 138
20.2.6 Playback . . . 138
20.3 Project platform . . . 139
20.3.1 Math channels . . . 139
20.3.2 Input data restrictions . . . 139
20.4 General . . . 139

References **140**

Appendix **I**

A Data channels **I**

Part I

INTRODUCTION

Chapter 1

Project background

In this chapter we will go through the context of this project, and what our goals and motivation are. The purpose of writing this chapter is not only to clarify to the reader the 'whats' and 'whys' regarding this project. It is also a way for ourselves to acknowledge the different aspects of the project, and make some important decisions early on. We start off by explaining the reasons for doing the project, the choice of assignment and the motivations that lie behind this choice. A brief introduction to the student group Revolve, our customer, and their motivation for requesting this project then follows. Lastly, we define our goals and stakeholders for this specific assignment and set up a rough schedule for the project lifetime.

1.1 Project context

This is the Master's Thesis for 5th year students at NTNU's Computer and Information Science department. The project is the entire workload of the tenth semester. Our assignment is to develop a data analysis software designed for visualizing and analyzing the data gathered by a race car. The project builds upon the so-called In-depth Project, TDT4506, which was half the workload of the ninth semester. In this project we developed the basic framework for such a software and did a prestudy on the problem domain.

1.2 Assignment choice and motivation

The assignment chosen is titled "Developing software for presentation and analysis of data from a race car". The assignment is given by the student organization Revolve, which is seen as an external customer. By choosing this project we wish

to train our ability to examine different areas that need software products to be designed. It is also an interesting task to design a relatively large piece of "real" software, as both of us have only participated in smaller school projects before. The fact that this is also within the world of cars and racing is another motivating factor, since this is an area both project members are very interested in. There will be more on racing, data logging and data analysis in Part II Prestudy.

1.2.1 Assignment description

This is the assignment description delivered to NTNU by Revolve, translated to English.

[External] Developing software for presentation and analysis of data from a race car
Revolve NTNU (http://www.revolve.no/) is a student organization created in order to participate in Formula Student. Formula Student is a student competition with a focus on development of technology and competence. The participants shall develop and build a race car which fulfills formula SAE rules and which, during the summer of 2012, shall compete at the Silverstone race track in England. One of the parts of this project is to develop software to analyze logged data and viewing this data in real-time.

Main components of this assignment will be:

- User-friendly software for analysis of logged data
- Real-time presentation of data from the car

Further parts of the assignment could be:

- Filtering data to construct the driven track
- Real-time estimation of driving conditions, optimized racing line and lap times
- Simulation of the driving
- Estimation of physical parameters based on logged data

This assignment will be executed in cooperation with Revolve NTNU, especially the group that is responsible for the data logging systems.

1.2.2 Changes

The assignment description is identical to the one Revolve provided for the in-depth study project. In other words, it has not been updated for the master project. However, through the in-depth project, some of the priorities have changed. The most important of the changes is the fact that real-time presentation of data from

the car is no longer considered important. Revolve wants the project to be centered around analyzing already logged data to begin with, since they currently have no possibility of transferring live data to a computer. Further changes to the software requirements are elaborated in the requirement analysis chapter, see section 8.

1.3 Revolve

Revolve NTNU is a student organization [1] which is going to participate in the Formula Student competition. Formula Student is one of the biggest so-called "educational motorsport competitions" [2]. There are teams from many universities from all around the world competing against each other. The goal is to build a prototype race car that will be evaluated by judges. The winner is not necessarily the team that can drive their car the fastest around a track, but rather the focus will be on many factors such as the build design and quality of the car, innovation, technical solutions and of course lap times in different challenges. The competition is held annually in the United Kingdom during summer. Revolve was founded in spring 2010 by four students who wanted to participate in more exciting projects. The goal from the start was always to participate in Formula Student. Revolve has since matured into a big project that incorporates many different educational aspects. Naturally, engineering plays a big part in the project, but there are also work to be done in marketing, economics, working with sponsors and project management [1].

Revolve currently have their own office at Valgrinda, provided by IPK, Institute for Production and Quality Technique. They are about 40 members, and most of them work on the car as volunteers, not through school projects [1].

1.3.1 Racing data analysis

Data logging and analysis are techniques used by almost all professional racing teams. Many teams even have a separate position which is filled by a data engineer [3]. Good data logging and analyzing can shave off precious tenths of a second on a racer's lap times [3, 4]. Revolve have members who are working on the data logging and acquisition hardware. Instead of buying the analysis software, they wanted to make it themselves. Since there are no system developers in their organization, they put this project up as a in-depth project for fifth-year computer science students, with a possibility to continue the work as a Master Thesis.

There are several reasons for them to launch this project. It's cheaper than buying software for analysis (free, actually), and most of the software designed to analyze racing data has to be used along with data acquisition hardware from the same company. Having their own "consultants" developing the program from scratch means they will have a lot more choices when it comes to customization to their

Figure 1.1: A picture of the Revolve group from their go-kart driving tests [1].

needs. Support is easier when the customer has a direct cellphone number to the developers, as well as the fact that this software will support plugins which can be developed by other students. Also, since the Revolve car will be made by students, it's nice that the software used is made by students as well.

1.4 Project goals

The main goal of the assignment proposed is to produce a working racing data analysis software, or RDAS, with the capabilities for analyzing data. This includes support for video, graphs, reports, math functions, playback (or simulation) and more. In the in-depth project we examined the RDAS problem domain, developed a requirements specification and a very basic example prototype of how a RDAS may be developed. As stated in the chapter concerning further work in the in-depth project report, the architecture proposed in the in-depth project has a few deficiencies and needs more work. Furthermore, the implemented prototype was created just to demonstrate the architecture, and has very little of required functionality. The focus in the master project will therefore be on the following goals:

- Continue the work from the in-depth project, more specifically; expand and improve the planned software architecture.
- Implement a functioning system that fulfills requirement specification.

- Work on the deployment of the software and find a way to measure its usability.

We also have side goals that include learning more about racing, data acquisition, data analysis and software development. We would also like to achieve a grade of B or better.

1.5 Stakeholders

The stakeholders in this project are the project group, the Revolve group and the course staff. In this section a description of each of the stakeholders and their primary concerns can be found.

1.5.1 Project group

Our primary concerns are the quality of the work; the architecture of the system, the programming and the documentation. The architecture and general quality of the software is very important since we want the software to be easy to continue working with for other developers.

1.5.2 Revolve

Their primary concerns are that the program must be completed before the race car is. This way they can test run the car and set it up properly before the Formula Student competition. They aim to have the car working in early April 2012. There must also be a focus on usability, performance, further development and maintainability of the program.

1.5.3 Course staff

The course staff consists of Alf Inge Wang. He will also be concerned with the quality of the work, but with a higher focus on the documentation, as well the academical and educational value of the project.

Chapter 2

Development

In this chapter, we'll mention the choices of different development methods and tools that we used in the project, and the reasoning behind choosing them.

2.1 Development processes

The development process, or model, describes the overall structure of a software development project. There are a many different development processes to choose from. Some are very rigid, some are more agile. Some are good for big projects with many participants, some are better for smaller projects. We believe that choosing the correct development process will have major impact on the result of our project, and therefore spent some time considering the alternatives. For reasons described in the following subsections, we chose to mainly use the modified waterfall model for our project, but for the implementation phase we chose to divide the tasks into groups of tasks, much like sprints in the SCRUM development process.

2.1.1 Modified waterfall model

The original waterfall model is a very rigid, and once the team complete a phase they don't go back and change anything [5]. The modified model has more lee-way in terms of changing the requirements when the project is in the design phase, for instance. We have used the modified waterfall model as a model for the overall scheduling of the project, see section 2.3.

We believe that dividing the work into different phases and setting a fixed time limit on these phases will help the progress of the project, and most importantly will help us keep track of the progress and remaining time. Every week we can consult the schedule to see how we are doing on progress. If we go over the allotted end date for a phase, we know we have to try and make up that time somewhere else.

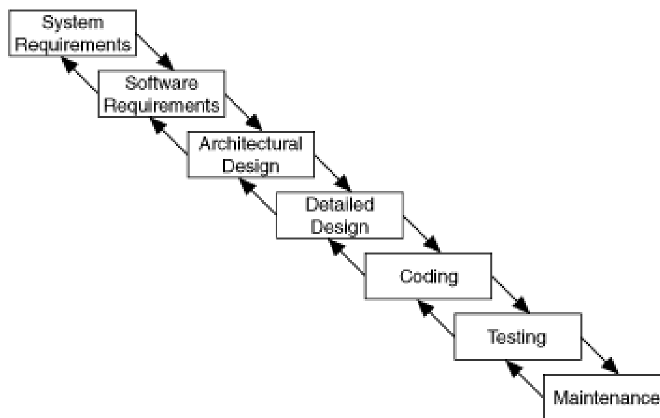

Figure 2.1: A modified waterfall model [6].

2.1.2 Scrum

The Waterfall Model is rigid, and as a contrast to that we have Scrum. Scrum is an agile process. It's based around small teams and there's less focus on formal documentation. With smaller groups and more specified individual roles, Scrum hopes to decrease the effect of social loafing [7]. Social loafing is the decreased individual performance that happens in teams. The larger the teams, the bigger the decrease. According to Fried [8] this can be as much as half the individual performance for groups of eight. Scrum focuses on having frequent meetings and doing the work in "sprints", usually about 30 days duration.

We chose to be inspired by this method for the implementation work because agile methods are known to help increase modifiability in a system [10]. Modifiability

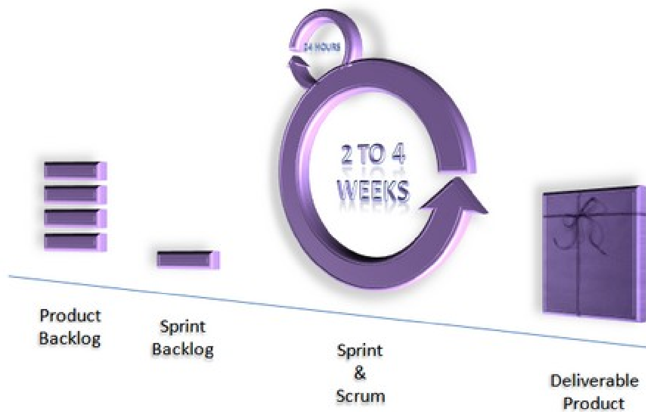

Figure 2.2: Quick overview of the Scrum process [9].

and modularity are very important in this project. It is also helpful for us to split the implementation into smaller sprints so we can get feedback faster, both from each other and from the end-users, and can react quicker to changes in the requirements.[7].

2.2 Development tools

In this section we list the different tools we used in the project.

2.2.1 Documentation

LaTeX - LaTeX is a document preparation program. This is the software used for the final edit of the report.

JabRef - JabRef is a tool for editing the bibliography file used in a LaTeX document.

Google Docs - We used Google Docs [11] for writing the report. It's a very easy-to-use tool that doesn't require any software installed. This also opens up for the possibility of working collocated.

Adobe Illustrator - We used Adobe Illustrator to create and prepare most of the illustrations and figures for the report.

Microsoft Visio - Microsoft Visio is the alternative we chose for drawing UML diagrams.

2.2.2 Implementation

NetBeans - NetBeans is the main IDE[1] that was used by both group members. This is the IDE we have the most experience with, and it felt like the right choice, especially for writing Java and for easy GUI-creation using mostly Swing.

Subversion - We used Subversion as our versioning software[2], mostly because NTNU "Orakeltjenesten" can give students access to a group-area where they can host heir own Subversion-server there, and the fact that both project members have experience with it.

2.2.3 Communication

ProjectPlace - ProjectPlace is a web-based project management tool. This is the tool already in use by Revolve for planning activities and meetings.

Facebook - Facebook is a social networking website. We often used the Facebook Chat for informal talks about the project. For other communcation needs outside meetings, phone and email was used.

2.3 Schedule

To ensure progress in the project, we have set up a preliminary schedule for the different phases involved in the project.

ID	Task Name	Start	Finish	Duration
1	Startup	13.01.2012	26.01.2012	2w
2	Review of In-Depth Project	20.01.2012	02.02.2012	2w
3	Architecture	03.02.2012	23.02.2012	3w
4	Implementation and testing	17.02.2012	26.04.2012	10w
5	Deployment	27.04.2012	03.05.2012	1w
6	Evaluation	04.05.2012	17.05.2012	2w
7	Documentation and finalization	17.05.2012	08.06.2012	3,4w

jan 2012 · feb 2012 · mar 2012 · apr 2012 · mai 2012

15.1 · 22.1 · 29.1 · 5.2 · 12.2 · 19.2 · 26.2 · 4.3 · 11.3 · 18.3 · 25.3 · 1.4 · 8.4 · 15.4 · 22.4 · 29.4 · 6.5 · 13.5 · 20.5 · 27.5 · 3.6

Figure 2.3: Our preliminary schedule.

[1] IDE stands for Integrated Development Environment. In other words, this is the software that is used for development.

[2] Version, or revision, control software is used to manage changes to files. This is often used when a team of developers are working on one project, so that multiple developers can work simultaneously on the source code.

Startup - The startup will consist of initial planning, setup of workspace resources and meetings with the project's stakeholders.

Review of in-depth project - In this phase we will be looking at the results we have from the in-depth project and try to take some lessons with us to the master project.

Architecture - In the next phase the research around the architecture will be done, and we will complete the system's architecture.

Implementation and testing - All the development, as well as the testing, will be done in this phase.

Evaluation - The research around usability and how to evaluate it will be done here. We will gather evaluations from the end-users, as well as personally evaluate how the project went.

Documentation and finalization - The report will be finalized during this phase. All project documentation will be finalized, such as the JavaDoc, and user manuals.

Tasks 1 and 2 typically rely more on the project members being collocated, while Architecture and Implementation opens up for more distributed work. We have planned some overlapping between some of the activities. The Startup and the Review of In-Depth Project will naturally overlap slightly. Architecture and Implementation and testing also, since there will be some testing of Architectural elements.

Chapter 3

Research

In this chapter we will give a brief introduction to our research, the questions we've chosen to focus on and the methodology we've used to find and deduce information on the subject.

3.1 Introduction

In order to be aware of the choices we make and to see how well we reach our goals, research is going to play an important part. According to Dawson [12], the first thing one needs to do is ask oneself "the five Ws":

"What is your research?"
Our research is centered around racing data analysis, and more specifically around the software used in this field. We will be doing more research on what it takes to make a good RDAS. More specifically, we will be looking at how we can make a RDAS which have high modifiability and usability. We will also find a way to measure the usability of the resulting software.

"Why do you want to do the research? In other words what is its purpose?"
The main reason for doing the research is to make as good an RDAS as possible. We also feel that research in this field will complement the rest of our master thesis perfectly. Another helpful motivation is that we are both very interested in this field, and not only the field of racing but also the field of software development.

"Who will be our participants, or research subjects?"
The main research subject is the software itself, and the racing data analysis problem domain. When it comes to testing the usability of the program, we will be using members of the Revolve group, in other words the end-users.

"Where will the research be conducted?"
Much of our research will be literature reviews with some e-mail and phone communication. This research will mainly be conducted from our workspaces, either at home or at the school. We will also be conducting user tests of the program. These will be distributed to the users via e-mails, along with a runnable test-program.

"When are you going to do your research?"
The research will be conducted from the starting date of our master thesis, January 13th 2012, until the end date, June 8th 2012.

3.2 Questions

Due to this project being a software engineering project, less focus is put on pure research. Still, many types of research have been required throughout the project lifetime, and here we will try to sum up the essence of what we are researching in two questions.

RQ1: How can we measure usability?
RQ2: How can we design a system with high modifiability?

Other research work conducted in the project is connected to the implementation phase, where lack of API knowledge and shortcomings in programming experience lead to searching for solutions. There was also a lot of literature search conducted during the prestudy in the in-depth project.

3.3 Methods

There are several different research methods that can be used in software engineering. In this section we will look closer at the methods used in our research.

3.3.1 The empirical method

When using the empirical method, data is gathered in order to confirm a hypothesis [13]. In our case, we will run SUS-tests [14] on the program with the end-users. This is mainly how we will be answering RQ1.

For more information about the SUS-tests and results, see section 17.2.

3.3.2 The engineering method

"A scientist builds in order to learn; an engineer learns in order to build." - Fred Brooks.

This method entails building and testing a system according to a hypothesis [13]. This is what we are doing with our RDAS and our need to make it flexible and modifiable. The software and its architecture has undergone a lot of changes during the process in order to make it better. This is mainly how we will be answering RQ2 and implementation challenges.

3.3.3 Literature search

A literature search, or review, is to analyze literature already written on the subject [13]. We will be using the data collected using literature search in order to improve our own project.

Part II

PRESTUDY

Chapter 4

Prestudy introduction

In the in-depth project, we performed a thorough study on the problem domain of race car data analysis. In this part we have included a shortened and updated version of the results of this study.

4.1 Motivation and goal

In order to get the best possible results for our development project it is important that we understand the the domain which we are conducting the development in. Also, to be able to design a high quality software product, it is vital to achieve a good understanding of what the customer is expecting. To accomplish that, in addition to a thorough requirements analysis and good communication with the customer, a study of the problem domain is necessary. Having elaborate knowledge of the problem domain helps us, the developers, make independent and correct decisions regarding the design of the software. The prestudy documentation will also function as a reference in the design phase. The goals of this documentation is to gain thorough knowledge of:

- Basic racing terminology knowledge.
- What data logging is, and how it is conducted.
- How data is analyzed and how data analysis can help improve racing performance.
- Existing software for this purpose.

This will result in an understanding of the world the system is suppose to interact with, and thus help us understand how the system will be used.

4.2 Sources

In the prestudy process, most of the literature read was online articles and The Data Logging Manual by Graham Templeman [3]. Throughout the project lifetime we also participated in a few, very informative, meetings. They were mainly about racing strategy, the car being built, and the Formula Student competition [2]. Although this is not directly relevant to this software project, these meetings gave a deeper understanding of the problem domain and how the software may be used. Also, Andreas Ernestus, an amateur race driver, gave a lecture where he demonstrated how he uses analysis software.

4.3 Racing terminology

Much of the terminology used in racing literature are not common words and terms. Like any other business, racing has its own language. In order to understand the problem domain when doing the prestudy, we need basic knowledge of this language. To achieve this, we had to look up many different terms, and we collected a list of encountered and frequently used terms for further reference. Most of these definitions are from formula1.com's glossary [15].

Apex The middle point of the inside line around a corner at which drivers aim their cars.

Brake balance The balance between the front and rear braking power. In race cars, this is often represented as a switch in the cockpit to alter the split of the car's braking power.

Chassis The main part of a race car, to which the engine and suspension are attached, is called the chassis.

Cockpit The section of the chassis in which the driver sits.

Counter-steering This is when the driver turns the steering wheel out of the turn, in other words in the opposite direction of the turn. This is often done in order to stop the car from sliding out of control, i.e. to counter oversteering.

G-force A physical force equivalent to one unit of gravity that is multiplied during rapid changes of direction or velocity. Racing drivers experience severe g-forces as they corner, accelerate and brake. Most commonly measure in gs. One g is approximately $9.8m/s^2$.

Grip The amount of traction a car has at any given point, affecting how easy it is for the driver to keep control through corners.

Lap The amount of laps is the measure of how many times the driver has driven around the track.

Line Also referred to as racing line. This is the route the vehicle takes around a turn. The fastest line is usually not the shortest line, but the line that allows the car to travel at the highest possible speed through the turn.

Long g Longitudinal g is the amount of g-forces on the car in the forward and backward direction, in other words from acceleration and deacceleration.

Oversteer When a car's rear end doesn't want to go around a corner and tries to overtake the front end as the driver turns in towards the apex. In layman's terms, when the rear wheels slide out in a turn. This often requires countersteering and throttle control to correct.

Pits An area of the track separated from the start/finish straight by a wall, where the cars are brought for new tyres and fuel during the race, or for set-up changes in practice, each stopping at their respective pit garages.

Power band This is the range of RPM where the engine is able to operate effectively. It is usually defined by being from the RPM where the engine produces peak torque to the RPM where it produces peak horsepower.

Rolling lap A lap that is started when the car has speed, i.e. rolls over the finish line, as opposed to a standing lap where the car is stationary when it starts.

Rev limit Many cars have a limited maximum engine RPM, the rev limit. This is to prevent engine damage and the driver going outside the power band.

RPM The amount of engine revolutions per minute. Often referred to as revs.

Telemetry A system that beams data related to the engine and chassis to computers in the pit garage, or a data storage unite on the car, so that engineers can monitor that car's behaviour.

Throttle Often referred to as the "gas" or "accelerator", the throttle is used to regulate how much power the driver wants from the engine.

Torque Literally, the turning or twisting force of an engine, torque is generally used as a measure of an engine's flexibility. An engine may be very powerful, but if it has little torque then that power may only be available over a limited rev range, making it of limited use to the driver. An engine with more torque - even if it has less power - may actually prove quicker on many tracks, as the power is available over a far wider rev range and hence more accessible. Good torque is particularly vital on circuits with a number of mid- to slow-speed turns, where acceleration out of the corners is essential to a good lap time.

Traction The degree to which a car is able to transfer its power onto the track surface for forward progress.

Understeer Where the front end of the car doesn't want to turn into a corner and slides wide as the driver tries to turn in towards the apex. In layman's turn, when the car does not want to turn and the front wheels start sliding.

Chapter 5

Race car data acquisition

This chapter deals with what data acquisition is, and how it works. It is not essential for this project, but it will help us understand what is going on with the rest of the system the software is suppose to interact with, and why.

5.1 What is it?

Data acquisition is a prerequisite for analyzing and improving car and driver performance. In the simplest terms, data acquisition is a method of gathering information for use to answer specific questions [16]. Early data acquisition was done manually with a stopwatch, a pen, and a piece of paper. But the world has changed, and we are being offered a better way to gather the data we need to answer the questions we have about our race cars. In fact, we can measure things now that we never dreamed possible. Now that the ability exists to analyze each and every performance parameter of the car and the driver, accurate use of this data can provide a key advantage on the racetrack. While computers and electronic dataloggers have been around for at least 30 years, they have been very expensive in their early days. In the last 15 years or so they have become much more affordable. By now, all major professional racing teams have sensors on their cars which collect data from the car itself. Most of them also have a data engineer on their team with responsibility for the computer equipment [3]. Many amateur racers also use data acquisition and analysis to improve their racing performance.

5.2 How?

Sensors are installed on the car which log the condition of different aspects of the car through-out a race, for instance the car's speed or the voltage of the car's battery. A sample is one measurement of this data, and there is often collected 100 samples each seconds (100Hz) for every sensor. The data is then saved to the data logger's internal memory or USB memory stick, after which it can be analyzed on a computer using data analysis software. The possibilities of what

Figure 5.1: A picture of a Race Technology DL1 Data Logger [17].

to collect is endless, but the most common sensors are for speed, RPM (amount of engine revolutions per minute), the lateral g-forces on the car, steering wheel position, throttle pedal position and brake pedal position or brake pressure [3, 16]. There is even the possibility of expanding beyond gathering data from the car by taking a look at the driver's heart rate and breathing [16].

Chapter 6

Race car data analysis

Here we will explain what race car data analysis is, and how the data collected from a car's sensors is used. We will use this study to try to answer the question "what are the typical features of racing analysis software?" In order for the raw data gathered by the loggers mentioned in chapter 5 to become useful information it has to be processed, then analyzed. This is where data analysis and proper software comes into play.

6.1 Data channels

Each sensor's data output is gathered in separate data channels. As mentioned before, the possibilities of what to measure are many, but some channels are more interesting and important than others. In this section we will go through the standard data channels that is a must-have in any race car data acquisition setup. They are:

- Speed
- Engine RPM
- Lateral G
- Steering
- Throttle

6.1.1 Speed

The most important data channel is obviously the speed channel. Speed is what we are interested in when it comes to racing. The faster we go, the sooner we will

reach the finish line, and therefore speed is the ultimate goal of our efforts. The speed trace has a characteristic shape; it climbs and dips. The dips represent the corners, and the climbs represent the straight areas of the track. Since speed is the ultimate end product, the speed strip chart, see section 6.2.1, is a good place to start any analysis. By putting speed traces from different laps on top of each other, it is easier to identify problem areas [3, 4].

6.1.2 Engine RPM

There are many reasons to be interested in engine RPM. This trace also has a characteristic shape; it drops and rises under gear changes and falls steeply under braking. Users interested in engine health might want to know if the engine has been over-revved. When it comes to driving performance, it may be interesting to check how fast the gearing is done, if the driver is having low RPM in corners and if the RPM reaches the preferred heights at the longest straights. These are only examples of how the engine RPM channel can be used. Most of the time, it is convenient to plot the RPM channel together with the speed channel, since these are closely related data [3].

6.1.3 Lateral g-force

Lateral g is the measure of cornering force on the car. If the car is turning sharply at a high speed, lateral g will be high. A sharp turn can also cause lateral g to be negative. We assume that negative g-force means left-hand turn, positive g-force means right-hand turn. This is very intuitive when viewing the data in the graphs. Lateral g is mostly used to see how the car, and not to mention the driver, is doing in the turns. It will give insight into at what line the driver is taking the turn. "The classical, constant radius racing line will be reflected by lateral g-trace that is symmetrical, building up and falling away in a steady manner" [3]. The more modern, and faster, line will take a later apex and get on the power earlier. This will show as a more asymmetric trace, climbing steeply and decaying slowly [3, 4].

6.1.4 Steering

The steering channel records the steering input from the driver. This channel can be thought of as how the driver wants the car to turn, and the lateral g as how the car actually turns. Steering, especially combined with lateral g, is very helpful to determine over- and understeering problems. It can also be used to see how the driver is performing on the track [3, 4].

6.1.5 Throttle

The throttle channel is simply the values of how much throttle is being applied. The trace should be very easy to recognize, since it will be on maximum pressure for about 60-80% of the time [3]. The throttle channel gives insight into how the driver is behaving. Aggressive drivers will have very swift increases and decreases in throttle pressure, while more level-headed drivers will have smoother changes [3, 4].

6.1.6 Math

Math channels is a tool to make new use of existing channels. It can be used to combine channels in various ways, and also to calibrate them. Typical use would be to show the derivative of speed, which gives the acceleration, or for instance to show the current gear ratio by combining the speed, wheel circumference and engine RPM [3].

6.2 Displaying data

In order to help the user in the analysis process, there are numerous different ways of visualizing and displaying the data acquired. In this section some of these are examined to give an understanding of how and why these features are important to include in analysis software [3, 4].

6.2.1 Strip chart

The traditional strip chart is by far the most used visualization technique. It shows a standard two-dimensional chart with the selected data channel on the y-axis and either time or distance on the x-axis. Using distance is the conventional way of doing it, as it allows one lap to be superimposed on another for comparison. If time is used as the scale for the x-axis, one lap will lag behind the other and the traces will get out of sync. Figure 6.1 shows this effect. Generally, there is little reason to use time as the base except for special investigations. A useful feature in strip charts is the ability to shade the background corresponding to the different segments of the lap. This helps the user understand where the different events are actually taking place. In the example figure, the corner segments are shaded with a green color [3].

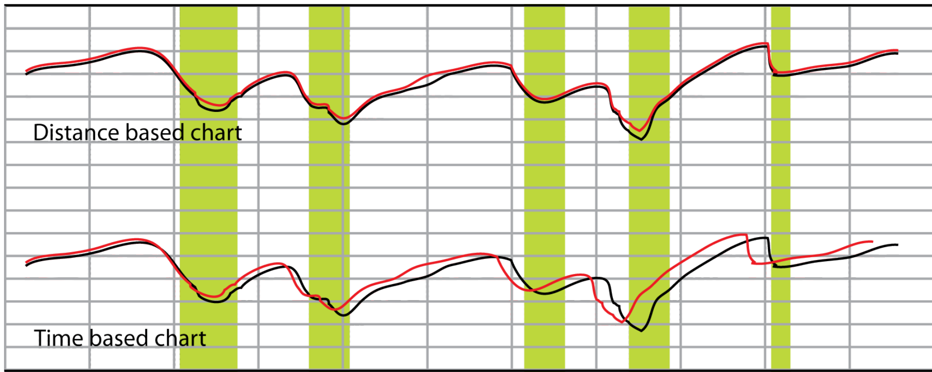

Figure 6.1: An example of a strip chart.

6.2.2 XY graphs

XY graphs are a way of comparing one variable with another. These graphs can sometimes present information in a surprising way. Instead of having time or distance on the x-axis like in the standard strip charts, it is possible to choose whatever seems interesting. Each occurance of a data pair is plotted as a dot in the chart. Some useful combinations of data for XY graphs are [3]:

- lateral g vs. long g (friction circle)
- speed vs. RPM (gear chart)
- lateral g vs. speed (aero effects)
- steering vs. lateral g (handling)
- front brake pressure vs. rear brake pressure (brake balance)
- lateral/long g vs. oil pressure (oil surge)

It is, for instance, possible to evaluate driving technique by creating a friction circle [18], and from that it is possible to interpret the driving style. Another example is oil surge. The oil surge chart can help give us an idea whether or not the car is suffering from surge during cornering or braking. If the oil pressure readings are clustered together and do not drift downwards under high g-forces then there is no problem. Figure 6.2 shows that pressure can fall during hard cornering in either direction.

6.2.3 Track map

The track map is an excellent way of putting data into context. There is a form of reassurance in it, that confirms our assumptions. If there is one thing that the

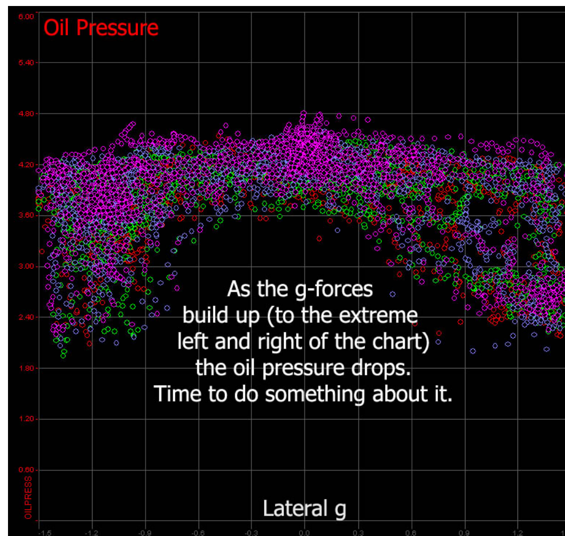

Figure 6.2: An example of a strip chart of a car's oil pressure [18].

user of the software, and specially the driver can relate well to, it is the map. This makes it a necessity in user-friendly analysis software. When working with other data representations, a cursor on the map linked with the current activity may show the user where the car is at that point in geographical terms. However, as the experience level increases, less time will be spent looking at the map. The user will be less dependent on it, because he starts to recognize where he is in the data just by looking at other features. Nevertheless, it is still nice to have access to a map for learning through this form of visualization and note taking purposes.

Normally, there is no pre-made model of the race track. However, the track map can easily be generated from GPS data. These maps generally have high quality and represent the track well, by little or no effort, since no calculations are required. The task of drawing the map is as simple as transforming the list of GPS coordinates into a matrix of points ready for display. An alternative method for generating a track map is by combining the speed and lateral g data channels. This method often requires a lot of effort in order to generate a decent representation of the track. The quality of the maps using this method vary to some extent. Some systems give almost nonsensical output initially, that needs to be tweaked to overcome the errors. Another factor that may complicate the map generation process, is badly set up and calibrated sensors measuring speed and lateral g inaccurately.

One of the most interesting features a track map can have, is the ability to shade and color-code different areas of the map to reflect different types of data. This way, the user can get all sorts of extra insights just by looking at the map. Examples of useful rules that can be applied to shade and color the map can be by throttle

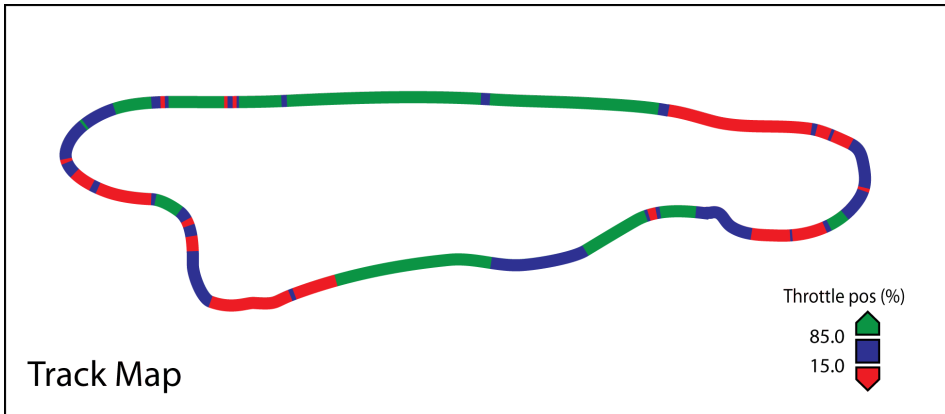

Figure 6.3: An example of a map with color coding.

position, gear and gear changes, braking or speed. An example of throttle position coloring rules are shown in figure 6.3, where the green areas represent 85% or more throttle, red areas represent 15% or less throttle, and blue areas represent the mid-range throttle positions [3].

6.2.4 Time slip

The "time slip" line is a clever way to show the user where time is gained and lost on the track. It is a distance-based strip chart that uses a reference lap as a baseline. This reference lap is usually the fastest lap, but it could also be interesting to use other laps as baselines. The chart then has another line that represents the time difference in the lap being compared to the reference lap. Positive values means the driver is losing time, and negative values means he is gaining time. In other words, if the time slip line is heading upwards it is bad news and time is being lost. When the line heads down, the compared lap is gaining time on the reference lap, and this is good news. It should also be possible to compare several laps at the same time.

Furthermore, scaling the graph properly is an important issue, since looking at differences as small as hundredths of a second is generally a waste of time. This requires the user of the analysis software to look for which scale the time slip chart is shown in, but this could also somehow be managed automatically by the software [3, 4]. Figure 6.4 shows an example of how a time slip line works in practice. The top trace shows speed, and the lower trace is the time slip line. This example shows two corners linked by a short straight.

The blue trace is better because, although the blue driver brakes about 40 meters earlier for corner 1, he is able to carry more speed through the corner. This obviously gives more speed on the straight connecting the corners. Through corner

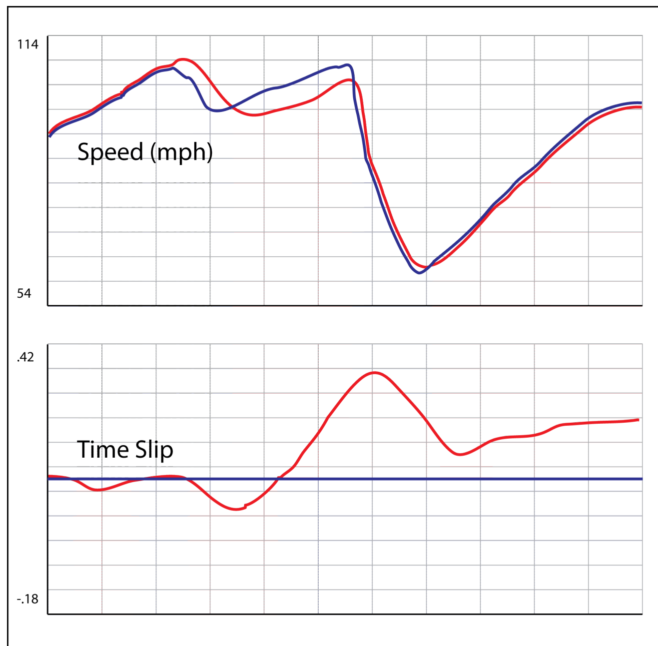

Figure 6.4: An example graph of time slip.

2, early braking and a slightly slower passage lets the driver get back on to the throttle a little earlier. This is not compensated for by the red driver who is having a higher cornering speed through corner 2. Late braking on the red lap meant that the driver felt like being on the brakes for longer, and shed more speed than necessary [3].

6.2.5 Sector and split times

Another approach to the split times is by using a detailed printout of all sorts of lap and sector times. Split times are interesting not only to see what the lap times were, but also to be able to create a "fantasy" lap made up of the fastest segments of each lap. This fastest possible lap, however, is not very realistic, as having greater speed in some area will affect the beginning of another area. Therefore, the fastest rolling lap is also looked at. The fastest rolling lap takes the fastest sequence of sectors irrespective of where they start. For example, if lap 4 had a slow start, but a good last few sectors, and the start of lap 5 was good, then the start of lap 5 and the end of lap 4 can be combined and more realistically represent what the car and driver are actually capable of.

What is important to look for is that the times are consistent to some degree. If they are not, then something is probably wrong with either the car or the driver.

Noticing any issues here helps direct the attention to other data visualization techniques to locate what might be the problem (e.g. by comparing speed charts for the various laps) [3].

Absolute split times			1		2		3		4	5		
2007 Oct 18 BH Indy S1	run 2 lap 2	7.689	6.312	4.715	6.470	3.387	3.790	5.763	3.627	11.548	2.956	00.56.258
2007 Oct 18 BH Indy S1	run 2 lap 3	7.591	6.266	4.855	7.831	3.852	4.181	5.796	3.957	11.518	2.967	00.58.813
2007 Oct 18 BH Indy S1	run 2 lap 4	7.514	6.343	4.427	5.959	3.270	3.757	5.540	3.439	10.887	2.967	00.54.103
2007 Oct 18 BH Indy S1	run 2 lap 5	7.471	5.688	4.292	5.844	3.282	**3.573**	5.431	3.201	10.761	2.957	00.52.501
2007 Oct 18 BH Indy S1	run 2 lap 6	7.406	5.538	4.203	5.838	3.297	3.647	5.455	3.173	10.376	2.881	00.51.814
2007 Oct 18 BH Indy S1	run 2 lap 7	7.307	6.076	4.370	**5.752**	3.262	3.990	5.785	3.234	10.424	2.899	00.53.100
2007 Oct 18 BH Indy S1	run 2 lap 8	7.427	5.756	4.249	5.987	3.316	3.668	5.445	3.255	10.322	2.939	00.52.364
2007 Oct 18 BH Indy S1 (be	**run 2 lap 9**	7.321	**5.277**	4.162	5.836	3.298	3.593	5.492	**3.161**	**10.186**	2.922	**00.51.247**
2007 Oct 18 BH Indy S1	run 2 lap 10	**7.301**	5.460	**4.146**	5.809	3.301	3.990	5.737	3.221	10.207	2.894	00.52.065
2007 Oct 18 BH Indy S1	run 3 lap 2	7.425	5.775	4.377	6.268	3.416	4.078	5.566	3.285	10.426	2.905	00.53.522
2007 Oct 18 BH Indy S1	run 3 lap 3	7.417	6.132	4.418	5.839	3.229	3.767	5.371	3.208	10.421	**2.841**	00.52.642
2007 Oct 18 BH Indy S1	run 3 lap 4	7.362	5.842	4.193	5.835	**3.215**	3.757	**5.361**	3.826	10.664	3.099	00.53.154
minimum value		7.301	5.277	4.146	5.752	3.215	3.573	5.361	3.161	10.186	2.841	
maximum value		7.689	6.343	4.855	7.831	3.852	4.181	5.796	3.,957	11.548	3.099	
average value		7.436	5.872	4.367	6.106	3.344	3.816	5.562	3.382	10.645	2.936	
std deviation		0.117	0.353	0.22	0.583	0.17	0.198	0.165	0.273	0.465	0.064	
Theoretical best lap												
2007 Oct 18 BH Indy S1	**best**	**7.301**	**5.277**	**4.146**	**5.752**	**3.215**	**3.573**	**5.361**	**3.161**	**10.186**	**2.841**	**00.50.813**

Figure 6.5: An example graph of a sector report. The sector times highlighted with yellow are the fastest.

6.2.6 Channel reports

When analysing race car data, various channel statistics might be interesting to look at. They provide information not just about how things are behaving on that day. They can also be added to a set of records or book-keeping that all racing competitors should have. Engine health is an obvious choice here, and we would e.g. be interested in maximum, minimum and average values for pressures, temperatures and RPM. These data tell us something about whether the engine is operating within the desired limits or not. Also, providing statistics about the various g-forces and top speeds give us information about the handling. The same goes for almost any area of interest, like driver inputs such as braking.

Generally, the more sophisticated the channel reports are the better. In addition to showing minimum, maximum and average levels for a channel, splitting it over the segments of a lap and showing statistical data such as standard deviation might be useful functions. [3]

6.2.7 Histograms

Histograms, or bar charts, are charts that show the percentage of a particular variable occurring within specified intervals. They can generally be applied to any data channel, but are specially useful for engine RPM, speed and throttle.

Lap	avg Speed_1	avg Throttle	max Front Brake	max Rear Brake	min Corner Radius	max Brakes used for
run 1 - lap 3 03.12.343	37.1	19.2	19.6	36.5	**-155.00**	12.50
run 1 - lap 4 01.31.826	77.6	52.4	12.5	30.7	-155.00	11.10
run 1 - lap 5 02.01.441	58.2	21.3	11.5	24.0	-155.00	8.00
run 1 - lap 6 02.03.517	57.4	24.4	10.7	20.1	-155.00	**13.40**
run 1 - lap 7 01.20.039	87.6	58.1	16.0	31.7	-155.00	10.90
run 1 - lap 8 01.19.278	88.4	58.8	14.4	33.3	-155.00	8.90
run 1 - lap 9 01.19.486	88.0	61.7	17.4	40.3	-155.00	9.80
run 1 - lap 10 01.20.154	87.3	63.1	**20.5**	**41.3**	-155.00	10.50
run 1 - lap 11 01.19.351	88.2	60.2	12.7	32.6	-155.00	9.00
run 1 - lap 12 01.18.756	88.9	66.6	15.1	33.5	-155.00	9.30
run 1 - lap 13 01.18.276	89.5	64.5	13.2	32.4	-155.00	8.20
run 1 - lap 14 01.17.400	**90.5**	**66.9**	**15.1**	**34.0**	**-155.00**	**8.30**
run 1 - lap 15 01.18.537	89.1	64.7	18.4	35.7	-155.00	9.10

Figure 6.6: Shows the channel report for the braking pressure.

Engine RPM can be useful because it shows the amount of time the engine spends in each parts of its range, and a histogram is a quick way to ensure that the engine is not forced to operate out of the power band or above the comfortable limit. Analysing this gives an insight into whether the gearing is being done correctly, so this is one of the things to take into account when making gearing decisions.

Organizing speed into a histogram can tell us something about how the track is looking, by displaying the time spent in each speed range. This can help give directions on how to set up the car for the race in the best possible way. If a lot of time is spent at low speeds it tells us that this is a track with many sharp turns, which means agility and traction might be smart to prioritize. Throttle histograms are useful to see how much of the time is spent with the throttle wide open or firmly shut. Time spent between the two is a good indicator of how difficult the car and track are, and efforts spent improving the car will result in a smaller area in the middle of the histogram. A typical observation done in a throttle histogram is that after driving several laps and driving sessions on the same track, the time spent in the mid-range reduces significantly to the benefit of the times spent at full throttle. This indicates that the driver's confidence in the car has increased. The histogram example in figure 6.7 shows an example of the improvement made over a days testing [3].

6.2.8 Video

Using video as a part of the analysis system is, at the time of writing, a technology that is still maturing. Not a lot of software has this functionality, so a video is often played in a third-party media player while analysing the data.

Typical usage is having a window with video taken from a camera placed on the

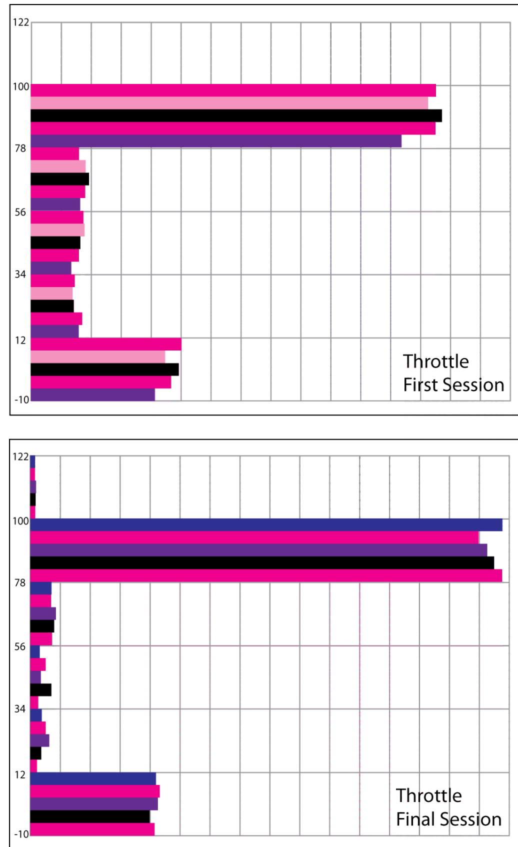

Figure 6.7: The throttle channel shown as a histogram.

driver's helmet or on the car. This video is synced with the data gathered from the sensors on the car. It can give a better overview of many situations, especially the line taken through a turn. This will help figuring out where a problem lies; with the car, or with the driver [3].

6.3 Combining different types of visualization

Showing only one data channel using one type of data visualization is rarely enough to give us the entire picture. Therefore, many different types of presentation techniques are often carefully combined to ensure an efficient analysis process. In this section we will look at how different channels and visualization techniques can be used to compose "pages" of related information. The possibilities of combinations are endless. We have looked at some of the more common setups [3].

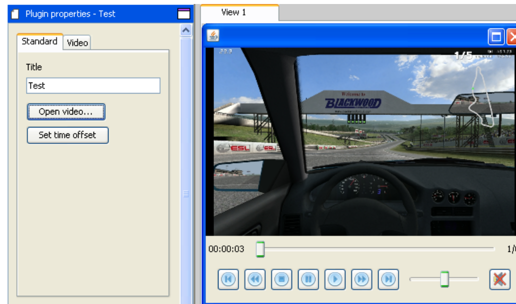

Figure 6.8: A screenshot from the video functionality in our software.

6.3.1 Engine health

Engine health is simply a matter of seeing that engine RPM, cooling liquid temperatures, oil temperatures, oil pressure and fuel pressure stay within reasonable limits. What is considered healthy engine RPM depends on each engine, but cooling liquid temperatures should be well below boiling and stay relatively constant throughout the race once they reach operating temperatures [3]. Strip charts showing these values should therefore be combined on one screen.

6.3.2 Driver inputs

To analyze driver inputs, combining all channels that represent driver input and looking at them together is a good start. This will give the user an idea of how the driver is doing in the race. Typical channels to combine are steering, braking pressure, throttle pressure, gear changes and clutch pedal (if there is one!) [3].

6.3.3 G-forces

Showing all the data from g-forces together, typically lateral g, long g and lateral + long g in a math channel, helps spot whether acceptable absolute values are being reached [3]. This depends on the kind of car being analyzed. For an amateur race car, a maximum sustained lateral force of 1g might be okay, but for a Formula One car, anything below 3g would be considered low.

6.3.4 Brakes

A good way to see if the brakes are functioning as expected is to look at speed, brake pressure and longitudinal g together. It is then possible to see how the

braking is affecting the car, and maximum deacceleration through looking at the g-forces.

6.4 Further analysis examples

The main idea behind both data acquisition and analysis, is to go faster. This is done through using the information acquired to change the way the driver performs and the way the car is set up. Generally, the more sensors, the easier it is to figure out what changes could increase performance.

For example, if the car is understeering, this is visible through getting less g-force than the actual amount of steering being done. This problem can be identified through overlaying the graph for lateral g-forces with the graph for steering wheel position. After this is discovered, there is a need to figure out if there is a problem with the car, or with the driver. Is the driver steering into the turn smoothly, or jaggedly? How is the throttle position throughout the turn? There are many factors to consider.

One of the most typical things to customize when racing, are the gear ratios. A high gear ratio will give a better acceleration, but lower top speed. A low gear ratio will give a slower acceleration, but higher top speed. All the gears on a car have different gear ratios. First gear has the highest gear ratio, and the top gear has the lowest [3]. The idea is to find a gear ratio that is low enough to give "enough" top speed. What that speed is depends on the track, the car and the driver. A very simplistic approach to finding the correct gear ratio is to look at the RPM channel, preferably in combination with the gear and speed channel. If, on the highest speed sections of the track, the car is closing in on, but not hitting, the rev limiter or unhealthy high engine RPMs while in the top gear, the car has a good gear ratio. If a higher gear ratio is set, the car will most likely hit the rev limiter, which stops the car from accelerating further. If a lower gear ratio is set, a slower acceleration occurs without getting a higher top speed (since we won't be hitting that top speed because of the slower acceleration).

This is of course a simplified way of looking at it. There are several other factors that should be considered. For each turn the user should analyze what gear and RPM range is being used when accelerating out of the turn. If the car is not within the power band of the engine, a higher or lower gear ratio might help. Of course, sometimes it is as easy as using a different gear during the turn, or improve the line so the driver can take the turn at a higher speed.

To give another example, let's say the driver says there is a problem with the traction of the car. When trying to pinpoint a problem, it is either the driver or the car that's at fault. The engineers will often blame the driver, and the driver will often blame the car, i.e. the engineers. If the user looks at the data and make a math channel that displays GPS speed divided by RPM, he will see where the

traction problems occur. This channel should look fairly smooth, with big changes when a gear shift occurs. If it is jagged, that could indicate traction issues [3]. This can again be looked at with steering and throttle input from the driver, to see if he is being too aggressive, or if the car's setup is wrong [4].

These are just a few examples of problems that can pinpointed, and hopefully solved, using racing analysis software. For more examples, we refer to the field guide in the Data Logging Manual [3].

Chapter 7

Existing software

Many software products for analyzing logged data from racing vehicles already exist. In order to see what functionality should be included in our own software, we had a look at the other products out there. Examining these products also gives us ideas and tips on what might be common problems in such software.

7.1 Products examined

The first product we tested was i2 Pro from MoTeC [19]. This is a powerful software with many good functions, but the GUI had a tendency to get a bit cluttered with too much information at once.

Next up was Race Studio 2.0 by Aim Sports Studios [20]. It isn't as powerful as i2, and also has pretty limited choices when it comes to customizing the GUI.

Lastly we tested Race Technology's Data Analysis Software [21]. We found that this software had a very good, customizable GUI. In addition it was plenty powerful for most amateur racers.

7.1.1 i2 Pro

The first thing the user notices is that i2 has very powerful zooming tools, and the graphs are a very integral part of this software. The segments are solved in a nice way, and it's easy to switch between laps and look at specific segments. There is also a simulation mode, which shows the steering wheel, throttle position, brake pedal position, g-forces and an "over-/understeer" meter. This is probably based on comparing the g-forces to the steering input, and it's a good idea for some further work on our software.

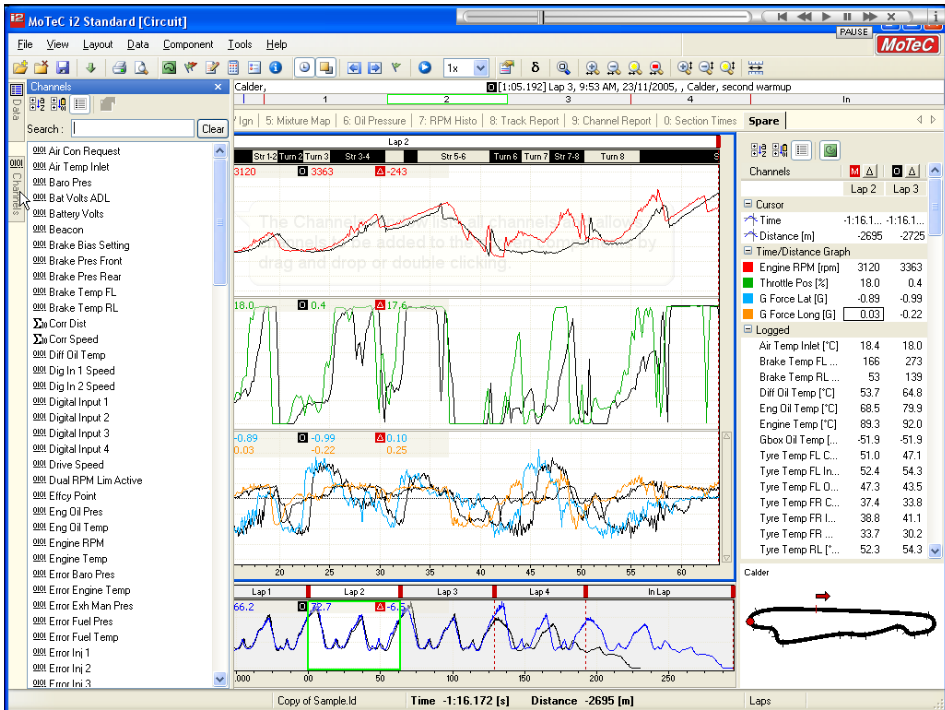

Figure 7.1: A screenshot of the i2 software..

Other than that i2 has a lot of the same functions as our program, but it does not have support for video playback and synchronization. It also does not work with any other data loggers than MoTeC ones. i2 does not seem to have any support for saving a "view", or worksheet as it's called in i2.

7.1.2 Race Studio 2.0

Whereas i2 looks very professional and well-polished, Race Studio looks like it has been put together in a hurry. It's hard to combine things on one screen, and there's limited functionality. It does create a very nice GPS map, and it does have a basic tool for suspension analysis, but the lap times and channel reports are way too simplistic.

This program does not have support for video either. However, Aim Sports do deliver a data logger which can record video. This logger combines the data from the car with the video into a video signal that has an overlay, for instance of the speed or the RPM. It's not possible to separate these signals and analyze them in the software.

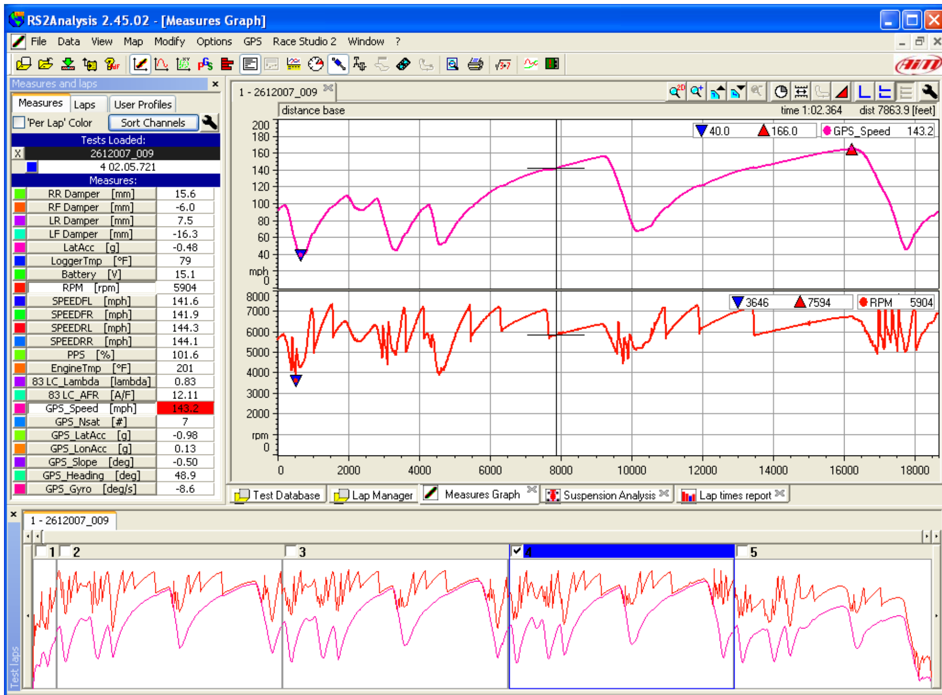

Figure 7.2: A screenshot of the Race Studio 2.0 software..

7.1.3 Race Technology's Data Analysis Software

First impressions are that this is a very user-friendly software. It's not that it has less functionality than the other programs, but the menu system is better thought out so it doesn't feel as cluttered. It also have some interesting features such as acceleration times (time from x mph to y mph), and a possibility of simulating changes to the car, for instance how increasing the power of the car will affect it. The simulation tool is also supposed to help find the areas where the driver can improve.

Again, it has most of the standard features of racing data analysis software. There is no support for video, and does not seem to be any support for saving a "view", a screen setup.

7.2 Summary

All in all, we found that there are many functions that are common in most data analysis software. All of them have support for standard visualization techniques

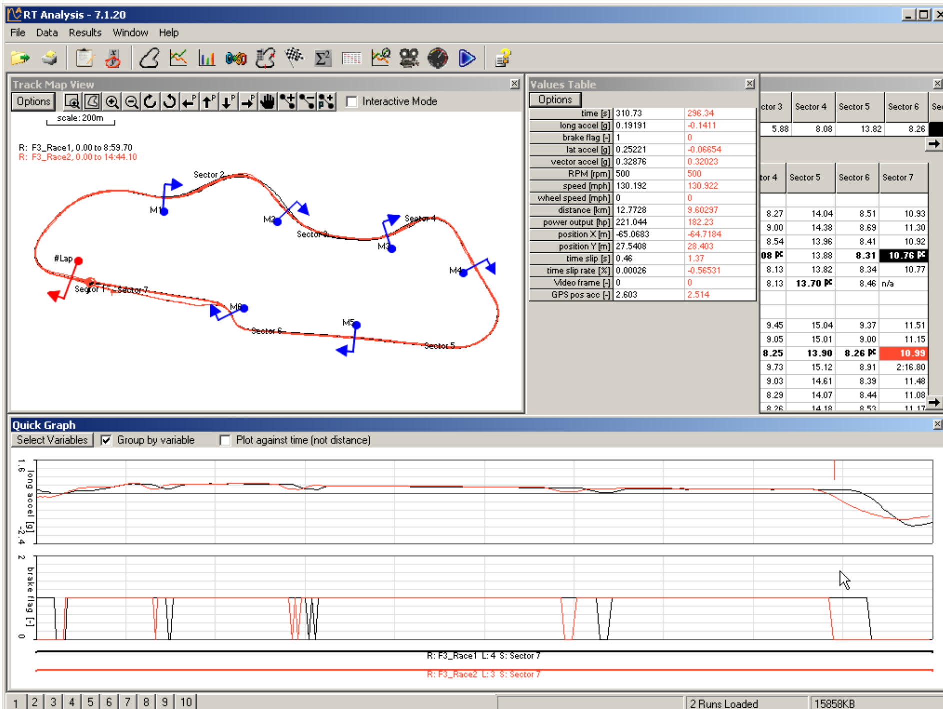

Figure 7.3: A screenshot of the RT Analysis software..

such as drawing multiple graphs, using XY graphs and histograms. Most also have support for playing through the data like a media file, sector times, channel reports and time slip. What separates the software is the degree of configurability; to what extent the user can customize his or hers user experience, and to some extent the amount of functionality offered.

Through this study, we have gotten some good pointers to what functionality is needed, and also on what is important in order to make the program easy to use while still being able to extensively setup the graphical interface.We have also gotten a feel for what functionality is rare, or completely missing from today's racing data analysis software.

Part III

OWN CONTRIBUTION

Chapter 8

Requirements specification

This section contains the requirements document for the software system to be implemented. We start off by describing the process of how we developed the requirements specification, then give a general overview of what is required of the system, before we get to the specific functional and non-functional requirements.

8.1 Requirements analysis process

In order to develop the requirements we arranged two meetings with representatives from Revolve. In the first meeting we discussed the desired functionality while taking notes. We then analyzed and looked through the requirements and evaluated how hard these requirements would be to implement. We then arranged a second meeting to verify the final set of requirements and add a priority to each requirement ranked by how important that particular functionality is for them. Revolve also answered questions we had about the requirements already specified, as well as added a few more requirements.

8.2 Requirements overview

The software to be implemented shall help the user analyze logged data from various sensors in a racing vehicle by organizing and visualizing the data. In general, it shall operate as follows:

The system shall be able to parse the logged data from a data file. Data from each sensor should be stored in separate data channels that each require individual calibration and, in some cases, smoothing, depending on which sensor is the origin. Data channels must be possible to divide into laps and lap segments (i.e. corners

and straights). It should also be possible to create "math channels" which shall be definable logic and derivative functions of the existing data channels. Furthermore, channel reports must be generated for each data channel. Channel reports should contain statistics like minimum, maximum and average value. The way the data channels shall be visualized is similar to the visualization techniques examined in the prestudy; through standard 2D graphs and charts, a visualization of the current geographical position in a generated track map, and possibility to view the video captured by a camera in the vehicle. Great emphasis is put on the flexibility and usability of the user interface. It shall be configurable and easy to use - it should be easy to set up any combination of data and presentation techniques to suit most analysis needs (e.g. comparing laps and channels). These specialized combinations created by the user should also be possible to save and re-use later.

A complete list of sensors, i.e. data channels, that are intended to be logged in the finished Revolve race car can be found in appendix A1.

8.3 Specific requirements

The functional and non-functional requirements were developed late during the in-depth project, and has received little changes since then. They have all been assigned a priority (C=Critical, H=High, M=Medium, L=Low) in collaboration with the customer, Revolve. The requirements assigned critical priority are an absolute necessity to make the system work. High and medium priority requirements represents mainly the desired functionality, while low priority requirements are meant to be less important bonus features.

8.3.1 Functional requirements

In this section we will list the functional requirements with an ID that is used throughout the report. We will also list the priority that have been set on this requirement by Revolve.

Table 8.1: The system requirements.

ID	Description	Priority
	ID - Input data	
ID1	The system shall be able to extract (parse) the logged data from a predefined data structure stored in a file	C
	DC - Data channels	
DC1	The system must support generation of channel reports that show the minimum, maximum and average values for the selected channel	H
DC2	Time and distance shall have their own data channels, like e.g. throttle and RPM	M
DC3	The system must support math channels with logic and derivative functions. It should be able to save custom math channel configurations for later re-use	M
DC4	The system must support calibration of channel data (e.g. translating from voltage to temperature) using a set of polynomials	H
DC5	The system must have the ability to smooth graphs	M
DC6	The system must support lap segmentation (corners and other types of segmentation) to make it easier to compare the same segment in different laps (manual)	H
DC7	It must be possible to get the lap times without the use of external sensors/beacons, for instance by using GPS	L
DC8	The system must support channel and segment color coding for ease of comparing segments	H
DC9	The system shall have be able automatically segment the race map into curves and straights	L

Continued on next page

Table 8.1 – continued from previous page

ID	Description	Priority
	GUI - Graphical user interface	
GUI1	The GUI shall provide an empty visualization layout interface where the user can place different visualizations (e.g. standard graphs or video)	H
GUI2	It should be possible to duplicate a visualization layout	L
GUI3	The GUI shall have multiple tabs containing the different visualization layouts	H
GUI4	Multiscreen support	L
GUI5	The visualization layout interface should have a snap to grid functionality for easy layout configuration	L
GUI6	The system shall have a map view to help the user put data into context	H
GUI7	The system shall be able to display video recorded from the vehicle and synchronize it with the other data	M+
GUI8	The system shall be able to put synchronized overlays on video, like speedometer and minimap	L
GUI9	Own GUI panel for visualization options	M
GUI10	Synchronized zoom between visualizations	M
	GV - Standard graph view	
GV1	It shall be possible to draw data as function of time	H
GV2	It shall be possible to draw data as function of distance	C
GV3	The system shall support time slip visualization with time gained or lost in different positions	M
GV4	The system must have the ability to view more than one data channel/lap/segment in the same view (color coded)	H
GV5	The standard graph visualizations shall support horizontal and vertical scaling and zoom	H
GV6	The standard graph visualizations must have the possibility to synchronize their cursor positions for improved understandability	
GV7	The system must support autofitting of segment/map/marked area to the current view	M

Continued on next page

Table 8.1 – continued from previous page

ID	Description	Priority
	XY - Standard XY-graph view	
XY1	The system must have the ability to draw XY-graphs, in which other data channels than time or distance are plotted against each other, e.g. speed versus RPM	M
XY2	The system must have the ability to view more than one lap/segment in the same view (color coded)	H
XY3	XY graphs shall support horizontal and vertical scaling and zoom	H
XY4	The XY graph visualizations must have the possibility to synchronize their cursor positions for improved understandability	H
	Other	
O1	The system shall support a undo/rollback history (with at least 100 actions)	M
O2	The system shall have customizable hotkeys	L
O3	The system should have the ability to store a work session to continue working on it at a later time	M
O4	The system shall support autosave (e.g. auto store work session every minute)	L
O5	The user should be able to store notes in the work sessions	M
O6	The system shall be able to export visualizations to a suitable format (e.g. PNG images or PDF documents)	L
	Non-functional requirements	
NF1	The system should be able to run on the hardware the laptop that Revolve will be using has, in other words Intel Core 2 Duo, 1GB RAM or more powerful	M
NF2	The program must run on the most common operating systems (Windows, Linux and MacOS)	L
NF3	Start-up of the system should take less than 5 seconds	M

Continued on next page

Table 8.1 – continued from previous page

ID	Description	Priority
NF4	The system should not stay in memory or as a background process once exited, and should not have memory leaks	H
NF5	The system has to come with a user guide	H
NF6	The code has to adhere to standard Java coding conventions and be documented by JavaDocs	H
NF7	The system shall be ready for testing early April 2012 when the car is scheduled to be finished	H

8.4 Changes

This section contains the changes done the requirements during the project lifetime.

The requirement ID2 has been removed. Revolve have not added technology capable of transferring real-time from the their data loggers to a recipient.

The follow two requirements have been added:

GUI10: Synchronized zoom between visualizations. M
O5: The user should be able to write and store notes in the work sessions. M

8.5 Scenario and use cases

To further describe the system we need some way to clarify and define the desired use of it. A suitable way to do this, is by creating a general scenario representing common usage of the software. We then divide the different operations into smaller packages, or use cases, for use in further design of the system.

8.5.1 Scenario

The racing team are at the racing track doing test runs. The driver just did two laps, and the team wants to analyze these two and compare them with each other. The datalogger has been active while driving the two laps, and the data is ready on a removable storage such as a USB flash memory. The data are given unsegmented in voltages, and needs calibration to represent understandable values. The team wants to analyze different aspects of the two laps and needs to use a combination of visualization techniques in a specific layout on the screen. The use of math functions is also needed for the complex analysis process. When the analysis is done, the users wants to store the results for later review and perhaps be able to reuse a new innovative combination of visualizations discovered in the process.

8.5.2 Basic use cases

From the scenario, we can derive use cases which represents the core functionality of the system. In the use cases we try to describe what the system should do for the user to achieve particular goals. Figure 8.1 shows a graphical representation of the different use cases identified. We also elaborate the most important use cases by using textual representations (UC1-4).

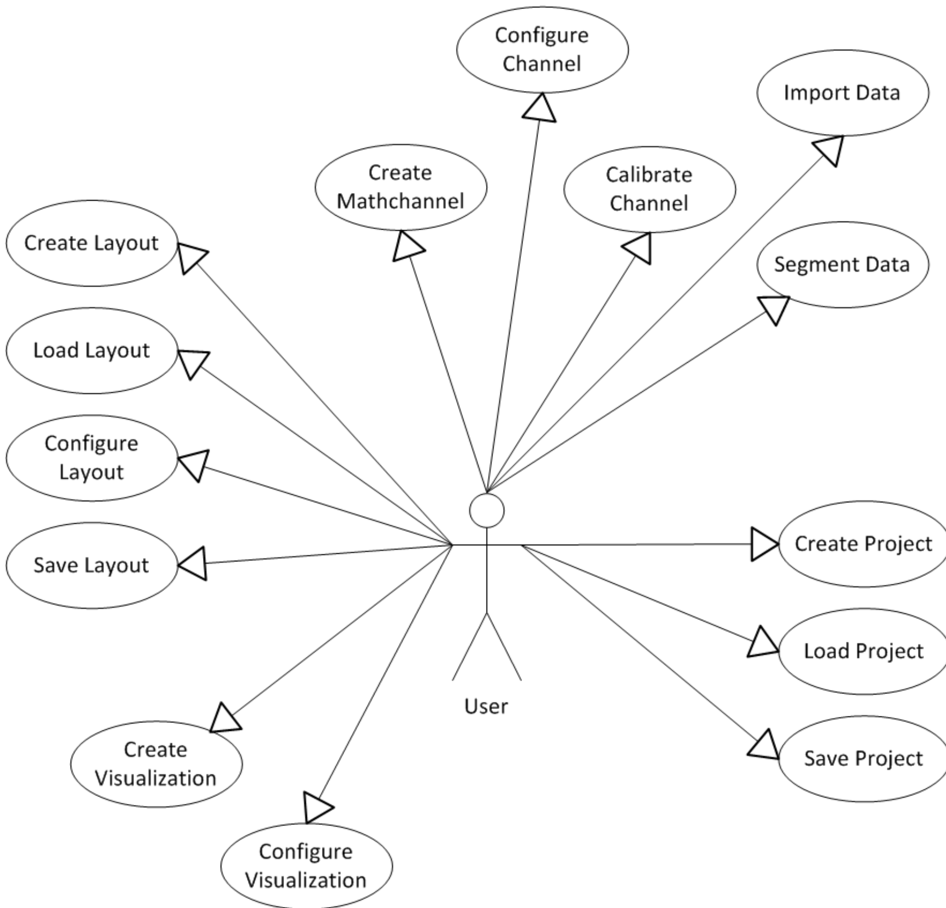

Figure 8.1: All use cases.

UC1: Create project and Import/Calibrate Data

1. To be able to start the analysis, the user needs start a blank session (analogue to creating a blank document in MS Word).
2. The system shall then present to the user a blank work session and an opportunity to import a set of sensor data.
3. The user then decides which data to import
4. The system shall then import the data and eventually notify the user when the process is complete.
5. To be able to view the data in meaningful context, the user needs select a way to calibrate the input data.
6. The system shall then present to the user an opportunity to set specific calibration polynomials for each channel.

UC2: Create layout

1. As the analysis processes are based on having different visualization components on the screen in some layout, the user needs to create a new layout to work on.
2. The system shall then add to the project a blank sheet to work on.

UC3: Create visualization and Configure Visualization

1. To be able to analyze the speed trace, the user needs to create a line chart visualization and configure it to show the desired graph.
2. The system shall then place a line chart in the current visualization layout which responds to customizations done by the user.

UC4: Save Layout and Load Layout

1. To be able to re-use a layout that may have taken a lot of time and effort to create, the user needs to store the layout with its configurations.
2. The system shall then store the layout to a persistent file system.
3. To re-use previously created layouts the user needs to load these into new projects.
4. The system shall then load the desired layout from the persistent file system and display it using the data channels in the current project.

UC5: Save Project and Load Project

1. To be able to continue working with a dataset and a set of visualization layouts at a later time, the user needs to store the entire session with its configurations.
2. The system shall then store the current work session to a persistent file system.
3. To re-use the previously stored work session, the user needs to load this from the file system.
4. The system shall then load the desired work session from the file system and display it in the graphical user interface.

Chapter 9

Design phase introduction

In this chapter we discuss important decisions made early in the design phase. First off we explain the chosen design process. Second we decide which architectural viewpoints to use as architectural documentation, and at last we go through the quality focus and design issues for the software design.

9.1 Process

It is crucial to have a systematic approach to the design process to achieve good results later in the project. Every aspect of system requirements needs to be considered, and the architecture must be documented in an unambiguous way. Many generic approaches to developing an architecture are suggested in textbooks and publications, but these are commonly modified to accommodate specific development team and project needs. Here we describe how we chose to approach the problem in this particular project using a custom set of techniques we found suitable.

Environment description We first combine prestudy knowledge and requirement specification to define a physical view. This view describes the world the software is suppose to interact with.

User interface and conceptual model In these steps we try to conceptualize the user needs and desires for the system into a user perspective model and a graphical user interface. This step is somewhat inspired by affordance based design [22]; focusing on how the application should look from the user's perspective and the actual use of the it. The term affordance was originally introduced by Gibson [23, 24]. An affordance is an actionable property of a thing (usually a tangible object, like a software system). Naturally, designing

objects with as few negative, and as many positive affordances as possible is desired.

Module Overview Focus is then shifted onto the developers' perspective, describing high level architecture. The system is divided into separate modules, each representing some functionality. We focus on giving a short graphical description of how these modules interact, and a short textual rationale for each module. How should the architecture look? How should the software be structured? These questions will be answered in this section.

Low level architecture Modules are then designed in detail, and class chart and sequence diagrams are used to demonstrate how the architecture is intended to work. The class charts give an overview of the classes used to realize each module, and the sequence diagrams show how the architecture are used to realize the basic use cases developed earlier in the process. Questions asked here are e.g. what design patterns can be used?

Architecture summary Lastly, we go through the architecture with respect to the different quality focuses and design issues stated in section 9.3, to clarify the strategies used to address each of these. This process does not only serve to produce an overview for the documentation, but will also help us detect deficiencies in the architecture.

9.2 Architectural viewpoints

Several models exist that can be used to document software architectures. Examples are the Bass model [25] and the 4+1 architectural view model [26]. We have experience in using both, and they have different pros and cons for our project. Because they are designed in a generic way to support most design needs, there are some views and aspects that seem irrelevant for us. We therefore decided not to follow any of the models exactly, but rather use a mix of techniques that we master well, and that suits our project in a natural way.

9.3 Design issues and quality focus

One of the reasons for putting effort into analyzing and developing an architecture is to achieve qualities needed by the software system. Some qualities are more important than others, but it is imperative to realize that they affect each other. One quality is never achieved in isolation. In this section we will discuss the identification of important design issues and chosen quality attributes

9.3.1 System lifetime and time-to market

From a business point of view, the system lifetime and time-to-market are important factors in this project. System lifetime is important because when we make the effort to implement software for a specific use, we would not want to have to replace it within a short time period. Rather we want it to have a high degree of usefulness and remain credible for as long as possible in spite of eventual changes in the surrounding environment. Regarding time-to-market, the customer (Revolve) is expecting the system to be ready for testing and early use early April 2012 when the race car construction is scheduled to be finished. This places a heavy constraint on the development of the system.

9.3.2 Usability

Usability is a general goal in all software design projects, and this project is no exception. The target users of the system emphasize their concern for a user-friendly system. Usability is a rather wide term defined by Wikipedia as "the ease of use and learnability of a human made object" [27]. ISO defines it as "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use" [28]. In other words, usability involves studying principles behind a system's perceived efficiency or elegance; it deals with how a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a given context.

9.3.3 Modifiability

Modifiability naturally refers to the property of being modifiable. We want the components of the system to be easy to change, as every aspect of Revolve's datalogging system and practice is in fact not clear at this point. Therefore, it is important to be able to adjust the system if needed in test and deployment phases.

9.3.4 Modularity

Modularity builds on a design technique that composes software of separate, interchangeable components called modules [29]. Breaking down program functionality into modules is a conceptual way to improve maintainability among other things. The modules should each accomplish one function or a set of related functions, and contain everything necessary to accomplish this. Modular programming also helps separating tasks and concerns which is a good quality in team development, as implementation tasks can be performed individually and in parallel.

9.3.5 Performance and scalability

Performance is also an important factor to consider in the development of this system. The last thing the user wants is a piece of software working slowly, making the analysis job tedious and inefficient. Optimizing data management should therefore be a key focus, especially in the implementation phase. As for scalability, we need to make sure that the system supports managing as large amounts of data as the users of the system desires.

9.3.6 Portability and platform independence

Making the software cross-platform is desirable in this project to make the software easier accessible at almost any location (i.e. in the garage or at the race track, or on any of the students' laptops). That software is cross-platform means that it is able to run on more than one operating system or computer architecture. This can be a difficult task since the different operating systems have a different set of programming interfaces and APIs [30]. Solving this problem may require a lot of effort, unless a technology like Java Virtual Machine is used. This issue will be discussed further in chapter 12.5.

Chapter 10

System design

10.1 System environment

As we have seen in the prestudy phase, race car data analysis software are in its simplest terms a system that takes a collection of data delivered by a race car data acquisition unit and presents it in a meaningful context. The system to be designed in this project is no different, as we have seen in the requirement analysis phase. As shown in figure 10.1, the sensors placed in the race car deliver data to the datalogger, which is storing everything in files on a removable media. The files are then transferred to a computer for use in the analysis software.

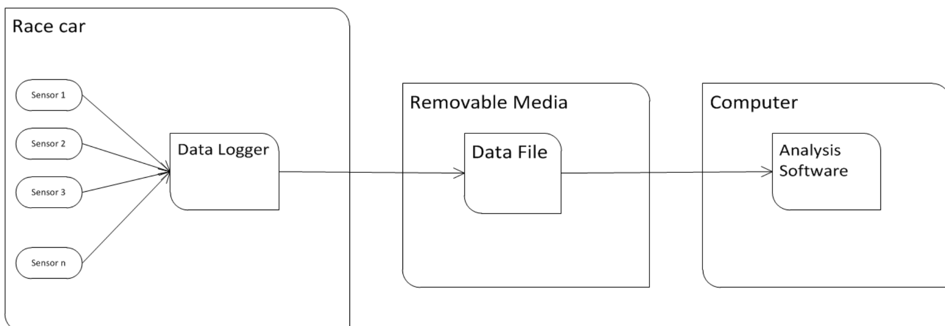

Figure 10.1: The Physical View.

10.2 User interface and conceptual model

In this section we present the decisions made in the process of designing the user interface and underlying conceptual model. The purpose of this is to establish a platform for further architectural development, where modules can easily be identified and described.

10.2.1 User interface

As we have seen when studying existing software in early phases, the different products have variations in the way of solving problems in the user interface. Interestingly, they mostly lack the possibility to configure the layout of different visualizations. We see this as a negative feature, as we want the user experience to be intuitive for multiple analysis tasks or purposes.

An example analogue to this would be a game of cards. If two card players are provided with an electronic card game device supporting one particular game, they would be forced to play that game forever. Give them a regular deck of cards, and the possibilities are endless. Furthermore, one should not use a sledgehammer to crack a nut. If the purpose of the work session is to analyze one specific parameter, and only that parameter, it would be great to have the opportunity to configure the user interface to show only the visualizations of interest instead of drowning the user in information.

In other words, a freely customizable interface would be a good way to improve the user experience and task efficiency; simply by eliminating the negative effects of using a static user interface. This approach was desired already in early meetings with representatives from Revolve during the requirement analysis process.

The idea of being able to move visualizations around resembles a graphical user interface known as an MDI (Multiple Document Interface) [31].

An MDI is a user interface that operates with multiple objects, or windows, placed within another window (in our case, visualization layout) [31]. The positioning of these windows can be configured by the user, preferably aligned to some grid, as seen in many popular software systems today. Without further explanation, we introduce the conceptual sketch of the GUI.

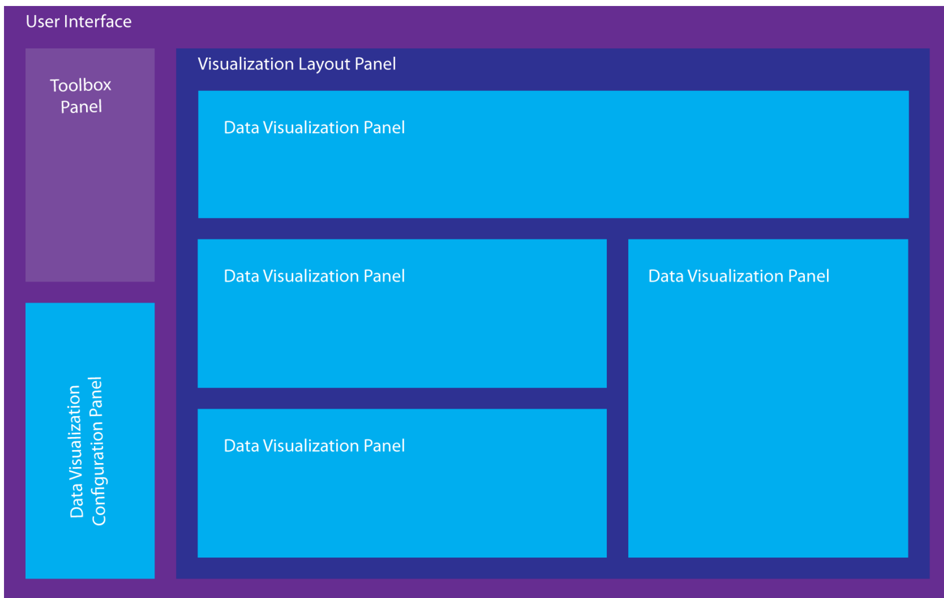

Figure 10.2: Conceptual sketch of the graphical user interface.

Data visualization panel This component will contain the actual data visualization, which is used for analysis. This can be e.g. a map, a data graph or video. The data visualization panels can be moved around in any configuration within the visualization layout panel.

Visualization Layout Panel This component serves as a canvas for placing data visualization panels onto. Since it is often desirable to work with several different visualization layouts at the same time in the data analysis, and be able to switch between them easily, several layout panels can be arranged in tabs.

Data Visualization Configuration Panel Each data visualization component will need a controller, to be able to set the different parameters for the component, e.g. selected laps and segments. These controllers will be docked into the data visualization configuration panel, which will switch controllers depending on which data visualization panel is having user focus.

Menus In addition, the main frame will have menus used to access functionality. These will be drop-down menus on the top of the frame. This is standard in most software applications and will therefore be easily understood by any computer user, opposed to other sorts of menus and toolboxes.

10.2.2 Conceptual system model

As seen from the users perspective, the system looks like something in figure 10.3. The graphical user interface with its layouts of visualizations sits on a project platform supporting all the functionality needed, like data structures and mathematical functions. The different data visualizations can be seen as small, independent packages of software running within the project environment, connecting to the project platform and GUI through a common interface. This type of small programs are commonly referred to as “plugins”, and we will refer to them as that hereafter. Furthermore, a visualization layout will be referred to as a "view" for simplicity (see figure 10.4). The figures in this section have been color coded to make it easier to distinguish the components in the different view perspectives.

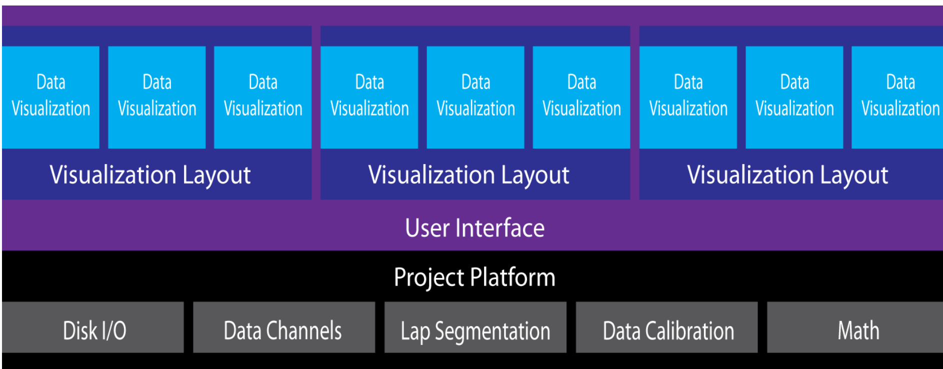

Figure 10.3: Conceptual sketch of the graphical user interface with the underlying project platform.

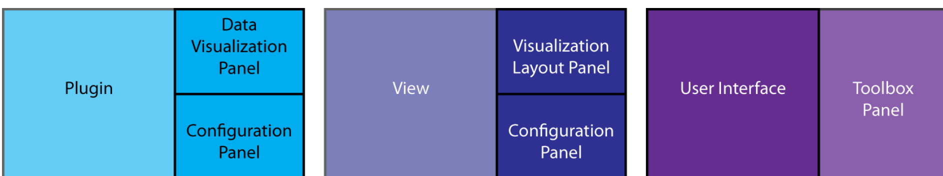

Figure 10.4: Conceptual sketch of the plugin, view and user interface modules.

Chapter 11

High level architecture

In this section we divide the desired functionality into modules. We first give a graphical overview with links representing usage relationships, and then give a short textual rationale for each module.

11.1 Modules overview

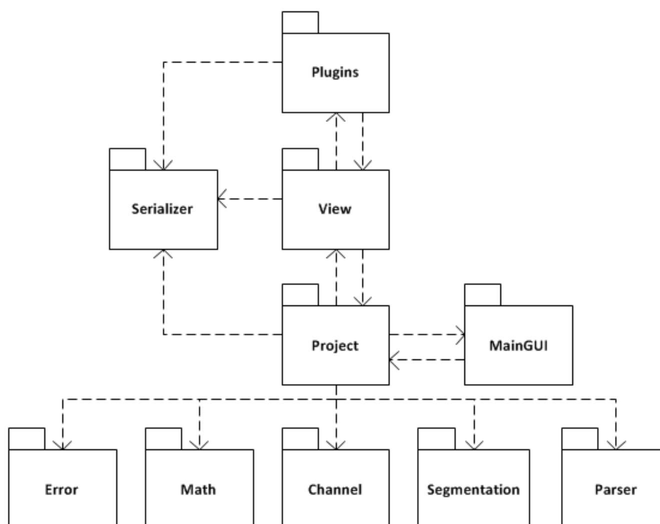

Figure 11.1: Graphical overview of the modules and the connections between them.

11.2 Modules rationale

This section contains a short textual rationale for each of the modules, with listings of module responsibilities where necessary. This documentation is used as a basis for the low level architecture.

Project module The project module is critical, and needed to fulfill most of the functional requirements. It serves as a representation of a work session (commonly referred to as a project). It is the heart of the system, acting as the glue between the different components, keeping track of project related data. More specifically, the project module needs to keep track of:

- Imported sets of sensor data in the form of a set of channels and math channels.
- The data segmentation setup.
- The set of desired views (visualization layouts).
- Actions done by the user (undo history).
- The notes written by the user during the work session.
- A customizable color configuration for channels and laps.

The project module must also be able to store the project state to a file system through the serializer module, and provide an interface for other modules to access the different resources.

Parser module The parser module will take care of parsing data files into data structures (channels) usable by the rest of the system. The module will fulfill requirement ID1, which is also a critical feature of the software.

Channel module The channel module represents the data structure the imported data and math channel data will be stored in.

Math module This module will be concerned with requirements dealing with heavier math, and we will gather all common math functionality here. The main responsibility of the module is to support calibration of the data. Examples of other possible uses are:

- Generating a track map.
- Fitting data values into a polynomial function.
- Generating math channel (user-modified channels) values.

Segmentation module The segmentation module shall take care of splitting the channel data into laps and lap segments, and is meant to fulfill the requirements regarding segmentation. The requirement specification encompasses both manual and automatic segmentation, and the specific responsibilities will be:

- Generate segmentation offsets based on manual input.
- Automatic segmentation of lap data.
- Keeping the offsets in a data structure.

Error module The error module is responsible for bookkeeping of all errors and exceptions that may occur at run-time. It also takes care of providing feedback to the user when errors occur. No specific functional requirements are relevant here, but it adds to usability.

Serializer module The serializer module will be responsible for storing plugin objects, views and projects to a persistent file system. Naturally, this module is also responsible for reverting this process.

MainGUI module The main GUI deals with all the functionality regarding the main frame of the user interface. This encompasses providing the user with a intuitive user interface (see figure 10.2) containing multiple views aranged in tabs, tool menus, and wizards for different standard routines.

Plugin module The plugin module is responsible for the plugin architecture that we will use to implement the different data visualization plugins. This module will take care of the loading and instantiation of external plugins and their depencies. Also, to ensure that every plugin will fit into the system, the plugin module provides a predefined way of communicating with them.

View module The view module represents the layout panels. One of the main responsibilities of the view module is layout persistence. By layout persistence we mean the ability to remember how the different visualizations are sized and positioned within the layout. The view module should also support functionality for a user friendly plugin arrangement process. It is to be designed in a way similar to the plugin, as we would like to be able to implement several types of views for different modes of plugin arrangement (snap to grid, docking etc). However, it will not be designed to support extensions compiled separately like the plugins.

11.2.1 Specific plugins

In addition to the core modules, we will also design a set of plugins to support specific visualization requirements.

Linechart plugin The linechart plugin will be the module used for showing standard data graphs, plotted against time or distance.

XY-chart plugin The XY-chart plugin is the module for showing any data channel plotted against another.

Map plugin The map plugin is concerned with creating and showing a map of the race track. This module will also be used as a tool for manual segmentation

of the laps.

Notepad plugin For conveniently taking notes and storing them with the project, the notepad plugin will be designed.

Playback plugin The playback plugin will be the module that is responsible for running a real time simulation of the data, where the different visualization plugins are synchronized with video.

Report plugin The report plugin is responsible for generating and showing data channel reports. As mentioned earlier, a channel report should contain minimum, maximum and average values for laps and segments.

Video plugin To be able to show and synchronize recorded video from the vehicle, we will design a video playback module.

Time slip plugin Lastly, the time slip plugin will be the module for showing time slip lines for the different laps of imported data.

Chapter 12

Low level architecture

In this phase of the project, we develop the architecture of the different modules in further detail to provide sufficient information for the implementation phase. The first section contains a short discussion on our research done in the problem area of plugin architectures. This research is included at this point, because it deals with plugin architecture frameworks that may have impact on the low level design. A description of each module, accompanied by class charts is then provided to give an overview of the core architectural guidelines. Lastly, we include the sequence diagrams for core functionality drawn during this low level design phase.

12.1 Plugin architecture

The idea of plugins have been around since the 1970s, when EDT Text Editor provided the ability to run an external program from within the editor, that was granted access to the text editor's memory buffer, thus allowing it to manipulate the current text session. Plugins are now commonly used in software systems, e.g.in web browsers, to play video, scan for viruses or display new file types. Examples of such plugins are the Adobe Flash Player and Microsoft Silverlight. Another good example is Microsoft Office, which also makes use of plugins to extend the abilities of the application.

Figure 12.1 displays a basic plugin architecture. The advantages of using a plugin architecture are clear. First of all, it simplifies the process of creating and deploying extra abilities to extend the software. It will also enable third party developers to create specialized abilties for their own purposes, without knowing much about the rest of the system. A plugin architecture can also be used to reduce the size of an application by letting some features be optional, and providing the opportunity to leave these out. However, this might not be relevant for this project, since the software will deal with data and video files much larger in size than the software

itself. Lastly, a plugin architecture can help resolve software licensing issues, by separating source libraries incorporating conflicting types of licenses.

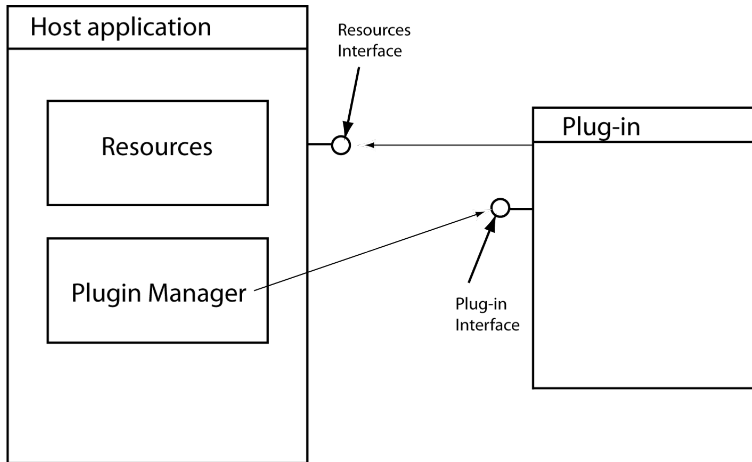

Figure 12.1: Basic plugin architecture

Several frameworks exist for plugin architectures and applications extensions. Examples we have examined are Java Plugin Framework (JPF) [32] and Open Servies Gateway initiative (OSGi) [33]. Frameworks also exist for other programming languages, but since we already have chosen Java as the programming language, see section 12.5, we only considered the frameworks written for Java.

After careful consideration we found the frameworks available, especially OSGi, to be a little too heavyweight for our purpose. They incorporate, among other things, security modules for restricting what plugins are allowed and what they have access to. This functionality is mostly designed for security critical applications like web browsers, and we see no use of this functionality in our project. The alternative that suited our project the best was JPF, or JSPF (lightweight version of JPF) , but it seems like the JPF project have not been active for a while, which makes using it a risk. A discontinued framework project may have undesired long-term effects on the system, as the JVM and other possible dependencies are in continuous development. We therefore chose to continue working on the architecture we started developing in the in-depth project. This decision was not only based on the lack of plausible open source frameworks, but also the fact that plugin functionality is an interesting concept we are eager to learn more about by implementing it ourselves.

12.2 Modules

We have not included class diagrams and descriptions for each of the specific plugins, because these all build on the same architecture. The way a specific plugin shall interact with the plugin framework is shown in figure 12.13. Other than that, each plugin implementation will encompass the two GUI components (controller and display) required and other dependencies.

12.2.1 Project and support modules

In these modules, there have not been used any specific architectural patterns. Most of the classes listed are abstract classes that do calculations on a given set of inputs (Calibrator, Parser, MathFunction, Segmentation), or data structure classes (Error, Polynomial, Channel and Project).

Project module

Project The project class should have attributes with getters and setters for each field, as shown in figure 12.2. When we apply the Model-View-Controller (MVC) pattern here, the Project class will serve as the model. In addition, the project class provides the undo and redo queue logic, and storage to disk through the serializer module.

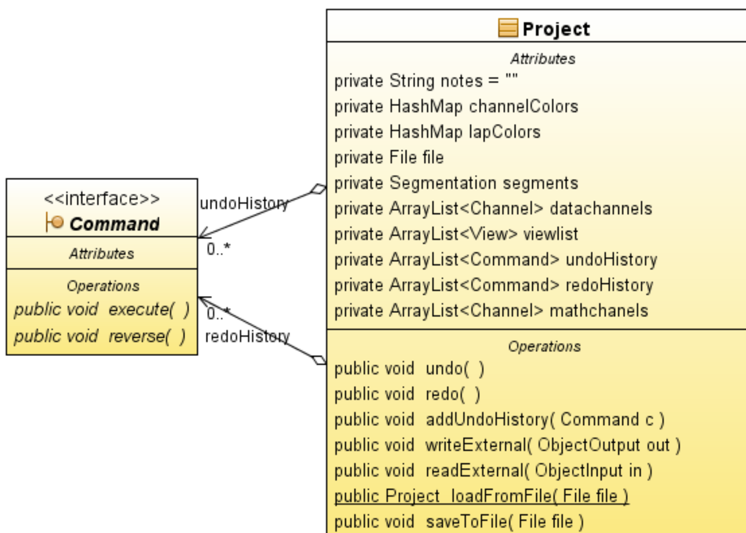

Figure 12.2: Class diagram for the Project module

Command This interface is the basis for the undo functionality in the system. It incorporates the command pattern, and wrap method invocations for executing and reversing the command inside an object.

Channel and parser modules

Parser This class will contain the logic for reading data logs from disk into channel structures, see figure 12.3.

Constants The constants class holds predefined index values for important channels to give the entire system global pointers to where (which index) the specific channels are found.

Channel This is the data structure holding the parsed data. It shall support data calibration through the math module, and therefore it will keep to sets of data. One set of raw data and eventually one set of calibrated data. If no calibration is set, getCalibratedValues() will return the raw data. The proper way to access the values of a channel is therefore calling getCalibratedValues().

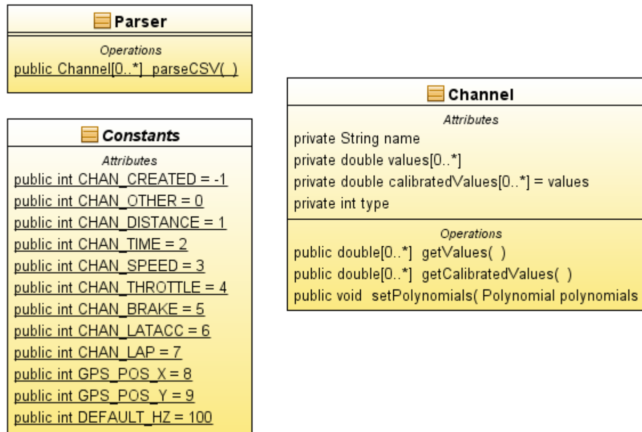

Figure 12.3: Class diagram for the parser and channel modules

Math module

MathFunction The MathFunction class contains methods for creating math channels. The class chart, see figure 12.4, gives an indication to what math functionality our math channels are planned to support. It also makes use of the map data structure to calculate a track map.

Map The map class is the datastructure used for map calculation.

Polynomial The polynomial class is a datastructure for representing polynomials assigned to channels. The polynomials will be used by the calibrator to calibrate them.

Calibrator The calibrator class contains logic for calculating a channel's calibrated values given its set polynomials.

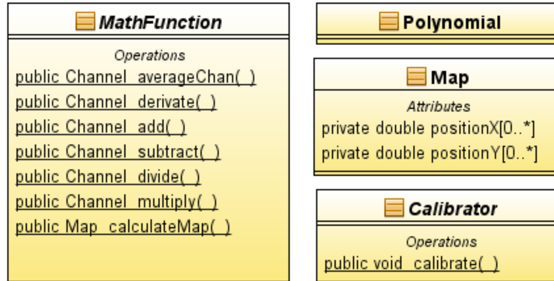

Figure 12.4: Class diagram for the math module

Segmentation module

This module is simply a data structure consisting of laps and segments.

Segmentation This is the main class representing a segmentation, keeping a list of lap objects and various convenient methods to access the data structure. In addition, the segmentation class contains methods e.g. for generating segments for all other laps based on one segmented lap.

Lap A lap object consists of a startOffset, an endOffset, a lap number to identify it and a list of segment objects.

Segment Similarly, a segment objects consists of an offset, and a number to identify it.

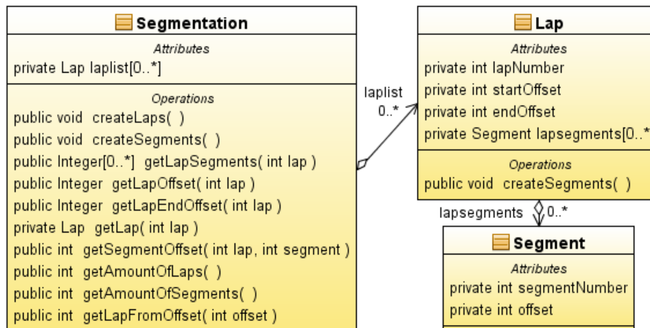

Figure 12.5: Class diagram for the segmentation module

Error module

ErrorLogger The error logger contains methods for adding errors to the log

Error Data structure for errors, containing a type, name and a message text.

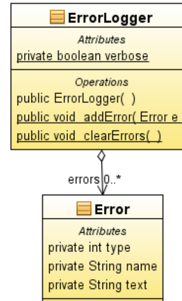

Figure 12.6: Class diagram for the error module

Serializer module

Serializer This class will contain methods for storing objects to disk, and for retrieving them from disk. This class will also support serializing objects to an intermediate format for further storage inside other objects. A basic and intuitive file structure for storage is shown in figure 12.8. A project file must contain the general properties of the project (e.g. segmentation), a list of data channels, and a list of all the registered view objects. Similarly, a view file must contain general view properties, some layout persistence object and a list of registered plugin objects.

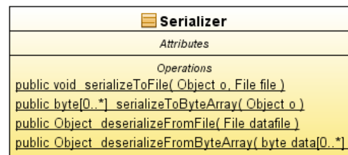

Figure 12.7: Class diagram of the Serializer module

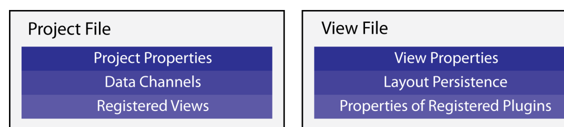

Figure 12.8: Conceptual sketch of Project and View file structure.

12.2.2 Main GUI

The MainGUI module will contain many classes to be able to implement a decent and sophisticated user interface. We have not accounted for these here, as none of us have much experience from Java GUI programming. We therefore make use of an interface to ensure that the main gui component can be easily replaced, without touching the rest of the source code. When applying the MVC pattern, the main GUI acts as both view and controller.

MainGUI The interface covers basic functionality of the main gui that any implementation must implement for the software to work. Examples are shown in the class chart, like adding views to tabs, knowing what is the current viewed tab and the ability to set which plugin to display properties from in the plugin property panel.

MainGUIImpl This will be the implementation of the main GUI.

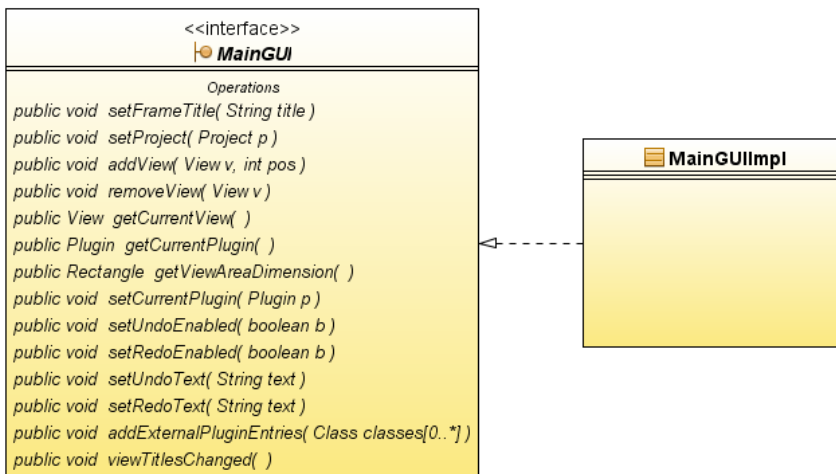

Figure 12.9: Class diagram of the Main GUI module.

12.2.3 Plugin module

The core elements in the plugin module are the plugin interface and the base class DefaultPlugin implementing this interface. As an example, see figure 12.13, the base class is extended by the linechart plugin. The linechart plugin overrides empty method bodies to implement its functionality, and eventually also extends the basic functionality to support update of e.g. plugin names in the GUI. If a method is not needed by a certain plugin, it should simply not be overridden.

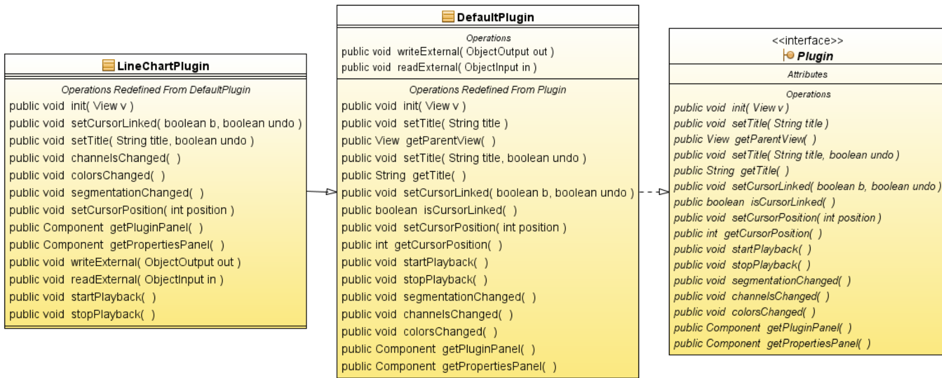

Figure 12.10: The Plugin module's core classes.

The second class diagram, see figure 12.11, shows the components designed for loading plugins and instantiating them in a generic way.

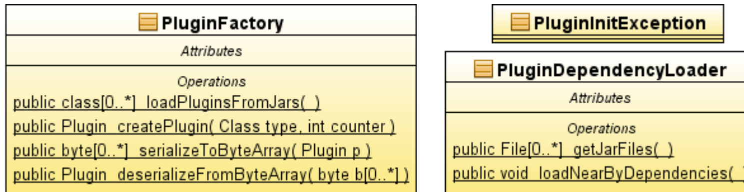

Figure 12.11: PluginFactory and PluginDependencyLoader

Plugin Interface The interface shall support functions for getting and setting the general plugin properties. These are the plugin title, the current cursor position, and a link cursor property. The link cursor property decides if the plugin should respond to synchronization attempts from the view. The interface should also contain methods for: initializing the plugin after its instantiation, delivering GUI components to the main GUI, updating its visualization based on changes in the project and a way for the plugin to deliver its user interface components to the main user interface, as seen in the class chart. Also, methods for starting and stopping playback have been added to support synchronized simulation in plugins who require to start its own thread (e.g. video).

PluginFactory We will use the factory method pattern [34] for creating the different plugins. The plugin factory will be responsible for all instantiation of plugin objects based on class objects. It will also implement methods to serialize plugin properties through the serializer module to a format ready to store to disk.

DefaultPlugin DefaultPlugin is the base plugin class. It implements basic func-

tionality for partial initialization, setting and getting title, cursorposition and the link cursor parameter. It shall also extend an externalizable interface, to enable serializing of the configured plugin properties through the serializer module. Specific plugins, or subclasses, will override the methods for reading and writing to the stream for including their own parameters.

PluginDependencyLoader The plugin dependency loader shall have methods for detecting jar packages and dependencies in a plugin folder. Furthermore, it shall load the jars into the current system dynamically and provide the PluginFactory with a set of plugin classes derived from the jar files.

PluginInitException If something goes wrong during the initialization of a plugin, this component can be used to cancel the process and produce an error report.

12.3 View module

As the class chart in figure 12.12 shows, the architecture of the view module is not very different from the plugin module. The same reasoning lies behind the use of an interface, the view factory and default view class. However, we have chosen to leave out the ability to load specific view packages externally. Several types of visualization layout tools (views) are less likely to be designed by others in the same manner as visualization plugins.

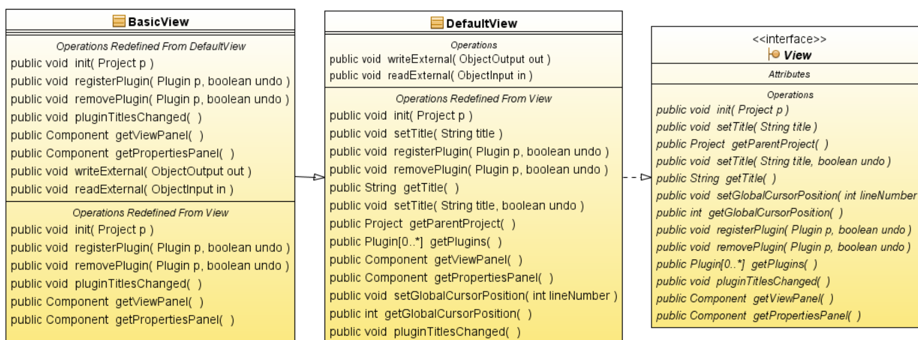

Figure 12.12: The View module's core classes.

View Interface The interface shall support functions for getting and setting properties, and a way for the view to deliver its user interface components to the main user interface.

ViewFactory The view factory incorporates the factory method pattern like the PluginFactory, and will be responsible for all view instantiations, also having

convenient methods for serializing the view with its list of plugins, layout and eventual other properties.

DefaultView Like the DefaultPlugin class, this class shall implement the functionality that is common for all views. This includes getting and setting of properties, delivery of user interface components, registering of plugins and synchronization of the registered plugins. Like in the plugin module, specific view implementations will inherit from the DefaultView to simplify and ensure successful development of new views.

ViewInitException The view initialization exception is used for the same purpose as the plugin initialization exception; to catch errors in the initialization procedure. This may happen e.g. if a plugin in the view that is being initialized fails.

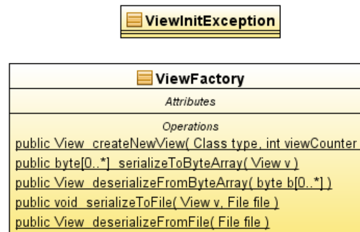

Figure 12.13: ViewFactory and ViewInitException

12.4 Sequence diagrams

To further explain and verify the architecture we have included some sequence diagrams that relates directly to the use cases discussed in section 8.5.2. By developing these sequence diagrams we also gain an advantage because we are continually verifying that the different tasks needed are actually supported by the architecture.

SC1 demonstrates the instantiation of a project object, and the use of channels and the parser to read the data from disk. Also, a channel is calibrated using the math module. SC2 demonstrates how a view is created through the view factory, and added to the current project. SC3 shows how a plugin is created and configured. This is the most interesting one, as it both shows the creation of plugins and illustrates the MVC pattern, including undo-functionality. Lastly, SC4 shows the storage of a project object, and retrieval of a view through the serializer module (combining use cases UC4-5 from the requirement specification). These sequence diagrams are quite self explanatory but we also include a short textual description for each of them.

SC1: "Create project and Import/Calibrate Data"

The user requests to create a new project through the user interface, a new Project instance is created and set to be active in the user interface. When the user requests to import data through the user interface, it tells the project object to import data, which is done through the parser module. The user then sets a desired calibration for a channel, and the channel object's polynomials are set. The calibrator class in the math module then calculates the new values and stores them into the channel object.

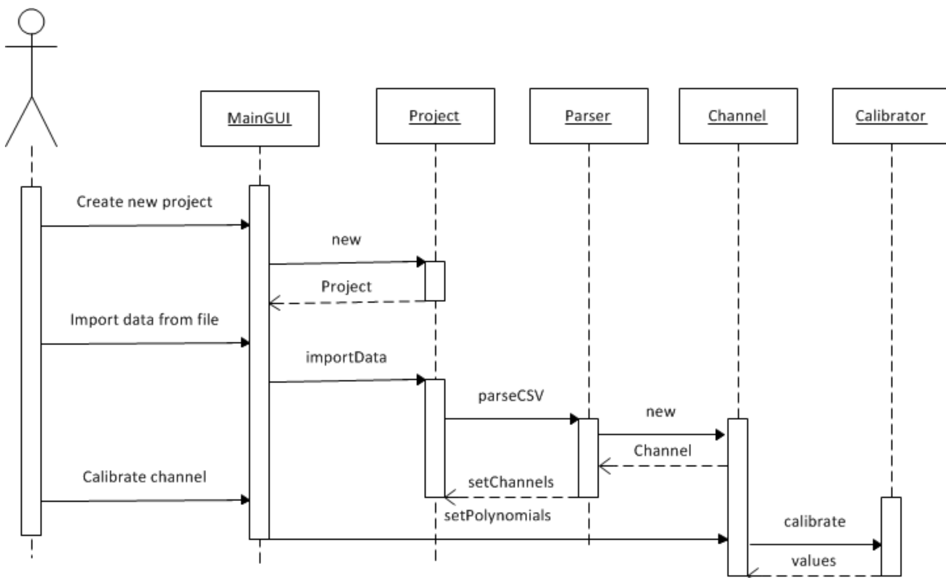

Figure 12.14: Sequence diagram of Use Case 1 (UC1).

SC2: "Create layout"

The user requests to create a new view through the user interface. The instantiation of a view happens through the ViewFactory, and the view is then added to the user interface. To add it to the user interface, the MainGUI needs to fetch the GUI component of the View (getViewPanel). This way of adding the view to the main GUI is analogue to adding a plugin to a view.

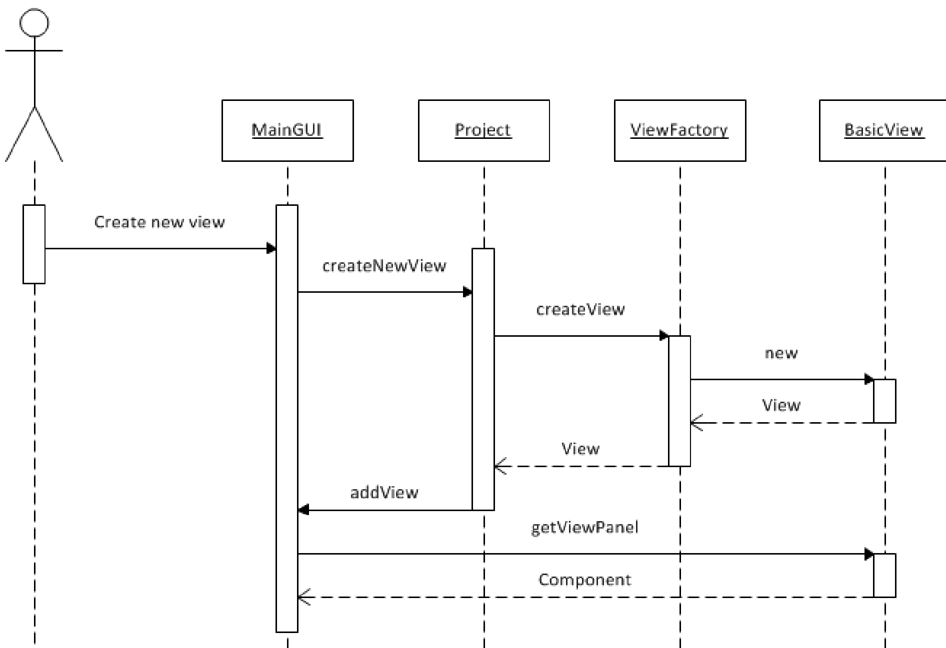

Figure 12.15: Sequence diagram of Use Case 2 (UC2).

SC3: "Create and configure visualization"

The user requests to create a new Plugin via the user interface. The instantiation of a Plugin happens through the PluginFactory, and the Plugin is then registered in the target View. The View then needs to fetch the GUI component of the Plugin to have something to show. When the user requests to change some property of the Plugin, the property pannel calls setProperty in the Plugin logic and the Plugin's GUI panels are updated accordingly through the MVC pattern.

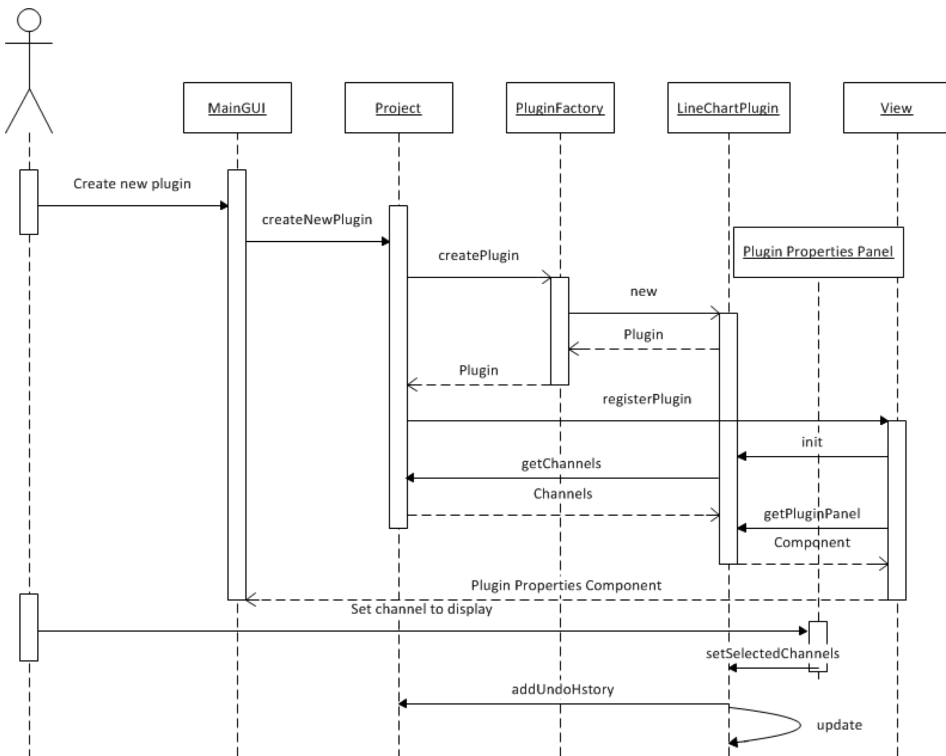

Figure 12.16: Sequence diagram of Use Case 3 (UC3).

SC4: "Save and load layout"

The last sequence diagram shows how the serializer module can be used for retrieving projects, views and plugins, and also storing them. Since the sequence charts would turn out to be very simliar, we have included the process of saving a project, and loading a view in the same chart to demonstrate both these functionalities. First off, the user requests to save the current project through the main gui. The project class' saveToFile method shall then be invoked. The project then requests from the serializer to write to disk. The serializer then makes use of the writeExternal method implemented in the project class, to 'flatten' the project and store it to disk. The user then requests the loading of a view from disk through the main gui. The project module then asks the ViewFactory class to instantiate the serialized view. The ViewFactory also makes use of the serializer module and the readExternal method implemented by the view class.

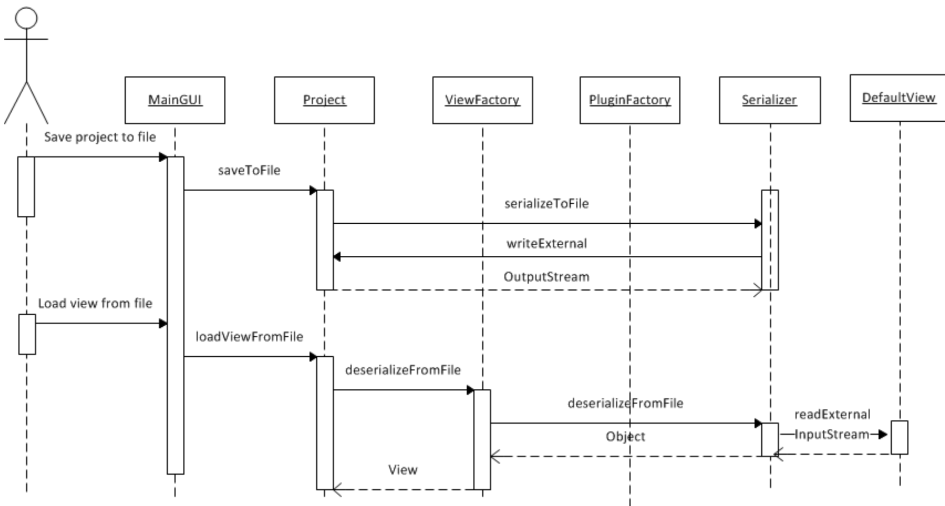

Figure 12.17: Sequence diagram of Use Cases 4 and 5 (UC4-5).

12.5 Choice of programming language

In this section we describe the reasoning behind the choice of programming language which was done during the in-depth project. The choice of programming language plays an important role in the development of software, and there are many factors to consider, e.g. platform support, performance or experience with the language and the tools used. After considering the pros and cons, we chose to use Java for our software.

Pros

- Java is a language both group members are very comfortable with.
- It's the language we have the most experience with from both school, jobs and personal use.
- The GUI usability in this software is very important to the Revolve group. Swing provides extensive functionality, and there are a vast amount of libraries and open-source code that we can make use of, instead of having to "reinvent the wheel". This saves time.
- Java can be used on multiple platforms, which the Revolve group sees as practical, and we see this as one of our main design issues. Java does this by abstracting away the differences of the platforms. This does comes with a performance penalty, but the alternatives for platform independence are much more demanding in terms of man-hours [30].
- Most of the Computer Science students at NTNU have a lot of experience with Java. This is a plus since there might be more school projects done in relation to this software.

Cons

- Java performance might not be as high as the performance of our second choice, which was C++. Despite this, the performance of Java has been improving the past few years [35, 36, 37] and we did not see this as a problem. Also, because of the addition of biased unlocking [35], Java has better support for multicore processors.
- Memory usage in Java is in general higher than in C++.

By the end of the day, Java was the natural choice. We both have some experience with C++, but not as much as with Java. If we were to use a different language than Java or C++, we might have had to learn the basics of the language before we could start with the project itself. This would not have been feasible considering the small amount of time available for a development project of this size.

It should also be noted that we will be using the newest version of the Java Development Kit. At the time of writing, this is JDK 7 Update 4. At least Java 7 Update 1 is needed in order to make full use of the software we are developing.

Chapter 13

Architecture summary

In this chapter we review the planned architecture and summarize how we have attempted to address the design issues and quality focuses.

To address time-to-market and product lifetime issues we have used a modular programming approach. This way we can implement working parts of the systems, and add new functionality in increments. In the case of a delayed final product, we are then able to deliver a partial product that works. In order to achieve a long product lifetime, trying developing a system with high modularity and maintainability is important.

We have thought of several different strategies to ensure a high usability. We believe that having an intuitive layout of the GUI and avoiding non-obvious functionality is a solid approach to usability. Also the addition of the undo-functionality and user feedback on errors improves the user experience. We also believe that the high configurability of the visualization layouts makes the software usable for more analysis purposes. Lastly, it is worth to mention that a customizable interface is not alpha omega. Much of the usability of this software lies in the implementation of the different plugins. Therefore we have put som extra effort into discussing possible designs with representatives from Revolve.

To improve the modifiability and maintainability of the system, in addition to the modular programming approach, we will use a Model-View-Controller pattern as the basis for our architecture. This pattern separates system logic and data from user input and from rendering, i.e. the GUI.

To address the modularity quality focus, we have introduced the use of interfaces. This is a common approach to modular programming and the design of plugin architecture [38]. The issue of portability and platform dependency is partially solved by choosing Java as programming language with its virtual machine (JVM) solution.

Chapter 14

Implementation

In this chapter we will give a brief overview over how the system was implemented, and the results from the tests of the system. We will also mention open source libraries used for the implementation of the different components.

14.1 Introduction

The implementation was a major part of this project. This chapter will contain the lowest level view of the system with code examples and discussions around specific classes. For further details, see the JavaDoc documentation and source code.

14.1.1 Approach

As discussed in the first part of this report, we have chosen to divid the implementation phase into so-called sprints [7]. A sprint is a way of classifying tasks in Scrum. One sprint finishes a certain portion of the end-product, and contains a number of tasks [7]. However, we are only using the sprint and backlog concept and do not adopt all the principles of Scrum. At the end of each sprint we do a wrap-up check on what has been implemented successfully and what has not.

14.1.2 Testing

The testing in this project has been mostly ad-hoc testing [39]. It is a less structured approach to software testing, but enables us to find important defects very quickly.

The method of testing we have mainly used is so-called “white-box testing” [40]. This is a method of testing which is used to test the internal structures of an application. It has been applied on a unit testing level, where we test the individual classes and their functions [41, 39].

In addition we’ve had some alpha and beta testing of the software. Revolve had their alpha test in November 2011, during the in-depth project, and then a quick beta test in form of a SUS-questionnaire in May 2012.

Towards the end of the development we also did some basic black-box testing [41, 39] to verify which requirements had been fulfilled. This is in many ways the opposite of white-box testing, since the person testing does not have access to the internal structures and the source code of the application while running them.

14.1.3 Open Source

Throughout this project we’ve been using some open source software in the form of Java libraries. In this section we will quickly go through the different licenses pertinent to this project.

Open Source is a development method that focuses on releasing software that has an open source code and is freely redistributable [42]. This enables many end-users from all over the world to collaborate on developing software for themselves, instead of buying proprietary software from manufacturers.

The first signs of open source licensing was started by Richard Stallman in the early 1980s, when the Free Software Foundation [43] was started [44]. In 1998 several computer hackers launched the Open Source Initiative [42]. In present day the community has grown considerably, with sites like SourceForge [45] having over 100,000 registered users and 10,000 open source projects available [44].

GNU General Public License

The GPL is one of the most common licenses for open-source projects. This license grants a wide range of rights to developers. A developer can legally copy, distribute and modify software under this license [46], but if a modified version of GPL licensed software is released the new version also have to be GPL licensed.

GNU Lesser General Public License

LGPL grants fewer rights than standard GPL. The LGP is often used for libraries that want to allow linking from non-GPL and non-open-source software. However, an advantage with LGPL is that libraries licensed with it can be included in software that is not LGPL licensed.

14.1.4 Test data

In order to properly test our software, we need some sort of input data to analyze. Since the race car is not finished and cannot gather test data, we needed to make our own. At first, we used random numbers generated with a Java algorithm. These worked well to test the parsing, initialization of data objects and graph drawing. However, in order to generate the map and better evaluate the graphs, we needed some actual racing data.

Live For Speed is an online racing simulator [47]. Since it has a realistic physics engine and advance functionality, such as being able to record laps, this proved to be an excellent tool to generate test data. After we recorded some laps, we used F1PerfView[1] to convert the Live For Speed recordings to CSV files [47]. These could then be read in our software and were used as test data throughout the implementation and test phases.

We also used Live for Speed and a tool called Fraps [48] in order to record videos. These videoes were then synchronized to the data we had from the game and used to test the video plugin.

14.2 Sprint 1

The first sprint we performed had focus on the core of the software; the functionality of the underlying data framework presented in the design phase. In this section we will go into detail about the modules implemented during this sprint and the tests that we executed.

14.2.1 Goals

To use as a sprint backlog, we created a list of specific goals for the sprint in advance. We plan to fully implement support for the following functionality:

- A working object class for representing projects or work sessions
- Framework for undo-functionality for property changes
- Possibility for parsing a CSV file of raw data
- Data structures for the raw data
- Math module (for calibrating data, generating math channels and track maps).

[1] F1PerfView is a telemetry viewer. It can view data from racing simulators, then convert this data to a format of choice.

- The data structure representing the segmentation of raw data into laps and lap segments, and functions for generating segmentations.
- An error handler logging system errors and giving feedback to the user.
- A generic way of storing projects or view layouts to disk for later use.
- A working plugin framework with support for the loading of external plugins (.jar files).
- A “base-plugin” built using the plugin framework.
- A working View interface
- A “base-view” built using the View interface

14.2.2 The project module

The project class is implemented according the specification given in the low level architecture phase. It contains the different data fields required of a project, and corresponding getters and setters. We follow the MVC pattern; the set methods all contain the possibility for eventual calls to GUI and plugin updates. The project class contains logic for organizing the commands in corresponding undo- and redo-queues. Finally, the project class also implements the Externalizable interface provided by the Java framework for use by the serializer module to enable storage of the project object. This will be discussed further when implementing the serializer.

The most interesting part in this module is the components dealing with the desired undo-history functionality. As Java does not support passing method calls as parameters, we implement this by using the Command interface, incorporating the command pattern. Objects implementing this interface contain two methods, execute() and reverse(), for wrapping method invocations representing the property change. All set methods for properties that should support undo functionality should make use of an undo-parameter, and the implementation could look something like this:

```
public void setTitle2(String title, boolean undo) {
        if (!undo) {
            //Change the model
            //Notify changelisteners
        } else {
            final String oldValue = this.getTitle();
            final String newValue = title;
            final <Object> source = this;
            Command c = new Command() {
                @Override
                public void execute() {
                    source.setTitle(newValue, false);
```

```
                }
                @Override
                public void reverse() {
                    source.setTitle(oldValue, false);
                }
                @Override
                public String toString() {
                    return "Change Title";
                }
            };
            c.execute();
            project.addUndoHistory(c);
        }
    }
```

The command objects are instantiated as anonymous objects, so that the method bodies can be defined directly in the code without creating a new class for each property change. The new command object is then passed to the project class for bookkeeping. As this method relies on object memory references it will not be possible to store and re-use the undo-history together with the project. This could be solved by using absolute references to the different objects instead of the dynamic memory references. However this is seen as a unnecessary feature and will not be implemented. Lastly, the toString() method returns a textual description of the command used to show what the next undo or redo step would be in the main gui.

14.2.3 Parser and channel data structure

The parser of the software is, in this case, meant as the class that parses the raw racing data files and initializes Java objects from these. We have spent a lot of time on this class in order to optimize the parsing and initialization of raw data, since this is quite time-consuming with very large data sets. Performance was of high priority to the Revolve Group. The formal requirement was no more than 5 seconds loading time in order to parse a file, initialize objects and calibrate the data. Here is an example of raw data in CSV (Comma-Seperated Values) format:

```
Distance,Time,Speed,Steering,Throttle
0.350609,0.000000,133.152415,-0.825073,1.000000
0.719796,0.010000,133.219748,-0.825073,1.000000
1.089049,0.020000,133.287149,-0.825073,1.000000
1.458383,0.030000,133.354578,-0.825073,1.000000
```

In order to read this, the parser first must find the names of the channels. These names are located on the first line without comment-symbols (*, / or %), usually line one. Then, in order to determine the amount of samples we use BufferedInputStream and read bytes, counting lines as we see a newline symbol. This is very fast and takes well under 100ms, even for the largest datasets. This makes the

rest of the process faster by pre-allocating space in the arrays that store values. Then each line is read, split at the comma and each value is parsed from String to Double. This is then stored into a channel data structure.

Once all the data is read and store into data structures, the parser checks for a calibration property-file. This is an optional part of the data importing process. It was added so that users can define their own rules for calibrating specific channels, instead of having to add polynomials manually after the data is read. Here is an example of how a Calibration property file looks like:

```
Gear=1
Speed=0|1.50
Lateral G=5.55e-10|1.25e-5
```

This means that the channel named “Gear” will have 1 added to it and Speed will be calibrated by the polynomial $0+1.5*x$, in other words increased by 50%. Lateral G needs more exact calibration and will use the polynomial $5.55e-10+1.25e-5*x$. The structure then is Name=Polynomial 1|...|Polynomial n-1|Polynomial n.

For the parser and the structure of the analysis data we started with channel objects, which in turn holds ArrayLists of doubles for the values and StringReader to read the data. This combination worked well, and we had no performance issues with the test data given to us by Revolve. However, worst-case scenario performance tests were too slow. This scenario is 20 sensors collecting data for 40 minutes at 100Hz (this is 240 000 samples, times 20 sensors) and took about 7400ms of complete loading time, with calibration. The structure was converted to channel objects holding arrays instead of ArrayLists of doubles for values, and BufferedReader to read with String.substring() for splitting the values. These changes gave loading times, with calibration, under 4000ms. Memory usage also went down, and seems to be about maximum 90MB in the worst-case scenario.

14.2.4 Calibrator and math channels

The sensors used on our race car usually measure and capture Voltage values. This means that even if the sensor is for oil temperature, the value it captures is for instance 4,33. This value could translate to 90 degrees celsius. Because of this the raw values will have to be calibrated before they can be put to use.

From the requirements, it was important that the user could add polynomial functions for calibrating the sensors. Example of a polynomial function used for calibration:

$$f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1} + a_n x^n$$
$$f(x) = 10 + 0.34x + 0.0043x^2 + 1.334x^3$$

When calibrating the values from a polynomial function, we start with the last polynomial (in the example that would be 1.334). This is set as the "result". The

result is then set to the value to be calibrated (let's use x = 10) times the current result, plus the next polynomial (in this case 0.0043). This is then repeated for as many times as there are polynomials.

$result = poly_n = poly_3 = 1.334$
$result = x * result + poly_{n-1} = 10 * 1.334 + 0.0043 = 13.3443$
$result = 10 * 13.3443 + 0.34 = 133.783$
$result = 10 * 133.783 + 10 = 1347$
$y = 1347$

This is done for every sample. The function is fast, since it does not use the standard Math.pow(x, n). This method is in the Java API and is used to give the result of x^n. We started out using the pow() method, but experienced an overall calibration performance increase of about 70% when switching to the current method.

14.2.5 Segmentation

Without any sort of segmentation, all the data is just a continuous stream of the values. This class makes it impossible to calculate in what section of the track something happened, or even in what lap it happened. We implemented the datastructure described in the low level architecture to hold the different laps and segments, and a class with functions to segment the data. The structure consist of an ArrayList of laps which each has their own ArrayList of segments, in other words the data is split into laps and the laps are all split into segments. Both the segments and laps have an integer value called offset, this is where the lap, or segment, starts in the array of samples. The program does not need to store where the segments end, since this is stored in the next segment.

To segment the laps is easy, since we have a data channel which purpose is to keep track of what lap the car is currently on. This is done through use of a lap beacon. The beacon is set where the lap begin/end, and each time the car passes it registers the beacon and ups the lap-counter by one. The segmentation module can then use that data channel to see at what offset the value increases.

When finding offsets for the segments, however, input from the user is preferred. The user knows best how many segments is preferred, and where the track should be split into sectors. The way we do it is as follows:

1. The user splits one lap into sectors using the map plugin.
2. Percentages of the total lap distance are calculated from how long the sectors are.
3. These percentages are then used to split all the other laps into sectors of the same size, relative to the length of that lap.

Time Report - Track Sections (All Laps)

	Lap 1	Lap 2	Lap 3	Lap 4	Eclectic	Rolling Minimum
Str 0-1 End En (End)	0:11.623	0:11.495	0:12.138	0:11.427	0:11.427	0:11.427
Turn 1	0:06.148	0:05.867	0:05.888	0:05.920	0:05.867	0:05.920
Str 1-2	0:02.574	0:02.493	0:02.545	0:02.532	0:02.493	0:02.532
Turn 2	0:02.598	0:02.533	0:02.521	0:02.606	0:02.521	0:02.606
Turn 3	0:02.857	0:02.656	0:02.730	0:02.725	0:02.656	0:02.725
Str 3-4	0:05.777	0:05.519	0:05.586	0:05.321	0:05.321	0:05.321
Turn 4	0:01.866	0:01.798	0:01.850	0:01.797	0:01.797	0:01.797
Turn 5	0:01.522	0:01.449	0:01.477	0:01.429	0:01.429	0:01.429
Str 5-6	0:07.494	0:07.098	0:07.729	0:07.207	0:07.098	0:07.207
Turn 6	0:03.099	0:03.249	0:03.044	0:03.220	0:03.044	0:03.220
Turn 7	0:02.740	0:02.721	0:02.757	0:02.612	0:02.612	0:02.612
Str 7-8	0:03.595	0:03.527	0:03.779	0:03.414	0:03.414	0:03.414
Turn 8	0:05.565	0:05.459	0:05.569	0:05.396	0:05.396	0:05.396
Str 0-1 End En (Start)	0:07.698	0:07.813	0:07.573	0:08.145	0:07.573	0:07.573
Totals	1:05.163	**1:03.682**	1:05.192	1:03.759	1:02.653	1:03.186

Figure 14.1: Example of split, or segment, times in MoTeC i2 [19].

This is done since each lap will never be the same length. There will be certain changes in how the driver hits the turns, the speed he is driving at and other factors. Let's say one lap is 15 meters longer than the other lap. If we were calculating the sectors based on distance, instead of % of distance, the 15 meters would be added to the final sector automatically. This would make the other sectors appear faster than they were, and the final sector appear slower than it were. Using percentages of distance, these 15 meters will be spread out between all the sectors, giving more correct timing data.

Another way to split the track into sectors would be the same way that we split the data into laps. Have a sector beacon at every sector, and a data channel to record it. Unfortunately the Revolve team does not have enough beacons for this.

14.2.6 Error module

The error module is implemented as a static class that will be accessible from anywhere in the source code. An example of usage would be:

```
ErrorLogger.addError(new Error("View error", "View failed to initialize. "
                     + "View initialization aborted."));
```

This code adds a new view error to the error log and gives the user feedback in the form of a popup dialog.

14.2.7 Serializer

The serializer class makes use of Java Serialization API [49]. This API provides functionality to flatten objects into serialized data ready for e.g. storage to disk. Any object that is to be serialized must implement the Serializable interface. However, if we use this interface, the entire object will be serialized. We are not interested in storing every field in a object. GUI components for instance, are not necessary to store.

This can be solved in two ways. The first possibility is to mark every field not to be stored with the 'transient' keyword. This requires careful inspection when any new field shall be added to an object, which is not a desirable feature. The other alternative is to make use of the Externalizable interface, which allows us to define how the object shall be stored by picking the required fields ourselves. This interface incorporates tho methods readExternal(InputStream in) and writeExternal(OutputStream out) for reading and writing. The project object, view objects and plugins will all implement these methods. Below is an example, showing how the base plugin class will store and retrieve its settings.

```
@Override
public void writeExternal(ObjectOutput out) throws IOException {
    out.writeObject(this.title);
    out.writeBoolean(this.cursorLinked);
    out.writeInt(this.cursorPosition);
}
@Override
public void readExternal(ObjectInput in)
            throws IOException, ClassNotFoundException {
    this.title = (String)in.readObject();
    this.cursorLinked = in.readBoolean();
    this.cursorPosition = in.readInt();
}
```

As well as supporting writing the objects to a file system, we also implement the possibility for serializing them to an intermediate format (byte arrays) to be able to duplicate objects, and store them inside other objects. To clarify, plugins can be serialized to byte arrays and stored in a view object, which can again be serialized to new byte arrays and stored in a project object.

14.2.8 Plugin framework

The plugin module architecture have been covered well in the earlier phases. Its implementation is following the architecture specification closely. Therefore we have chosen to leave out most of the implementation details here, and rather focus on the challenges we had while programming it.

The plugin framework makes us of Java Reflection [50]. Reflection is an advanced

and powerful technique allowing applications to perform operations that would otherwise be impossible. It is normally used by systems which need to modify their behavior during runtime.

PluginFactory is implemented making use of this technology, and it can be used to create object instances in a dynamic fashion (e.g. different types of plugins). PluginFactory is able to instantiate objects from their class files directly by using the forName() and newInstance() methods provided by the reflection API.

It is also possible to scan for classes implementing the plugin interface while the program is running, and therefore no hardcoded installation of new plugins will be needed. We have implemented the PluginDependencyLoader, which searches the working folder (the folder the applications runs from) for eventual jar packages containing plugins or dependencies. The jar package has to have the same name as the full classname of the plugin, including the package names. For instance, "plugin.testplugin.TestPlugin.jar".

Adding new locations to the class path is problematic, as the addURL() method of the system classloader is protected. One way to solve this is by subclassing the classloader, taking advantage of its delegation model to dynamically add new class paths. This is the common approach but we found a simpler way, using reflection to access the addULR() method of the system classloader. How this can be done is shown in the example below.

```
URLClassLoader classloader =
        (URLClassLoader)ClassLoader.getSystemClassLoader();
Method method = URLClassLoader.class.getDeclaredMethod("addURL", URL.class);
method.setAccessible(true);
method.invoke(classloader, urls);
```

Finally, the plugin base class implements the Externalizable interface required to store itself. The specific plugin implementations will override these methods for adding its own specific data fields. Here, a problem arises. If a plugin component is updated after the user have used the previous version to save projects, the serializer interface will no longer recognize these, because the serialVersionUID property of the class has changed. This is solved by forcing the serialVersionUID field of the class to one specific value. Furthermore, the versioning of plugin properties are solved by using a versioning parameter that is stored with the rest of the fields.

14.2.9 View interface

Because the architecture of the plugin and the view module is so similar, the implementation will also be similar; the view module is also implemented according to the architecture specification, by using an interface and a default base class. The ViewFactory is also realized using Java reflection, and we refer to the plugin module section for information on that subject. Like the plugin module, the view module also has a base class, implementing the Externalizable interface.

14.2.10 Sprint wrap-up

Most of the goals set in the startup phase of this sprint are accomplished, and we are ready to move on to the next sprint. Eventual details regarding each goal are given below.

- A working object class for representing projects or work sessions OK
  The class is fully implemented, with all specified functionality.
- Framework for undo-functionality for property changes OK
  The framework is ready for use by any component who requires this and can communicate with the current project object.
- Possibility for parsing a CSV file of raw data OK
  The parser is working perfectly for the test data we are using, and the performance is excellent.
- Data structures for the raw data OK
  The channel data structure is implemented, with all required functionality.
- Math module (for calibrating data, generating math channels and track maps) OK
  The calibration and math channel functionality is up and running, but calculation of track maps have been postponed until the implementation of the map plugin.
- The data structure representing the segmentation of raw data into laps and lap segments, and functions for generating segmentations. OK
  The module responsibilities regarding the automatic lap segmentation and distinguishing between laps without the use of external beacons have not been implemented yet.
- An error handler logging system errors and giving feedback to the user OK
- A generic way of storing projects or view layouts to disk for later use OK
  The serializer module is ready for use by any component implementing the externalizable interface.
- A working plugin framework with support for the loading of external plugins (.jar files) OK
  We have done several tests regarding the instantiation of plugin objects both internally (from within the source packages) and externally (from .jar files). The plugin framework is working as intended.
- A “base-plugin” built using the plugin framework. OK
  The base plugin implements common functionality and is ready to be subclassed by specific plugin implementations.
- A working View interface OK
  The view module is implemented with an interface to ensure that the view

components are easily interchangeable.

- A "base-view" built using the View interface OK
  The base view implements common view functionality and is ready to be subclassed by specific view implementations.

14.3 Sprint 2

The second implementation sprint of the project was focused around the GUI and the implementation of a user-friendly view component.

14.3.1 Goals

Again, we list the specific goals we aim to reach during this sprint. We are to implement a working main GUI that should provide the user with intuitive components for:

- Channel configuration (including creating math channels)
- Channel calibration
- Importing data
- Saving and loading projects and views
- Undo / Redo buttons
- Showing views able to contain plugins
- Showing a properties panel (controller) for a plugin
- Drop-down menus for access to functionality

14.3.2 MainGUI

To implement the graphical user interface we used standard Java Swing components either as is, or by extending them with some functionality. As none of us have any real experience in using this framework, some study work was required to implement all the desired functions. To our relief, The GUI code was mostly generated using the NetBeans IDE, which simplifies the process of doing the design layout and implementing eventhandlers like actionlisteners. This helped us save a lot of time.

We also used a panel docking framework for the plugin properties panel. The docking framework used is called DockingFrames [51] and it's designed for convenient arrangement of panels within a GUI frame. Because we are using this, the addition of extra GUI panels at a later point will be easy to implement, and

easy for the user to configure to his or hers liking. We also tried out another docking framework called MyDoggy [52]. MyDoggy has the advantage of having the same look and feel on all platforms. However, we quickly decided not to use the MyDoggy framework because it was showing buggy behavior and was tedious to customize. We found the DockingFrames framework to be much more suitable for our use. Both are licensed under LGPL 2.1 [53].

Core system functionality were organized in drop down menus, see figure 14.2. The plugin dropdown menu also supports dynamic registering of plugin entries, which will be done at startup if the plugin dependency loader have located any usable .jar files nearby.

As the rest of the system, the main GUI takes part in the MVC pattern, which means it implements methods that can be called by the various setters in the models.

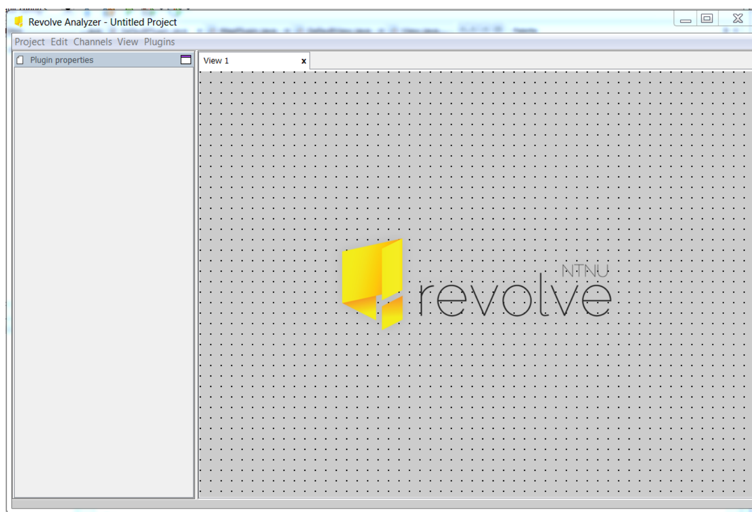

Figure 14.2: The implemented main GUI frame with a basic view component

14.3.3 The BasicView component

Trying to implement a prototype of a basic view component, many approaches were tried out and discarded. First off, we tried to use open source docking frameworks As discussed earlier in the report, some functionality for user aid when arranging plugin panels are desired, like "snap-to-grid" functionality or similar. To realize this we have looked at possibilities for using the docking framework from the main GUI implementation. However, after some consideration we decided that we should implement a simpler interface ourselves. The main reasons we chose to abandon

the docking type of views are the disadvantages of having to fill the entire screen at all times, and the fact that the aspect ratio of the plugin view panels would not be preserved, see figure 14.3.

When it comes to layout persistence, the basic view stores the layout persistence by gathering relative window coordinates in a hashmap.

Finally, we implemented the basic view by using a normal JDesktopPane with a set of JInternalFrames. Simple snap to grid functionality was implemented using windowlisteners, overriding the windowMoved and windowResized events. We also added the Revolve logo in the background of the view to improve the look and feel of it. The result can be seen in figure 14.2.

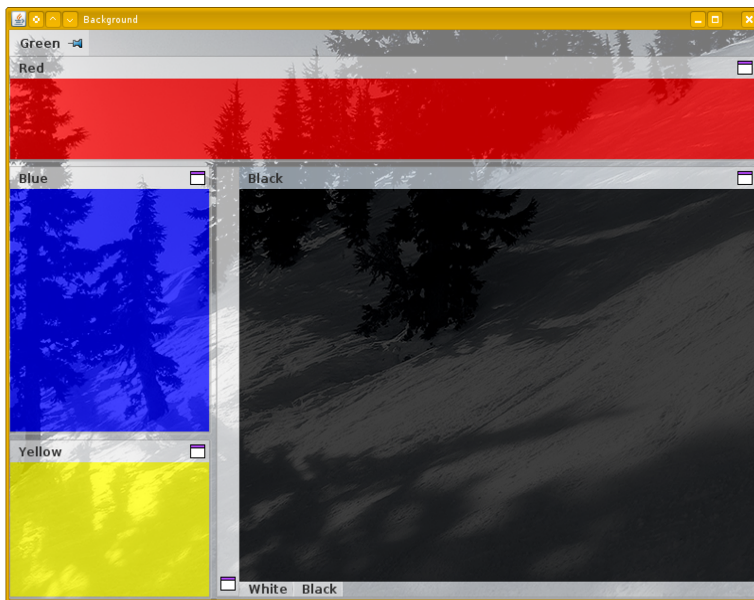

Figure 14.3: A demonstration of the DockingFrames framework.

14.3.4 Sprint wrap-up

Again, we address each of the sprint goals, and how we successed in reaching them. In general, we have implemented all the goals set for the sprint, but in some areas we see room for future improvements. A good example of this would be to include more advanced help wizards for the different functions which will improve usability.

- Channel configuration OK
- Channel calibration OK

- Importing data OK
- Saving and loading projects and views OK
- Undo / Redo buttons OK
- Showing views able to contain plugins OK
- Showing a properties panel (controller) for a plugin OK
- Drop-down menus for access to functionality OK

14.4 Sprint 3

The third sprint is focused around the development of the plugins for the software. This is a crucial part of the project, because what may be seen as the real functionality in the software is realized here.

14.4.1 Goals

The goals in this sprint is to develop working prototypes of the following plugins:

- LineChart
- XYChart
- TimeSlip
- Map
- Video
- Playback
- Notepad
- Report

14.4.2 Linechart, XY-chart and TimeSlip plugins

The linechart, xy-chart and timeslip plugins are so similar that we discuss them all in one section. They all share the same ability to plot data points onto a 2D-chart. However, in spite of their similarity and the possibility to include all three functionalities into one plugin, we have chosen to make separate plugins for simplicity.

The linechart plugin keeps a set of selected channels, selected laps and segments, and which domain to plot the selected channels in (distance or time). Chart series

are then generated, segment by segment, to enable lap and segment color coding when they are superimposed onto each other. Otherwise, the channels decide the color coding. If any property is changed, the series will be rebuilt.

Similarly, the xy-chart plugin keeps two channels, that are to be plotted against each other. Channel coloring and superimposing laps is not relevant for this plugin, since no single channel will be drawn.

The timeslip plugin calculates the time differences in multiple laps for different positions. We solve this by generating a time difference channel for each base lap. Since the different laps differ in length, in the terms of data line numbers, we have to search the distance values to find corresponding lap positions. Again, we use optimized search functions like the binary search. In spite of this, the process is time consuming, so we have added a dialog indicating progress if it takes too long. However, when testing, we never experienced waiting any longer than 3 seconds.

Currently, the time difference channel is generated on each change of the base lap. This can possibly be improved by calculating them all at once in the initialization phase, or adding this functionality to the core modules of the software. After creation, the generated channel is plotted like a normal data channel against the distance channel like in the linechart plugin, representing the time slip traces.The only configurable properties of this graph will be lap coloring and base lap, in addition to the base plugin settings.

All three plugins support being synchronized to the view's global cursor, and can also send synchronization attempts to other plugins when the cursor is set by the user. To accomplish this, mouseclicks must be detected, and the coordinates must be translated into offsets in the data channels. This is solved using binary searches for the time and distance channels, and this puts a constraint on the input data the software can handle. If the driver is going in reverse at some point of the track, these algorithms will not work. However, this is very unlikely to happen.

To draw the different data charts, we used an external open source library for Java called JFreeChart [54]. This library is released under the Lesser GPL license 2.1 [53]. JFreeChart provides a nice and flexible API for drawing different types of data visualizations such as like line charts, XY graphs and sectordiagrams. It is an active project still releasing new versions. We will continue to use it since one of the core elements of our program is the drawing of graphs, and we enjoy using it.

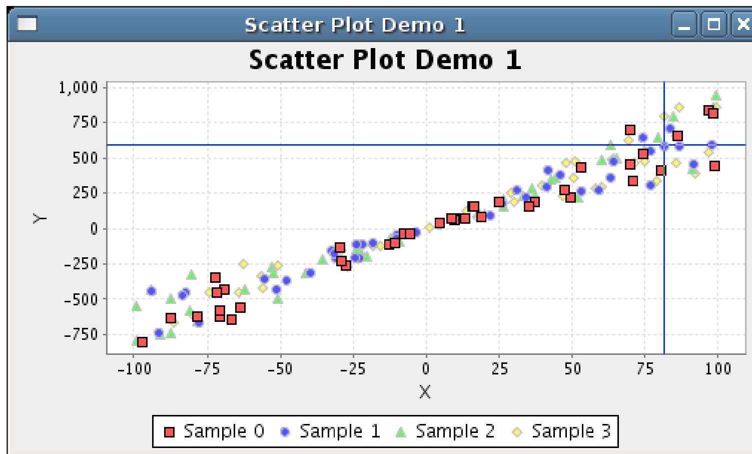

Figure 14.4: JFreeChart's support for plotting dots. This will be a good alternative for an XY-graph [54].

JFreeChart makes use of collections of XYItems to draw the graphs. This requires us to load channel data into this datastructure, before any graph can be drawn. This takes time, and leads to a slowly performing system, having redundant data in memory. To solve this, we have implemented a class 'DirectXYSeries' that overrides the functions in the XYItemCollection class to map values directly to our data channel structure instead. This showed to improve performance drastically.

Furthermore, using a custom series for delivering data to JFreeChart, we can add extra features, like defining start and stop offsets for the series and whether it should superimpose itself onto other laps. In other words, when the 'superimpose laps' setting is chosen in the linechart plugin, no data is recalculated or reloaded, only the superimposed parameter is set, and the graph can be redrawn, since the mapping class returns other values once this parameter is set. Another feature this solution supports is the opportunity to include a parameter deciding the draw density of the graph. This is a necessary feature to be able to increase performance when the graphs are to be animated in the simulation/playback. In addition to moving the cursor and the zoom range, we also implemented support for dividing the chart background into segments for better readability. This adds to the complexity of every refreshed animation frame. Other efforts to improve the drawing performance therefore included disabling anti-alias, doublebuffering and the drawing of grids.

Finally, the plugin property panels are implemented, containing the different settings. To implement checklist and color choosers we have used custom combinations of swing components. Colors are retrieved from the global project coloring scheme. Also, since these plugins require more settings and parameters, the property panels have been arranged in tabs to simplify access to them.

14.4.3 Map plugin

We also implemented another plugin, namely the track map. The implementation of this plugin was more complex, and needs to be elaborated.

As discussed earlier in the report, using GPS data is the easiest and most accurate way of generating track maps. However, since not all cars are equipped with a GPS sensor, or if GPS is unavailable for some reason, we have implemented a backup function. We added compatibility for calculating maps from lateral g-forces and speed. This functionality is in the math module. The way the calculations are done is the following:

1. Convert all units to SI-units: speed = m/s, g-force = m/s^2.
2. Find the radian of the turn: $speed^2/lateralG$.
3. Find the change of angle for this sample: $(speed/radian)/samplingHz$.
4. Find the speeds in direction x and direction y: $speed{*}cos/sin(currentAngle)$.
5. Add the change of angle to the current angle for next iteration: $a = a + \Delta a$.
6. Find the x and y positions: $position_x = position_x + speed_x/samplingHz$. Same for y.

This is done for every sample in the data. Then the positions are drawn on a map, and the end result is shown in figure 14.5. When using the lateral g-forces and

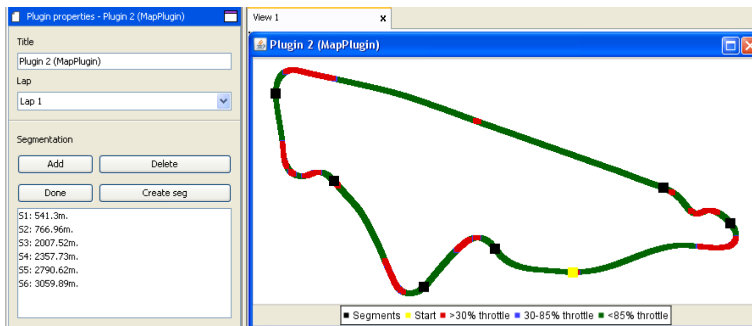

Figure 14.5: The map generated by the Map plugin from raw data.

speed, there are some error sources. Firstly, if the tires lose traction in a turn, the lateral g-forces on the car will be almost zero. This can be felt as a passenger in a car if the car loses control on a slippery surface. Another source of error is banked turns. A banked turn is a turn where the road banks, or inclines, sideways. The reason for banking a turn is to generate less lateral G and more comfortably turn at higher speeds. If there is a sensor that measures the downwards g-forces, then

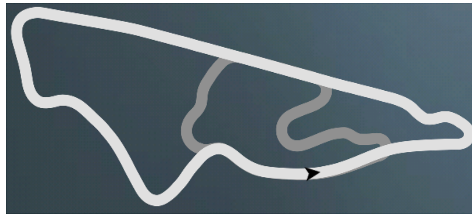

Figure 14.6: The actual map of the track.

data could be capture in the turns and used to get a more correct map calculation. So far an installation of such a sensor has not been planned in the Revolve car.

To solve this we smooth the map after creating it. This is also done by the math module. The approach is simple:

1. Find the difference between start X position and end X position for the given lap.
2. Do the same for Y.
3. Divide this difference with the amount of samples.
4. Add that result to each and every map point.

It is possible to change what lap to look at. This can be a nice feature to see how consistent the laps are. The user can also use the map plugin to segmentate the laps using the “Properties Panel” which can be found on the left in figure 14.5.

14.4.4 Report Plugin

The report plugin is designed to give a concise summary of either the segment times or the values for every channel in each segment. The segment times can be shown as total time for each segment or as the time slip on each segment compared to the fastest segment time. The best times will be highlighted. The channels can be shown as the average, maximum or minimum value for the selected channel. A quick example could be checking out the lowest, or minimum, gear that can be reached in the different segments and compare these for every lap.

Looking at figure 14.7, two different reports can be seen. The bottom one is a split time report, while the top one is the minimum speed achieved. The segments, S#1, S#2 and so forth, are the same ones that we created with the map plugin.

14.4.5 Video Plugin

Often it can be hard to get a feel for what exactly is happening on the race track, especially if there is only graphs, lines and maps to look at. The video plugin

Plugin properties - Test

Title

Test

Report from ...

Speed

Averages

Maximum

Minimum

Show split times

View 1

Laps/Segments	S#1	S#2	S#3	S#4	S#5
Lap # 1	38.38	73.46	66.65	67.01	80.82
Lap # 2	52.11	67.51	64.26	74.25	81.45

Laps/Segments	S#1	S#2	S#3	S#4	S#5
Lap #1	27.61	**32.2**	**15.29**	16.35	**17.75**
Lap #2	**27.02**	32.78	15.56	**16.16**	18.07

Figure 14.7: An example of two reports.

gives the user a unique perspective on the data by showing a synchronized video together with the graphs and maps. By using the playback plugin, see section 14.4.7,you can simulate the actual run. A video of this simulation is included in the delivery.

The video plugin works by using a library called vlcj [55]. vlcj is a Java Framework for the VLC Media Player [56]. It is licensed under GPL 3 [46]. VLC 2.0 or newer needs to be installed on the system that is using the video plugin. Also, if the computer is running 32-bit JVM, then it needs to run a 32-bit VLC, and the other way around. As can be seen in figure 14.8 the video plugin has some basic media

Figure 14.8: A screenshot from the video plugin.

player controls. These are useful if the user wants to watch the video on its own,

but the most common usage is to play the video synchronized with the rest of the plugins. There also appear to be a few overlays in the video (split time, laps, map) but this is simply because the video used is recorded from the video game Live for Speed.

14.4.6 Notepad Plugin

A basic notepad plugin was also created. It can be used from anything to writing details about the run to simply thoughts about how the car is performing. This

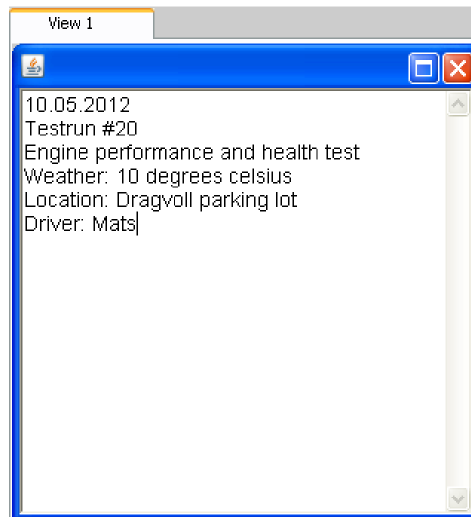

Figure 14.9: A screenshot from the notepad plugin.

plugin was the easiest to implement. It's simply a TextArea that can be resized as the user wants and has word wrapping. The notes the user makes are saved within the project itself, not as a part of the plugin. This plugin is unique in that there can only be one notepad plugin running for each view, but since it uses the notes from the project notepad plugins in different views will show the same text.

14.4.7 Playback

The playback plugin is responsible for performing a real time simulation, synchronizing the different visualization plugins. This is done by creating a thread that keeps sending sync attemps (setCursorPosition calls) in short intervals. The thread makes use of the Swing.InvokeAndWait parameter, to make sure very plugin updates itself before continuing. To sync with the time channel the plugin uses the system timer to check how long the updates took, and thereby determining

which offset to sync to in the next synchronization step. The playback currently does not sync against the actual time channel to improve performance. Instead it relies on that the data comes in a given frequency. This also puts a constraint to the format of the input data, like the search algorithms based on sorted lists used in the chart plugins. As mentioned earlier, performance is an issue when doing this kind of simulation, especially since the JFreeChart panels are not designed for animation purposes. To monitor the performance, we therefore implemented an FPS estimator, to keep track of how many updates the we managed to do pr second. Generally, the performance is acceptable (50fps) under the current test conditions (Core2Duo E8400, 4gb RAM running Windows 7). But we see drastic changes as we add more plugins to the view or maximize plugins to cover the entire screen.

14.4.8 Sprint wrap-up

This was the final implementation sprint. We have reached most of our goals within this sprint. Details on functionality:

Linechart

- Multiple laps superimposed. OK
  Can enable superimposing of laps.
- Lap and channel coloring. OK
  Channel color changes work in LineChart and globally.
- Multiple laps continuous. OK
  Can disable superimposing of laps.
- Select segments. OK
  Checking and unchecking laps and segments enables and disable them, respectively, in the graph
- Select different X-axis. OK
  Both distance and time work as an X-avis
- Global cursor. OK
  Changing the cursor in Linechart changes it in other plugins as well, if they have "Follow cursor" checked
- Select multiple channels. OK
  Checking and unchecking multiple channels enables and disables them, respectively, in the graph.
- Saving/loading. OK
  All information is saved and loaded.
- Segmented background. OK
  The plugin has a segmented background.

XYChart

- Select different X and Y-axis. OK
  Changing X and Y-axis updates correctly in the graph.
- Change lap color. OK
  Graph is correctly affected by lap color changes
- Global cursor. OK
  Reacts to changing cursor in other plugins and other plugins reacts to changing cursor in XY graph
- Saving/loading. OK
  All information is saved and loaded.

TimeSlip

- Select different base laps OK
  Setting the base laps triggers calculation of time differences and rebuilding the graph.
- Select different laps to compare OK
  Selecting laps to compare adds them to the graph with color coding corresponding to global project settings.
- Global cursor OK
  Reacts to changing cursor in other plugins and other plugins reacts to changing cursor in the timeslip chart.
- Saving/loading OK
  All information is saved and loaded.

Map

- Generate map from g-force and speed. OK
  Smoothing and map generating works well.
- Generate map from GPS data. NOT TESTED
  No GPS test data available.
- Create segments. OK
  Segmentation works correctly.
- Delete segments. OK
  Deleting segments work and updates in other plugins.
- Generate overlay from throttle. OK
  Different throttle responses give the correct overlay color.
- Saving/loading. OK
  Plugin is opened but what lap you are on is not loaded correctly.

- Global cursor. OK
  Changes lap and updates position. Can also send global cursor to other plugins.
- Channel change listeners. OK
  Updates when changes are sent.

Video

- Play video. OK
  Playing video works, but does seem to lag when playing videos encoded with Windows Media Player.
- Synchronize video. OK
  Synchronizing work, negatives values as well.
- Global cursor. OK
  Starting and stopping from Playback plugin works, but does not sync global cursor from linechart.

Playback

- Global cursor. OK
  Sends cursor changes to all plugins.
- Properly synced with time OK

Notepad

- Saving/loading. NOT OK
  Was later fixed.
- Line-wrap. OK
  Wraps long text lines.
- Resizing window. OK
  Line wrap updates when resizing window.

Report

- Generate split times. OK
  Split times generated and best ones highlighted.
- Generate average, max and minimum stats. OK
  Works for all channels.

14.5 Completeness

In this section we will test the software against the requirement specification. We conduct simple blackbox testing to verify that the required functionality is

available and working as desired. First off, we present the results of the testing, and then we address any requirement that shows to be problematic or not fulfilled.

Results

In this section we will give information about which requirements reside in which modules and if they are fulfilled or not.

Table 14.1: Requirements results.

ID	Priority	Core Modules	Result
ID1	C	Parser, Channel	Fulfilled
DC1	H	Report plugin, Math	Fulfilled
DC2	M	Parser, Channel	Fulfilled
DC3	M	Math, Channel	Fulfilled
DC4	H	Math, Channel	Fulfilled
DC5	M	Math	Not fulfilled
DC6	H	Segmentation	Fulfilled
DC7	L	Segmentation	Not fulfilled
DC8	H	Project, Plugins	Fulfilled
DC9	L	Segmentation	Not fulfilled
GUI1	H	View	Fulfilled
GUI2	L	MainGUI	Fulfilled
GUI3	H	MainGUI	Fulfilled
GUI4	L	MainGUI	Not fulfilled
GUI5	L	View	Fulfilled
GUI6	H	Map plugin, Math	Fulfilled
GUI7	M+	Video plugin, Playback plugin	Fulfilled
GUI8	L	Video plugin	Not fulfilled
GUI9	M	MainGUI, Plugins	Fulfilled
GUI10	M	View	Not fulfilled
GV1	H	Linechart plugin	Fulfilled
GV2	C	Linechart plugin	Fulfilled
GV3	M	Timeslip plugin	Fulfilled
GV4	H	Linechart plugin	Fulfilled
GV5	H	Linechart plugin	Fulfilled
GV6	M	View	Fulfilled
GV7	M	Linechart plugin	Fulfilled
XY1	M	XY-chart plugin	Fulfilled
XY2	H	XY-chart plugin	Fulfilled
XY3	H	XY-chart plugin	Fulfilled
XY4	H	XY-chart plugin	Fulfilled

Continued on next page

Table 14.1 – continued from previous page

ID	Priority	Core Modules	Result
O1	M	Project	Fulfilled
O2	L	MainGUI	Not fulfilled
O3	M	Project, Serializer	Fulfilled
O4	L	Project	Not fulfilled
O5	M	Notepad plugin	Fulfilled
O6	L	JFreeChart based Plugins	Fulfilled
NF1	M	System	Fulfilled
NF2	L	System	Runs under Windows and Linux
NF3	M	System	Fulfilled
NF4	H	System	Fulfilled
NF5	H	User guide	Fulfilled
NF6	H	System	Fulfilled
NF7	H	Process	Fulfilled

Discussion

Here we will explain why certain requirements were not implemented.

DC5: Graph smoothing This requirement was not specifically implemented due to lack of test data from the car, making it hard to test and adjust the smoothing or datafiltering. JFreeChart does however have some built-in support for smoothing graphs.

GUI10: Global zoom GUI1 is about a global, synchronized zoom that can affect all the plugins. This proved very hard to implement and since it was not a high priority it was postponed.

The other requirements that were not realized are:

DC7 Lap segmentation without the use of sensors/beacons

DC9 Automatic detection of curves and straights for lap segmentation

GUI4 Multiscreen support

GUI8 Video overlays

O2 Customizable hotkeys

O4 Autosave

All of these were assigned Low priority (out of Low, Medium, High, Critical). They were considered bonus features from the beginning, and because of the limited timeframe of the project these requirements were not addressed.

Other comments:

DC3 The math channel functionality is working, but it is not possible to store a matchchannel for use in other work sessions.

GUI2 Currently, the only way to duplicate a view is to save it to disk, and then load it back in.

14.6 Deployment

The deployment part of this project was not executed for several reasons. The car that Revolve Analyzer is supposed to analyze data from did not get finished in time. This means that we did not get raw data from the car's data loggers in time test the software with real-life data, put the finishing touches on it and then deploy it properly.

At the time of writing the software is simply distributed as a .jar file with a user guide and some example data.

Chapter 15

User guide

Here we will give some practical insight into how our software is used, with screenshots.

15.1 Dependencies

Firstly, in order to use the program, the computer will need to have Java 7 Update 1 or newer installed. This can be downloaded at http://www.java.com. In order to use the video plugin, the computer needs to have VLC 2 [55] or newer installed. This can be downloaded at http://www.videolan.org. If the computer runs 32-bit Java, the computer will need to run the 32-bit VLC version, and the other way around.

There are also some restrictions to the input data that should be paid attention to. The Lap channel has to be present in the data and increment positively and discretely, i.e. go from 1, to 2, to 3 and so forth. The same goes for the Distance channel, although it does not have increase discretely. The Time channel also has to be present, and needs to rise continuously and with a fixed amount (the data has to have a static sample rate). In other words, if the data is in 100 Hz, the time channel has to increase by 0.01 seconds every sample. It's also important that the Time channel is in seconds. This means that Time and Distance is not reset to 0 between laps.

15.2 Create project and load data

Let's say you want to create a new project and load raw data from a file. The test data used in our example is from the racing simulation game Live For Speed

(see section 14.1.4 for more information about Live for Speed), using the track Blackwood. The channels included in our test data are distance, time, speed, steering input, throttle pedal position, brake pedal position, engine RPM, gear, longitudinal g-force, lateral g-force and steering radius. Starting up the program will give you the main screen with a new project and a view. Select Project -> Import Data. This gives you the screen shown in figure 15.1. From here you browse to the data file you want to import and click Open. Another file browser will pop up. This is where you can select a calibration properties file. If you don't have one, click Cancel or just hit the Escape key on your keyboard.

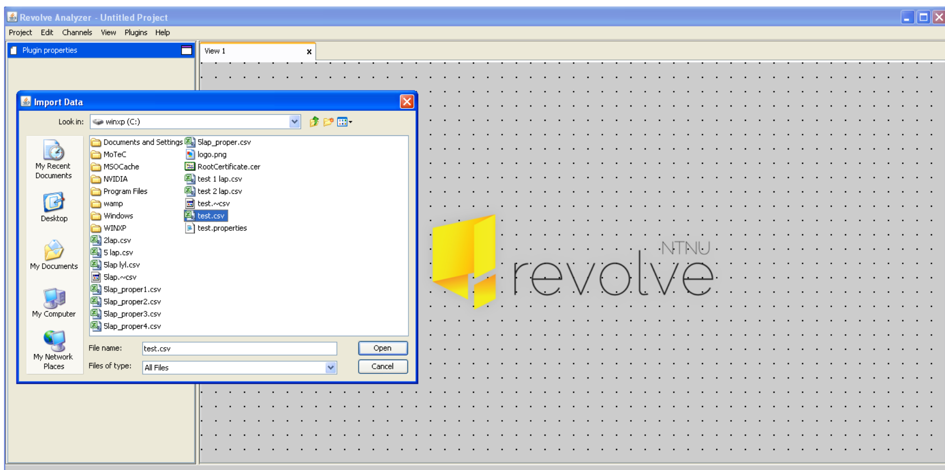

Figure 15.1: The screen shown when you click New Project.

15.3 Create a new view

When you create a new project, a view is automatically created. If you want to create another view, this is done by selecting View -> New View. A new view tab should be visible on your main screen now. If you'd like to rename it, double click the name of the view. To close it, click the X to the right of the view's name.

15.4 Calibrating channels

Some of the sensors will have to be calibrated in order to give data that makes sense or is in the proper unit. In order to calibrate channels, open the Channels menu and select Configure Channels. Select the channel you want to calibrate, then click Calibration. You will then be at the screen shown in figure 15.2. Click the "Add" button to add a polynomial to the formula used for calibrating the

channel. A preview of the formula is shown in the "Polynom" text field at the bottom. Clicking OK will calibrate the channel.

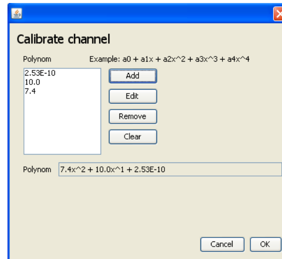

Figure 15.2: The "Channel Calibration" screen.

15.5 Analyzing data using graphs

We now come to the purpose of this software, to analyze the racing data. Let's use an example where you have been told by the driver that there are some under- and oversteering problems with the car. In order to get an overview of the car's handling we will be using the "Steering Input" and "Lateral G-Force" channel [3]. The first thing you do is click the "Plugins" menu at the top left of the screen, then select LineChart. Select the Channels tab on your left and check the boxes next to "LateralG" and "Steering". Go to the Segment tab and check Lap 1 and Segment 1. Your screen should then look like figure 15.3. If you want to change the colors of the lines in the graph, go to the Colors tab.

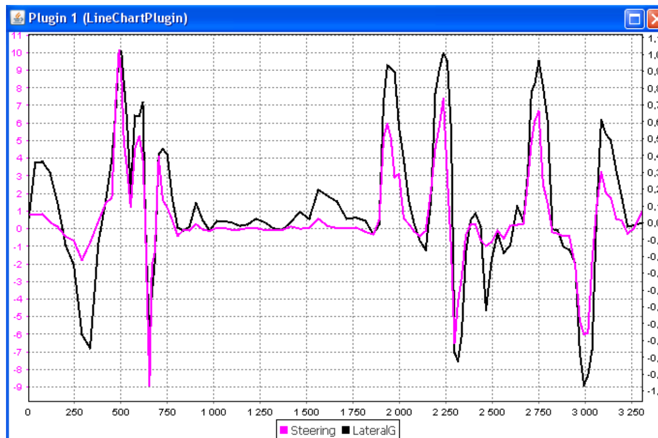

Figure 15.3: Steering Input, depicted in purple, and Lateral G-Force, depicted in black, graphs.

From the figure you can deduce that there is little oversteering, since there are no signs of counter-steering. It does, however, look like there is some understeering, since the steering in some turns is a lot higher, relative to the g-forces, than in other.

15.6 Generating a track map from the data

In order to generate a map, we add the plugin Map to the view. This is done through opening the Plugins menu, then selecting Map. You should then get the screen shown in figure 15.4.

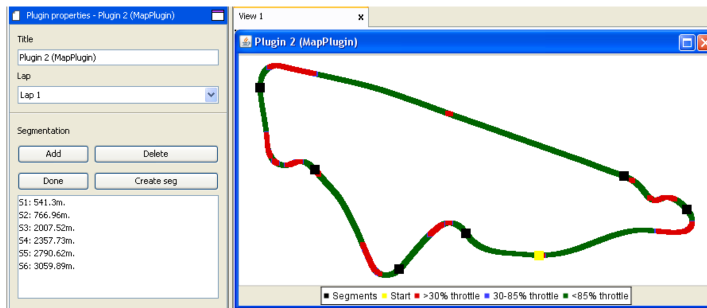

Figure 15.4: The map of the track Blackwood generated by the software.

The map can be generated in two ways: either from GPS coordinates or calculated based the lateral g-forces and the speed. Which way is used can be selected if you open the Channels menu and then click Configure Channels. Here you will find a checkbox which says "Use GPS for map". If GPS coordinates are not available, or you do not wish to use them, uncheck this box. In the map plugin you can also add and remove segments. Clicking Add on the left panel enables you to click on the map to add segments. Once you are happy with the segments made, click Create Segments.

15.7 Using video to analyze data

A neat feature in our software is the support for viewing a video alongside the rest of the plugins. To add the video plugin to your view, open the Plugins menu and click Video. If it's the first time you are starting the video plugin on your computer it will ask you for the install-directory of VLC. Once you have selected that, you will get a screen similar to the one figure 15.5 is depicting. Clicking "Open video.." will enable you to browse to the video you would like to use.

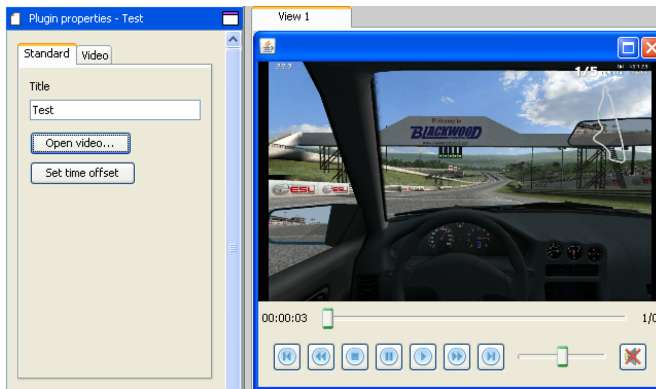

Figure 15.5: A screenshot of the video plugin.

Once this is done, you might want to Synchronize the video with the rest of the data collected, unless it already is. To check this, you can open the Plugins menu and select Playback. This is the plugin used to simulate the race from the data. Also open a Map Plugin and a LineChart plugin. Select a channel in LineChart that is very visible in the video recorded. For instance, if you can see the steering wheel in the video, selecting the steering channel might be a good idea, or the brake and throttle pressure if you can see the pedals. Once you have selected a channel, click on Start in the Playback plugin. All the plugins will start simulating the actual race. By looking at the video, the graph and the map, you should be able to identify certain "landmarks", for instance a sharp turn. By looking at the timer, you will see how far behind, or ahead, the video is of the data. Click "Set time offset" and input the amount of seconds. Negative values are supported.

Chapter 16

Plugin development guide

This chapter is meant to be used as a reference for developers that want to create a plugin for this software. For more detailed information, see the source code or JavaDoc.

16.1 The interface

All plugins are based on the DefaultPlugin interface. This interface has certain methods that will be called by the View. The attributes that every plugin has when created are as follows:

```
private View parentView;
private String title;
private int cursorPosition;
private boolean cursorLinked = true;
```

The parentView is the view that this plugin is shown in. The title is, obviously, the title of the plugin, shown in the top bar of the window. The cursorposition and linked parameter are used by the view to synchronize eventual cursors in the plugin displays. When e.g. playback is started, the playback plugin sends synchronization requests all plugins that has the cursorLinked parameter set (default setting). This will be further explained under the method setCursorPosition(int pos). All plugins also override methods in the Externalizable interface, which is used for saving and loading their state.

16.1.1 Required methods

This is a list of the methods that have to be overridden in a plugin.

init(View v) This is the method that initializes the plugin and it's the first method that is run when a new plugin is opened. Everything that the plugin needs to startup needs to be in this method. This method throws a PluginInitException. This is useful if you are dependent on certain channels being present, for instance the Map plugin is dependent on either GPS information or the Lateral Acceleration and Speed channels being available. This way, the initialization can be configured to abort if necessary. If a plugin is loaded from a project or view file, the stored parameters will be set in the new plugin instance before the init method is invoked. This means that we must check if these parameters already have been initialized before creating new fields with default values.

getPluginPanel() getPluginPanel() returns an object of the type java.awt.Component. This can be for instance a JPanel, or any other object that extends Component indirectly. The pluginPanel is the main graphical outlet for a plugin. For example in the LineChartPlugin, the pluginPanel is a JFree-chart where the graph is drawn. In Report, the pluginPanel is a JPanel which contains the ScrollPane and the JTable itself.

These are the only methods you absolutely have to implement in order to create a plugin.

16.1.2 Other methods

Here we will discuss the other methods that are in the DefaultPlugin interface. While many of these are not required in order to run and show a basic plugin, there are some methods here which every plugin should have.

getPropertiesPanel() This method is used to retrieve the properties-panel, in other words the panel shown on the left-hand side when your plugin is selected in a view. This method returns a Component, just like getPluginPanel(). In the plugins we've made so far this propertiesPanel has been used mostly for configuration of the plugin.

writeExternal(ObjectOutput out) To save a plugin's state, you use this method. If there are any variables you would like to save, you have to include them in this method. An example from LineChartPlugin shows how it is used.

```
out.writeInt(this.dataVersion);
out.writeObject(this.getSelectedChannels());
out.writeObject(this.getSelectedLaps());
out.writeObject(this.getSelectedSegments());
out.writeBoolean(lapsSuperImposed);
out.writeBoolean(followDomainCursor);
```

readExternal(ObjectInput in) This method works largely the same way that the writeExternal method does, except you have to cast objects to their

correct class. It is also important that you load the objects in the same order that you saved them in See this example from the LineChartPlugin.

```
this.selectedChannels = (ArrayList<String>) in.readObject();
this.selectedLaps = (ArrayList<Integer>) in.readObject();
this.selectedSegments = (ArrayList<Integer>) in.readObject();
this.lapsSuperImposed = in.readBoolean();
this.followDomainCursor = in.readBoolean();
```

getParentView() Used in order to get the View-object that this plugin is shown in. From the View-object you can get the Project-object, which is where all the data in the current project is stored. The default implementation of this method works fine for all plugins and you do not need to override it.

setCursorPosition(int position) This method is as mentioned used to set the cursorposition of the plugin. In other words if your plugin is a part of simulation playback, this method has to change your plugin's state. An example is the LineChartPlugin, where setCursorPosition moves a vertical line over the graph to show progress in the playback.

startPlayback() This is used if your plugin needs to change it's state when plugin starts.

stopPlayback() The same as startPlayback(), except for stopping.

segmentationChanged() If the segmentation of the active project changes, this method is evoked in all plugins. Used for updating the plugin with the new segmentation.

channelsChanged() The same as segmentationChanged(), but for channels.

colorsChanged() This method will be executed if the color for a certain channel changes.

16.2 Deployment

Put the compiled .class-files in a file-structure which mimics the package-structure (plugin.testplugin would be plugin/testplugin/). Make a .jar-file from this and put it in the same folder as RevolveAnalyze.jar. The filename should be plugin.-packagename-.-classname-.jar.

For more information, see the Java Docs for the system and the source code for the plugins that have already been made.

Part IV

EVALUATION AND CONCLUSIONS

Chapter 17

User evaluation

Here we will gather the evaluations we've received from the users of the program.

17.1 First impressions usertest

The first user evaluation was held at Revolve's offices in November 2011, while we were still doing the In-Depth Project. We arranged a meeting with the Revolve group, in other words with the actual users this software is meant for. We hoped we could get some feedback on the program, especially the user interface. The meeting lasted for about 90 minutes, during which we presented the prototype, and then the users got to try it a little.

17.1.1 Participants

The following people were present during the meeting:

Erling Kjelstrup, Project Manager (PM)
Ragnar R. Homb, Data Acquisition Systems (DA)
Mads Aasvik, Research & Development (RD)
Evelyn Livermore, Group Leader of Electronics and Engine Management (EE)
Per Øyvind Stadheim, Software Developer (SD1)
Lauritz Møllersen, Software Developer (SD2)

We invited everyone to join the meeting, and let the Revolve group send the people they felt was the most important users of the program. We feel we got a diverse group to get feedback from.

17.1.2 Feedback

This is some of the feedback we got during the meeting. The participants will be referred to as for instance PM for Project Manager Erling Kjelstrup.

Positives

The most important positive was that the users liked the general framework for the software. This is what we've spent most of the time developing. DA said that the graphical interface was "very intuitive", and liked how it was possible to double-click a graph to maximize it and study only that graph. EE was happy with the performance. The map generation from g-force and speed was considered a cool feature, although maybe not the most crucial one since GPS coordinates will be used to draw a map most of the time.

Negatives

There were some questions around missing tools, which is to be expected since this is not a finished product. The lack of a global marker was commented on by RD and DA, there is also (yet) no global zoom for all graphs. Math functions are present, but are very basic. DA said this was of no concern, since they only needed simple math support. Also not implemented yet was proper GUI persistence[1], but this is on the agenda. There is also a bug with the auto-zooming when the user adds new channels with very different absolute values. If the user has RPM and g-force on the same graph, the highest y-value shown on the graph might be 10000 RPM, which makes the g-forces very hard to see (since it's usually never above 2G). This needs to be fixed.

It may seem meaningless to discuss missing functionality in a piece of unfinished software, but this illustrates what is most important to the customer.

New features

During the meeting there were suggested some new features. DA suggested a "notepad plugin", somewhere the user could jot down some lines about the project at hand, which would be saved when the project is saved. This would be a helpful feature when there are several collocated persons analysing the same data. Another suggested feature is a legend on the graphs and maps, so it is possible to see what color is what channel or sector, respectively. We have also talked about video playback of the lap from the driver's point of view and synchronizing this to the sample data, but this is not a big priority.

[1]The ability to remember how the different visualizations are sized and positioned within the layout.

17.1.3 Conclusion

In general, the users were positive about the progress so far. There were some missing features, but we were aware of this. The suggestions for new features were good, and something to definitely consider adding in any further work.

17.2 SUS user test

The next user test we executed was the System Usability Scale test [14]. SUS means System Usability Scale. It is a simple, ten-item Likert scale [57]. A statement is presented and it's rated on a scale from 1 to 5, based on how much or how little the participant agrees with the statement. Ten statements are made [14].

"To calculate the SUS score, first sum the score contributions from each item. Each item's score contribution will range from 0 to 4. For items 1, 3, 5, 7,and 9 the score contribution is the scale position minus 1. For items 2, 4, 6, 8 and 10, the contribution is 5 minus the scale position. Multiply the sum of the scores by 2.5 to obtain the overall value of SUS. SUS scores have a range of 0 to 100" [14]. 0 means very low usability, 100 means very high usability.

The test was distributed via e-mail to everyone in the Revolve Group. They received a copy of the runnable software, some test data, the SUS-survey to fill in and some quick instructions. In addition to the 10 questions posed by SUS, we also included a comment field where the user could be more specific as to what was good, bad or simply missing from the software in general.

17.2.1 Alternatives to SUS

Instead of using SUS, we also considered doing workshop evaluations of the usability, but this is harder to quantify as well as find participants for. Another alternative we considered is called The Usability Metric for User Experience, or UMEX. This is basically a version of SUS where only the highest correlating questions from SUS are used [58]. UMEX is a four-item Likert Scale, with 7 points for every item.

17.2.2 SUS results

The SUS score for our program was 70 from a total of 8 user evaluations. According to a study of about 5000 users across 500 different system evaluations had an average SUS score of 68 [59]. Another study had an average of 66 [60]. According to these studies our usability is above average. Table 17.1 shows the SUS score contribution for each question (higher is better, even for the negative questions).

Table 17.1: The SUS score contributions.

Question	Contribution
1. I think that I would like to use this system frequently	3.375
2. I found the system unnecessarily complex	2.875
3.I thought the system was easy to use	2.625
4. I think that I would need the support of a technical person to be able to use this system	2.750
5. I found the various functions in this system were well integrated	3.000
6. I thought there was too much inconsistency in this system	3.125
7. I would imagine that most people would learn to use this system very quickly	2.500
8. I found the system very cumbersome to use	2.750
9. I felt very confident using the system	2.000
10. I needed to learn a lot of things before I could get going with this system	2.875

Analyzing the data, we get very positive ratings on the first and sixth questions, which are “I would like to use this system frequently” and “there was too much inconsistency in this system”, respectively. We lose some points on questions 3, 7 and 9, which are “the system was easy to use”, “most people would learn this system very quickly” and “I felt very confident using the system”. These are trends that should be paid attention to in any later development of the system.

17.2.3 Feedback

In addition to the SUS-score, there was a possibility of sending in additional comments. Here we will discuss some of these.

Positives

A lot of positive feedback. User 1 says “LineChart is very easy-to-use and intuitive (...) So many ways of customizing it!”. The simpleness of the system is mentioned as one of the big positives, there’s not a lot of “bloated” features. User 2 writes “The program looks very good and I’m looking forward to testing it with real-life data from our dataloggers!”. User 3 had this to say about the program: “Well done guys, this looks very promising!”.

Negatives

Not all of the feedback was positive. User 3 did not get the video plugin to work, this could be a human error or there could be some compatibility issues. User 2 experienced problems with the LineChart plugin where he could not change the channels shown. User 1 wished there where default selections of laps and segments in place, for instance select lap 1 and segment 1 by default when the user opens a plugin. He also felt the LineChart plugin was missing a readout for the cursor, to see what value the cursor was at. One of the quotes from User 1's feedback is "Map: how you make sectors does not feel very intuitive".

17.2.4 Conclusion

From the feedback and the SUS score and the textual feedback we can conclude that there are still a couple of improvements there could be done on the software, especially when it comes to the overall quality-feel.

Chapter 18

Self evaluation

This chapter consists of an evaluation of the different aspects of our own planning and work, and how we feel the project went in general. We end with some thoughts on what we learned from this project.

18.1 Planning

In any development project limited by time, scheduling is important. This is primarily to ensure that the project finishes within its set timeframe. Because of the fact that this project is quite large compared to other projects we have had, the scheduling becomes an even more important aspect. Overall, we feel that the planning part in our project was not prioritized to the extent it should have been. We did do some initial task scheduling in the startup phase, but we might have been a bit too eager to get to the architecture and implementation phases, instead of carefully making a more detailed schedule, with more specific milestones and deadlines. In hindsight, the project could have benefited from this. Also the schedule should have been more carefully followed up throughout the project.

18.2 Research

The research in this project was practically aligned and very interesting. There is not an abundance of research done or literature written on the subject of racing data analysis software, but through creating our own software and doing the evaluation test we got valuable knowledge within the field.

In general we could have spent more time searching for literature by sacrificing a little of the time we spent reading it. Having a wider range of research papers and

books to consult could have been helpful, but as mentioned, it's not easy to come by.

18.3 Development

Starting development is always exciting. From the start, we had the loose workload distribution that Per Øyvind would focus mainly on the GUI and architecture aspects, and Lauritz would focus on the lower level aspects of the software. There were of course a lot of overlap, but we feel that having some sort of main focus areas for each developer helped productivity, since we could get a very good overview of the parts of the system we were working on.

In general, we feel that the development phase went well, even though at times there were some major obstacles to overcome. We both learned a lot as software developers and we are proud of the software we have produced. Hopefully Revolve will put it to good use.

18.4 Report

We were determined to add to the project report during all phases of the project, as a natural part of the startup, design and development. This was done to some extent, but in hindsight not enough. It should have been a bigger focus throughout the project. In the end though, we are very happy with how the report turned out.

18.5 Individual thoughts

Here we also include some individual thoughts on how the project went and what we learned.

Lauritz "I learned about performance; how to optimize a system to give the best performance. In the previous project's I have participated in, performance has never really been a key issue. In this project however, there was a considerable amount of time spent trying to get the best possible performance. This involved testing, finding the problem area and finding a solution to this problem. I felt that this was a very interesting field."

Per Øyvind "I learned a lot about architecture, and faced some challenges when designing the plugin architecture. It was interesting to learn about reflection support in programming languages and using this technology to design a modular system that can detect new software modules automatically. I also enjoyed learning more about GUI programming."

Chapter 19

Conclusion

During this project we have gained a thorough understanding of the world of racing data analysis through literature study of the problem domain. We have also researched ways to create a highly modifiable and modular architecture with support for plugins. Equipped with this knowledge we have designed an architecture focusing on modifiability, as well as implemented software based on this architecture with high usability. This usability has been tested, and proven higher than average, through use of SUS.

The goal of this project was to continue working on the basic framework we had created during the In-Depth Project and turn this into a working RDAS. The software aimed to be powerful and user-friendly enough to enable Revolve to use it when they are competing in Formula Student. In this regard, we have reached our main goal. The software is up and running, and the feedback from both our project supervisor Alf Inge Wang and the Revolve Group have been good. In the Video Plugin, our software even has functionality not offered by the more advanced and expensive software from professional manufacturers.

The sidegoals were also reached, especially when it comes to insight to the racing world. We learned a lot about racing and cars. How different engines work, racing techniques, ways to setup the car to go faster, and how to identify and solve problems in the car setup.

Chapter 20

Further work

In this chapter we will list some of the possible future work that can be done within this project.

20.1 GUI

The GUI's look could be more aesthetically pleasing. For instance right now it's using the operating system's look and feel, which is highly functional but not very beautiful.

20.2 Plugins

In this section we will go through some of the work that could be done in the future on the different plugins we have created.

20.2.1 Linechart and XYChart

One thing that could be added in LineChart and XYChart is the possibility of saving default configurations of the graph window. For instance, one configuration called "Steering" that shows steering input and lateral g-force.

20.2.2 Map

Map should have support for changing the "overlay" of the drawn map. As it stands today, parts of the map will be drawn in different colors depending on if

the driver is braking heavily, accelerating heavily or inbetween. Other ways to draw the map could be segments getting different colors. The map could have parts of it colored depending on what kind of steering input is being applied, the speed the car is driving at or based on any other of the channels' values.

The way segments are added could also be improved, and the plugin could in general be more user-friendly.

20.2.3 Report

One thing that could be improved with Report is to add the possibility of showing all channels for a certain lap or segment. The functionality right now only allows the user to look at all laps for one channel. Having the ability to export a PDF from the Report for instance would be nice. Right now the user can export data from Report, but it's through copy/pasting into for instance Excel.

20.2.4 Video

In the Video Plugin, there should be a better way to synchronize the video with the data. For instance, it could be linked to the LineChart plugin and the program could "find" the same point in the LineChart and video using the global cursor, then the user could click a Synchronize-button. Automatic support for synchronizing could also be added, through for instance a special frame in the video that the program looked for.

Another cool feature would be the possibility of showing the video of two different laps at the same time, the same way LineChart can show the graph of two laps on top of each other.

20.2.5 Notepad

This plugin is very basic. There is no left-hand option panel, which all the other plugins have. In an extended version, the plugin could have functionality to change the font, font-size and other parameters.

20.2.6 Playback

Playback could have it's own timeline, or scrollbar, so that the user can scroll through all the data and control everything from the Playback plugin.

20.3 Project platform

20.3.1 Math channels

Math channels are implemented, and a GUI has been made to create and edit math channels. However, this has not been a big priority for Revolve, and as such the functionality is pretty basic. Math channels should have their own language, where the user can write formulas, for instance $channel(speed)/1000$ or even use conditional if-sentences.

20.3.2 Input data restrictions

At the moment there are some restrictions to the input data, as mention in section 15.1. Some of these restrictions could be removed through having better processing of the data.

There is not added any filtering of data to the software. This will, at the moment, have to be done outside of the software if needed, by for instance MatLab. A low-pass filter might be useful for the real-life data, sadly we only had the chance to try the software with simulator data. The reason for this is that we did not get access to real-life data until before the delivery data of the project. This means there is a lot of testing to be done with data that isn't simulated.

20.4 General

There are plenty of further work to be done when it comes to adding new plugins. One feature that would be nice is an own plugin for suspension analysis. This is a field of it's own, and involves a lot of mathematics. Through the Fast Fourier Transform algortihm the user can analyze the suspensions frequency with a plugin and then show these results.

Another plugin that would be cool to see is one for extended simulation, for instance through showing gadgets like the speedometer.

Bibliography

[1] Revolve ntnu. http://www.revolve.no. Last accessed 26.03.2011.

[2] Formula student uk. http://www.formulastudent.com/. Last accessed 25.05.2012.

[3] Templeman, G. *The Competition Car Data Logging Manual;* Veloce Publishing, 2008.

[4] Telemetry and data analysis introduction. http://scarbsf1.wordpress.com /2011/08/18/telemetry-and-data-analysis introduction/. Last accessed 25.05.2012.

[5] Royce, W. *ICSE '87 Proceedings of the 9th international conference on Software Engineering* **1987**, pages 1–9.

[6] Modified waterfall modell. http://zone.ni.com/images/reference/en XX/help/371361D-01/loc_eps_water_lifecycle.gif. Last accessed 26.03.2011.

[7] Rising, L.; Janoff, N. *IEEE Software* **2000**, *17 Issue 4*, 26–32.

[8] Fried, L. *Software Engineering Tools, Techniques, Practice 2*, 15–25.

[9] http://iosconsult.com/assets/images/scrum_diagram.jpg. Last accessed 22.05.2012.

[10] Kajko-Mattsson, M.; et.al. *Software Maintenance ICSM '06. 22nd IEEE International Conference* **2006**, pages 422–425.

[11] Google docs. http://docs.google.com. Last accessed 25.05.2012.

[12] Dawson, C. *Practical Research Methods A user-friendly guide to mastering research techniques and projects;* New Delhi, UBS Publishers Distributors, 2002.

[13] Wang, A. I. Using a mobile, agent-based environment to support cooperative software processes Master's thesis, NTNU, **2001**.

[14] Brooke, J. *Brooke* **1996**.

[15] Formula 1 the offical f1 website. http://www.formula1.com/inside_f1/glossary.html. Last accessed 25.05.2012.

[16] Data acquisition - a valuable tool for racers - stock racing car magazine. http://www.stockcarracing.com/techarticles/scrp_0703_data_acquisition /viewall.html. Last accessed 26.03.2011.

[17] Products - data loggers - dl1 classic. http://www.race technology.com/dl1_2_27.html. Last accessed 26.03.2011.

[18] aim-tec.co.uk logger. http://aim tec.co.uk/blog/?paged=2. Last accessed 25.05.2012.

[19] Motec. http://www.motec.com. Last accessed 25.05.2012.

[20] Aim sports | mychron. http://www.aimsports.com/. Last accessed 01.06.2012.

[21] Software - main analysis software. http://www.race technology.com/main_analysis _software_2_37.html. Last accessed 26.03.2011.

[22] Maier, J. R. A.; Fadel, G. M. **2006**.

[23] Gibson, J. J. *Perceiving, Acting, and Knowing* **1977**.

[24] Gibson, J. J. *The Ecological Approach to Visual Perception;* Psychology Press, 1979.

[25] Len Bass, P. C.; Kazman, R. *Software Architecture in Practice;* Addison-Wesley Professional, 2003.

[26] Kruchten, P. *IEEE Software* **1995**, *12 Issue 6*, 42–50.

[27] Usability. http://en.wikipedia.org/wiki/Usability. Last accessed 22.05.2012.

[28] Iso/tr 16982 ergonomics of human-system interaction. http://www.iso.org/iso/catalogue_detail?csnumber=31176. Last accessed 02.02.2012.

[29] Modular programming. http://en.wikipedia.org/wiki/Modularity_29.05.2012.

[30] Cross-platform. http://en.wikipedia.org/wiki/Cross platform. Last accessed 25.05.2012.

[31] Multiple document interface. http://en.wikipedia.org/wiki/Multiple_document _interface. Last accessed 22.05.2012.

[32] Java plugin framework (jpf). http://jpf.sourceforge.net/. Last accessed 25.05.2012.

[33] Osgi alliance main. http://www.osgi.org/Main/HomePage. Last accessed 26.03.2011.

[34] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software;* Addison-Wesley, 1994.

[35] Java se 6 performance white paper. http://java.sun.com/performance/reference/whitepapers/6_performance.html. Last accessed 25.05.2012.

[36] Ali-Reza Adl-Tabatabai, Guei-Yuan Lueh, V. M. P. J. M. S. *PLDI '98 Proceedings of the ACM SIGPLAN 1998 conference on Programming language design and implementation* **1998**, *1*, 1.

[37] Nine language performance round-up: Benchmarking math & file i/o. http://www.osnews.com/story/5602. Last accessed 26.03.2011.

[38] Mozart documentation. http://www.mozart oz.org/documentation/index.html. Last accessed 20.03.2011.

[39] Perry, W. *Effective methods for software testing;* John Wiley & Sons, Inc. New York, NY, USA, 2006.

[40] Ostrand, T. *Encyclopedia of Software Engineering* **2002**.

[41] Lou, L. *Technology Maturation and Research Strategies* **2000**.

[42] The open source definition. http://www.opensource.org/docs/osd. Last accessed 20.03.2011.

[43] Free software foundation. http://www.fsf.org/. Last accessed 25.05.2012.

[44] von Hippel, E. *MIT Sloan management review* **2001**.

[45] Sourceforge. http://sourceforge.net/. Last accessed 25.05.2012.

[46] Gnu general public license. http://www.opensource.org/licenses/gpl 2.0.php. Last accessed 20.03.2011.

[47] Live for speed - online racing simulator. http://www.lfs.net/. Last accessed 25.05.2012.

[48] Fraps real-time video capture and benchmarking. http://www.fraps.com/. Last accessed 25.05.2012.

[49] Java serialization api. http://java.sun.com/developer/technicalArticles/Programming/serialization/.

[50] Trail: The reflection api (java api tutorials). http:/docs.oracle.com/javase/tutorial/reflect/. Last accessed 26.03.2011.

[51] Docking frames. http://dock.javaforge.com/. Last accessed 02.05.2012.

[52] Mydoggy - my java docking framework. http://mydoggy.sourceforge.net/. Last accessed 22.05.2012.

[53] Gnu lesser general public license. http://www.gnu.org/licenses/lgpl.html. Last accessed 25.05.2012.

[54] Jfreechart. http://www.jfree.org/jfreechart/. Last accessed 25.05.2012.

[55] Videolan official page for the vlc media player. http://www.videolan.org/vlc/. Last accessed 26.03.2011.

[56] vlcj java framework for the vlc media player. http://code.google.com/p/vlcj/. Last accessed 13.05.2012.

[57] Carifio, J.; Perla, R. *Journal of Social Sciences 3* **2007**, *3*, 106–116.

[58] Finstad, K. *Interacting with Computers* **2010**, *22*, 323–327.

[59] Measuring usability with the system usability scale. http://www.measuringusability.com/sus.php. Last accessed 24.05.2012.

[60] Tullis, T.; Albert, B. *Measuring the User Experience;* Morgan Kaupman, 2008.

Appendix A

Data channels

A complete list of sensors intended to be used on the Revolve race car.

- Water temperature
- Intake temperature
- MAP
- Oxygen %
- TPS (Throttle Position Sensor)
- Oil pressure
- Fuel pressure
- Front wheel speed
- Banking
- EGT (Exhaust Gas Temperature)
- Gear
- Brake pressure front
- Brake pressure back
- Lateral g-force
- Steering wheel position
- Brake pedal position
- Brake caliper temperature
- Tire temperature

- Tire air pressure
- Suspension travel
- Clutch pedal position
- Polar receiver
- GPS position
- GPS speed