bloxp

VISUALRACING

Blog compiled using Bloxp

CONTENTS

Semester II: Week 6 – Patterns

Post #16

Semester II: Week 7 – Metrics

Post #17

Semester II: Week 8 – Installation

Post #18

Final Summary

#1

Week 1 – Our Vision

The goal of our project is to create a tool to enhance your sim racing experience and improve your capabilities behind the steering wheel of your racing simulator.

The tool will offer you the capabilities of visualizing various information from your connected racing simulation. A customizable interface will let you adjust the way the information is displayed. We plan on using simple key figures, different kinds of graphs and more advanced visualization methods. Furthermore the tool will allow you to analyze your driving behavior to help you improve your skills and lower your lap-times.

Lap times and information about velocity are some examples of what we want to display to the drivers. In addition to that we have access to much more detailed data, like the gear the driver is in, the engines RPM, engine and tire temperatures, tire wear and much more. Furthermore we can display information from the drivers input, like the position of the pedals. This allows the driver to see where on the track he could start accelerating a bit earlier or where he can improve the work with the clutch to improve the shift-times.

Our Project is divided into the following subtasks:

- visualization/Graphical User Interface
  - We have many ideas on how we want to display data, like 3D-elements, charts and graphs.
- reading the data stream from different racing simulations and transforming it into an unified model
  - There are many simulations out there, that allow us to read lots of data to visualize, we will start by transferring the data outputted by simulations like Assetto Corsa or RaceRoom Racing Experience into our unified data structure.
- recording and saving data as well as analyzing the long term development of

your driving behavior

The project is based on C++ and the Qt-Framework. The application will only run on Windows, because all common racing simulations only run on windows as well. You can find our repository on GitHub.

Permalink: https://visualracing.wordpress.com/2017/10/05/week1/

#2

Week 2 – Team & Technology

Last week we shared our vision for our project with you and today we'll let you know who we are and how we are going to be organized.

> "Get the right people. Then no matter what all else you might do wrong after that, the people will save you. That's what management is all about."
> ~ Tom DeMarco

Team Members

- Lars Hübner
  - *Tool Specialist*
  - *Designer*
  - *Backend- & Frontend-Developer**
- Christopher Klammt
  - *Frontend-Developer**
  - *Test-Designer / Tester*
- Felix Starke
  - *Project Manager*
  - *Backend-Developer (data management, analytics)**

**Implementer*

Roles

Project Manager The project manager has an overview of all current tasks and potential risks. However, he isn't the sole decision maker as all decisions will be discussed with all team members and will be made as a team.

Tool Specialist The tool specialist is responsible for choosing and configuring the tools that will be used throughout the lifecycle of our project. He is the main point of contact for all questions regarding the tools we use and the difficulties that come with them.

Designer His task is to have a good overview of the whole software architecture. He should communicate his plans and ideas with the whole team and make sure, that the finished software has a consistent look and feel.

Implementer All team members of our project will take part in the development process of our software. As described in the list above, each implementer has his own area of responsibility. However, all team members have a rough overview of the tasks and challenges of their colleagues, therefore working on subtasks connecting multiple areas can be done easily.

Test-Designer / Tester The test-designers task is to design and develop tests which ensure that the code quality and the overall stability of the software is granted and meets the highest possible standards. Additionally he uses our continuous integration tool to make sure all tests will be executed according to plan. He also overviews the test-results and communicates those within the team.

We based our role definitions on the roles as defined by the *IBM Rational Unified Process*, but it is important to note that we will not hesitate to swap these if needed. This can occur if we realize that the workload of a role is bigger than assumed so that we have to rethink our role allocation.

Technologies

Language

We will be using **C++** as our main programming language as part of our team has already gained some experience and we are very interested in diving deeper into C++.

Additionally, accessing the Shared Memory Block to read the data is very easy, due to the fact the C++ does not run in a virtual environment, unlike e.g. Java.

Framework

As to the framework we will use the **Qt Framework**.

> "Qt is a cross platform development framework written in C++."
> ~ www.qt.io

In the beginning the Framework was designed for user interfaces and provides in the latest version many other modules (e.g. Databases, XML, OpenGL, etc). Every module has a common scheme and is build from the same API design idea. The Framework is completely written in plain C++ and extends the the programming language C++ perfectly. Additionally Qt provides cross platform development for Windows, Linux and macOS built from just one source. It is also known and commonly used in embedded development.

IDE

As we are using the Qt Framework and it's User Interface technologies, we are using the powerful **Qt Creator** as our IDE, as it is optimized to work with the Qt Framework and offers a good UI-Designer.

Furthermore we will be using **QtTest** and **Google Test** for the testing of our software and GitHub as a version management system. To keep track of our tasks we will be using **YouTrack**.

We will be using **AppVeyor** for Continuous Integration.

Permalink: https://visualracing.wordpress.com/2017/10/16/week2/

#3

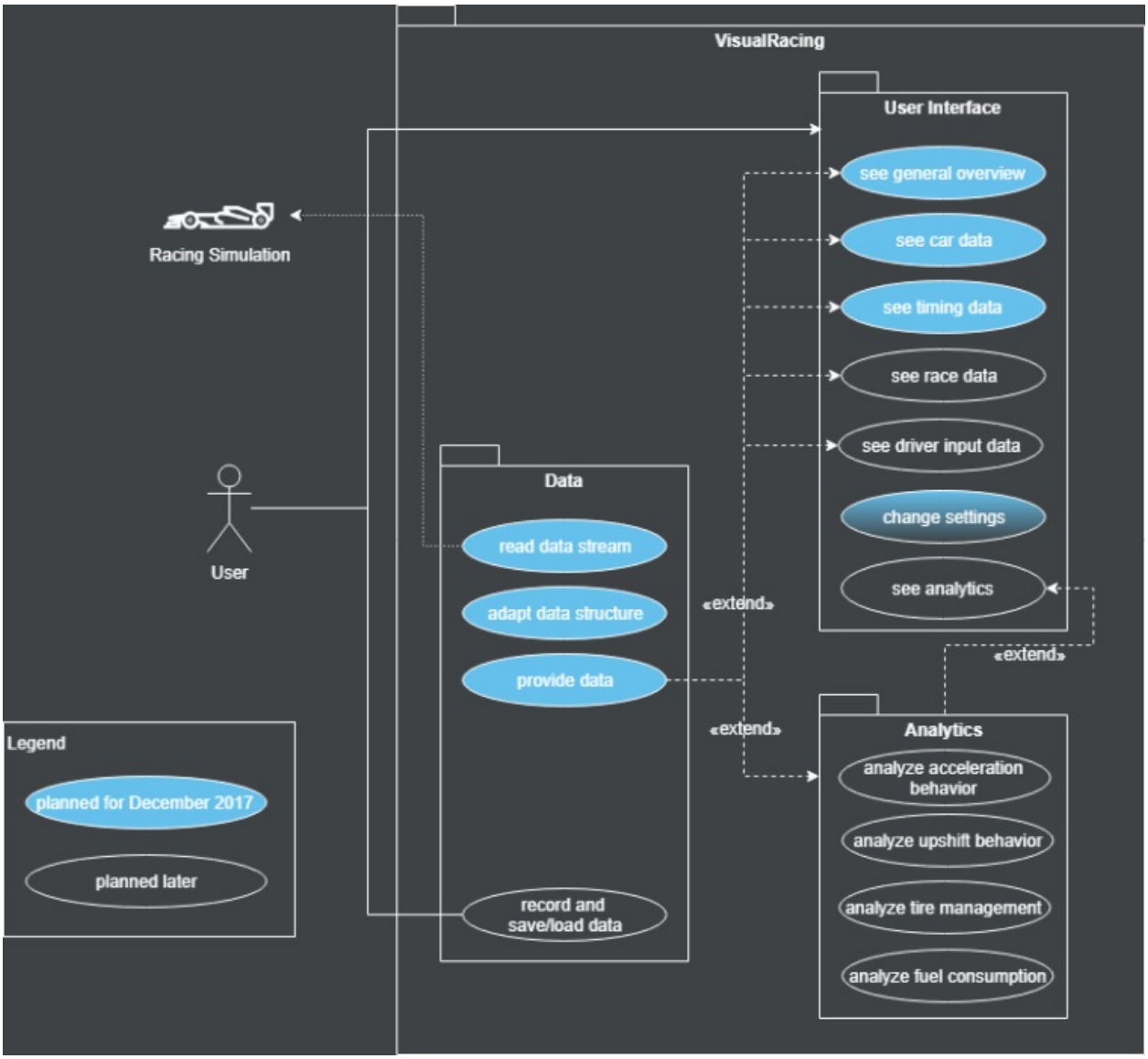
Week 3 – Software Requirements Specification

During the last week we worked on our Software Requirements Specification (SRS) Document and created an Use Case Diagram (UCD).

You can find the SRS here and the UCD below.

Please note that the SRS will be updated during the projects development process.

EDIT: As part of rescoping our project at the start of the second semester, we have updated our UCD. You can see the changes on GitHub and this blog post.



Permalink: https://visualracing.wordpress.com/2017/10/21/week3/

#4

Week 4 – Use Case Specification

Hi visualracing team,

great to hear from you again!

Your two use cases you chose are very detailed and well structured. Furthermore, your activity diagrams are comprehensible and easy to follow. Moreover, your mock-ups help to imagine the future design of your racing application.

What came into my mind is that you could add a revision history to your use case specification files to stick more closely to the template but I think it should be fine without it, too.

Cheers,

Pascal – chessmate team

Like Like

Permalink: https://visualracing.wordpress.com/2017/10/30/week4/

#5

Week 5 – Scrum

In the following two weeks our first Sprint will take place and our goal is to create a first prototype and write the narratives (.feature files) for the two use cases we already started documenting.

You can have a look at our YouTrack Scrum board here.

If you want to take a look at the burndown chart for our first sprint you can have a look here.

The tasks have an estimation for how long it will take to resolve them and an attribute for the actual amount of time spent on completing the task. This information is used for creating the burndown charts for each sprint.

Additionally you can have a look at this time report to get an overview on who worked how much (grouped by RUP workflow).

Permalink: https://visualracing.wordpress.com/2017/11/06/week5/

#6

Week 6 – Gherkin .feature files

Hi everybody,

during the last week we worked on creating .feature files for our application.

You are welcome to have a look at the .feature files:

As our Use-Case Read Data Stream is only part of our back-end, we could not create a .feature file for it. Therefore we have chosen a different Use-Case for this weeks task. Feel free to take a look at the new Use Case Specification we added for the Use-Case **Display Car Data**.

Additionally, we linked our .feature files in the corresponding Use Case Specifications and in the SRS:

To show you that syntax-highlighting works for our .feature files, we have added screenshots from within our IDE below.

Change Settings

```
Feature: Change Settings
    In order to change the settings to my preferences
    As a user
    I want to see the tab to change the Settings and change all settings accordingly

    Scenario: language changed from English to Deutsch
        Given I have selected language English
        When I select language Deutsch
        Then language Deutsch should be applied

    Scenario: theme changed form Dark to Light
        Given I have selected theme Dark
        When I select theme Light
        Then theme Light should be applied

    Scenario: unit changed from Metric to Imperial
        Given I have selected unit system Metric
        When I select unit system Imperial
        Then I unit system Imperial should be applied

    Scenario: language changed from Deutsch to English
        Given I have selected language Deutsch
        When I select language English
        Then language English should be applied

    Scenario: theme changed from Light to Dark
        Given I have selected theme Light
        When I select theme Dark
        Then theme Dark should be applied

    Scenario: unit changed from Imperial to Metric
        Given I have selected unit system Imperial
        When I select unit system Metric
        Then I unit system Metric should be applied
```

Display Car Data

```
Feature: Display car data
    In order to get detailed information about the car
    As a sim-racer
    I want to see the tab containing the car data

    Scenario: Switch to Car-Tab
        Given a racing simulation is running
        And the application is not currently displaying the Car-Tab
        When I select the Car-Tab
        Then I should no longer see what was displayed previously
        And I should see the Car-Tab containing information like tire temperatures

    Scenario: Display Car-Tab
        Given a racing simulation is running
        And the application is currently displaying the Car-Tab
        When I select the Car-Tab
        Then I should see the Car-Tab containing information like tire temperatures
```

For your information:

Sadly we didn't get the feature file testing to run with Qt, QML and C++, so we developed a little test to demonstrate how the testing with cucumber is supposed to work.

In this demo test case google will be opened, searched and tested whether the title of the tab is correct.

feature file

```
Feature: Google can search

Background:
    Given I am on "http://www.google.com"

Scenario: Search for a term
    When I fill in "q" found by "name" with "VisualRacing"
    And I submit
    Then I should see title "VisualRacing - Google-Suche"
```

step definitions

And finally you can see a short clip of the testing in action:

```
Given ("I am on {string}") do |url|
    @browser.navigate.to url
    sleep 1
end

When ("I fill in {string} found by {string} with {string}") do |value, type, keys|
    @element = @browser.find_element(type, value)
    @element.send_keys keys
    sleep 1
end

When ("I submit") do
    @element.submit
    sleep 1
end

Then ("I should see title {string}") do |title|
    sleep 1
    raise "Fail" if @browser.title != title
end
```

Permalink: https://visualracing.wordpress.com/2017/11/13/week6/

#7

Week 7 – Class Diagram

Hey guys,

last week we were busy working on the class diagrams for our project.

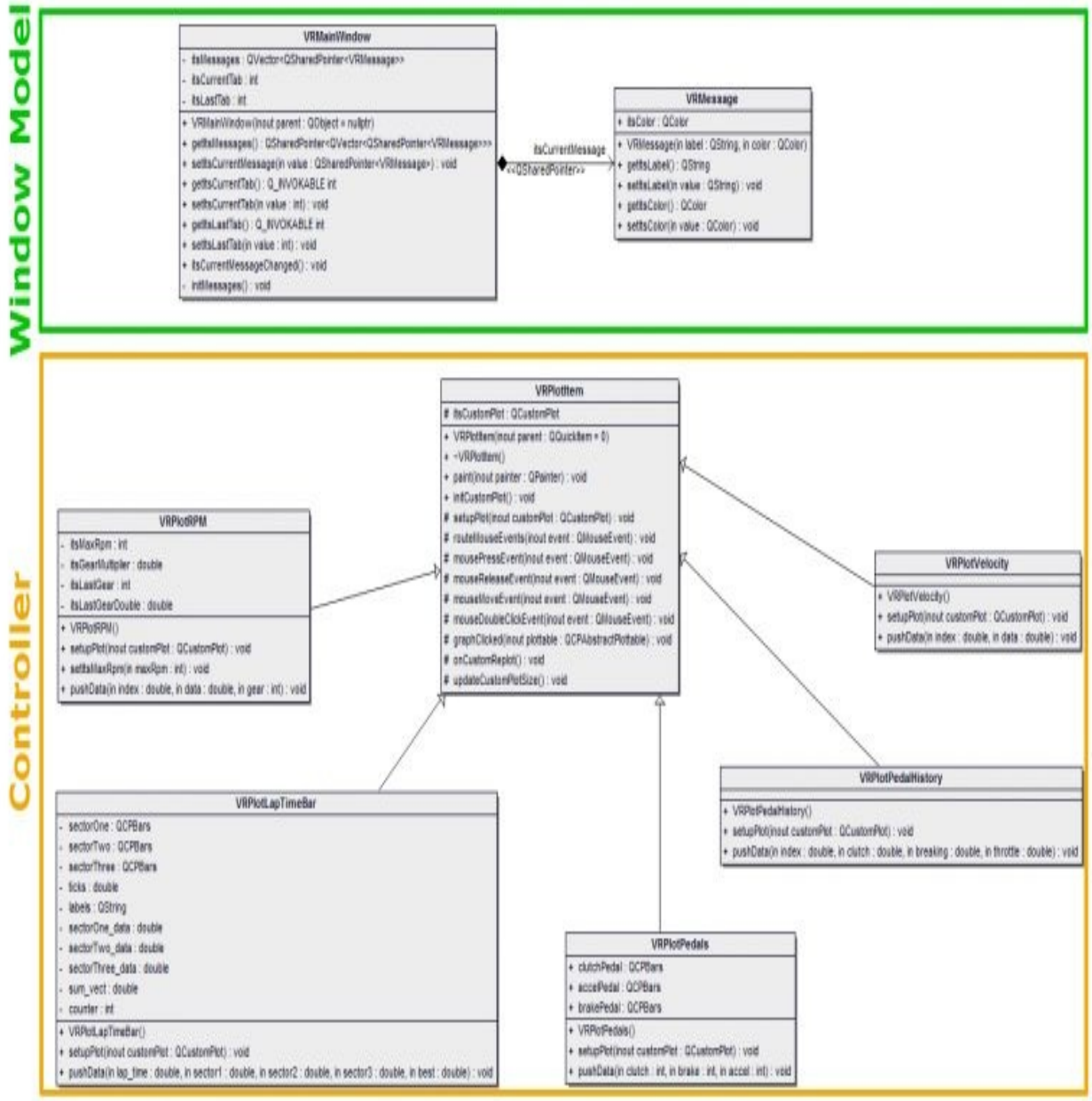
Additonally, we began working on the Software Architecture Document including the class diagrams. You can have a look at it on GitHub.

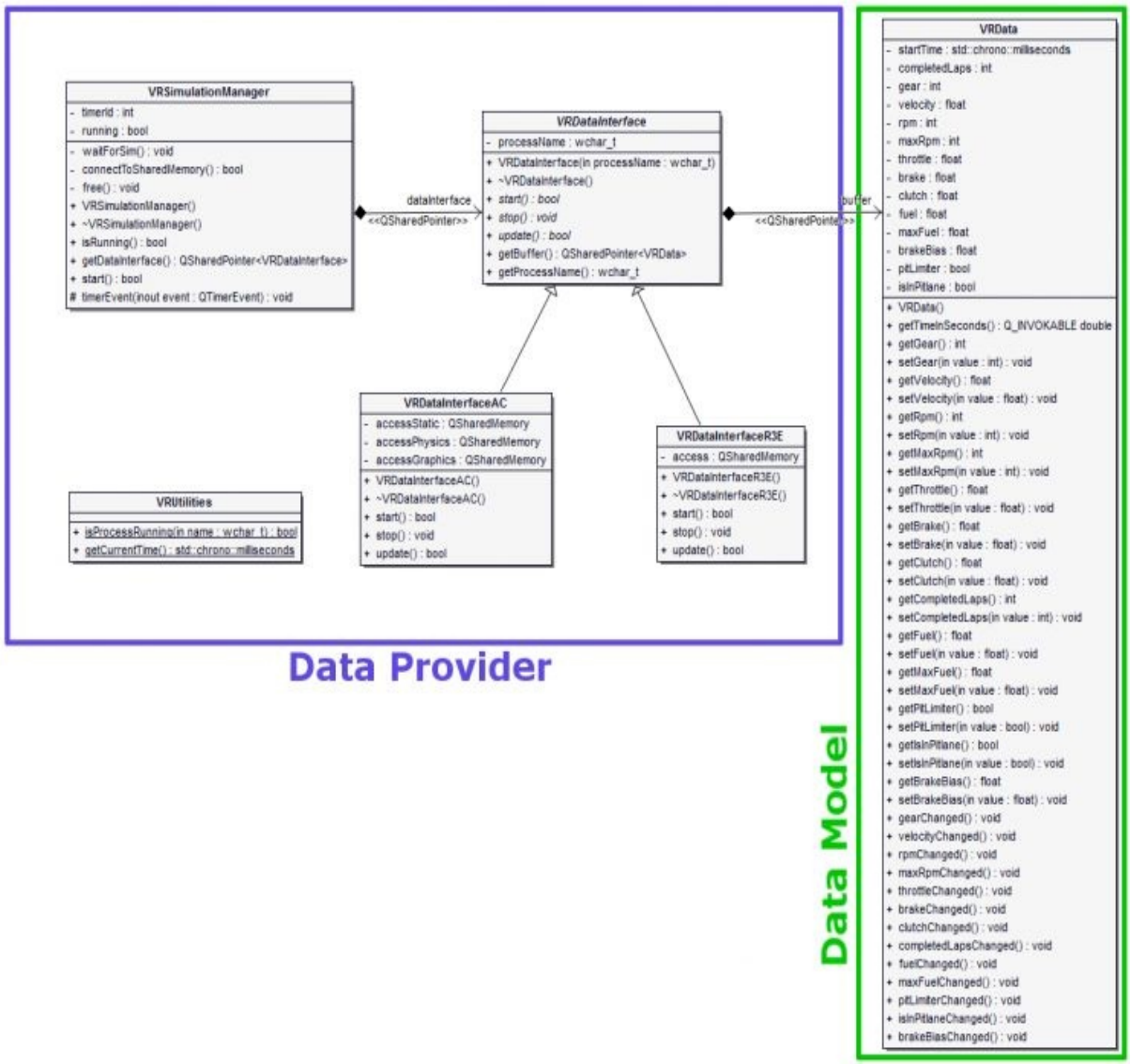
Our diagrams are available as SVG-Images here:

You are welcome to take a quick look at our class diagrams below:

custom QML-Object class diagram

datainterface class diagram


Window Model
VRMainWindow
- itsMessages : QVector<QSharedPointer<VRIMessage>>
- itsCurrentTab : int
- itsLastTab : int
+ VRIMainWindow(inout parent : QObject = nullptr)
+ getItsMessages() : QSharedPointer<QVector<QSharedPointer<VRIMessage>>>
+ setItsCurrentMessage(in value : QSharedPointer<VRIMessage>) : void
+ getItsCurrentTab() : Q_INVOKABLE int
+ setItsCurrentTab(in value : int) : void
+ getItsLastTab() : Q_INVOKABLE int
+ setItsLastTab(in value : int) : void
+ itsCurrentMessageChanged() : void
- initMessages() : void
itsCurrentMessage
<<QSharedPointer>>
VRIMessage
+ itsColor : QColor
+ VRIMessage(in label : QString, in color : QColor)
+ getItsLabel() : QString
+ setItsLabel(in value : QString) : void
+ getItsColor() : QColor
+ setItsColor(in value : QColor) : void
Controller
VRPlotItem
itsCustomPlot : QCustomPlot
+ VRPlotItem(inout parent : QQuickItem = 0)
+ ~VRPlotItem()
+ paint(inout painter : QPainter) : void
+ initCustomPlot() : void
# setupPlot(inout customPlot : QCustomPlot) : void
# routeMouseEvents(inout event : QMouseEvent) : void
# mousePressEvent(inout event : QMouseEvent) : void
# mouseReleaseEvent(inout event : QMouseEvent) : void
# mouseMoveEvent(inout event : QMouseEvent) : void
# mouseDoubleClickEvent(inout event : QMouseEvent) : void
# graphClicked(inout plottable : QCPAbstractPlottable) : void
# onCustomReplot() : void
# updateCustomPlotSize() : void
VRPlotRPM
- itsMaxRpm : int
- itsGearMultiplier : double
- itsLastGear : int
- itsLastGearDouble : double
+ VRPlotRPM()
+ setupPlot(inout customPlot : QCustomPlot) : void
+ setItsMaxRpm(in maxRpm : int) : void
+ pushData(in index : double, in data : double, in gear : int) : void
VRPlotVelocity
+ VRPlotVelocity()
+ setupPlot(inout customPlot : QCustomPlot) : void
+ pushData(in index : double, in data : double) : void
VRPlotPedalHistory
+ VRPlotPedalHistory()
+ setupPlot(inout customPlot : QCustomPlot) : void
+ pushData(in index : double, in clutch : double, in breaking : double, in throttle : double) : void
VRPlotLapTimeBar
- sectorOne : QCPBars
- sectorTwo : QCPBars
- sectorThree : QCPBars
- ticks : double
- labels : QString
- sectorOne_data : double
- sectorTwo_data : double
- sectorThree_data : double
- sum_vect : double
- counter : int
+ VRPlotLapTimeBar()
+ setupPlot(inout customPlot : QCustomPlot) : void
+ pushData(in lap_time : double, in sector1 : double, in sector2 : double, in sector3 : double, in best : double) : void
VRPlotPedals
+ clutchPedal : QCPBars
+ accelPedal : QCPBars
+ brakePedal : QCPBars
+ VRPlotPedals()
+ setupPlot(inout customPlot : QCustomPlot) : void
+ pushData(in clutch : int, in brake : int, in accel : int) : void



Permalink: https://visualracing.wordpress.com/2017/11/21/week7/

#8

Week 8 – Architecture

Hi guys,

as teased last week, we have created our Software Architecture Document according to the RUP-Template.

You can take a look at it here. Keep in mind, that our classes are still in development and the class diagrams might be changing accordingly.

Feel free to leave your thoughts in the comments below.

Kind regards,

your VisualRacing-Team

Permalink: https://visualracing.wordpress.com/2017/11/26/week8/

#9

Week 9 – Gantt Chart

Hi guys,

we have created our gantt chart. As the gantt chart created with YouTrack is rather confusing and offers limited possibilites , we additionally created a report which visualizes the cumulative flow of our issues (grouped by workflow).

One helpful aspect of the gantt chart is that it displays whether the time for a task was over- or underestimated. Moreover, the dependencies are somewhat visualized, but not really easy to understand.

The cumulative flow however shows us, in which phase our project is and which workflow currently is our main focus.

Feel free to comment below.

Sincerely,

the VisualRacing-Team

Permalink: https://visualracing.wordpress.com/2017/12/01/week9/

#10

Midterm Summary

This blog post is a summary of all the work we have done so far. Below you can find the links to each blog post (one for every week).

You can have a look at our midterm presentation on Prezi.

You can find the code to our project on GitHub.

Feel free to take a look at our Project Management tool YouTrack.

Permalink: https://visualracing.wordpress.com/2017/12/13/midterm/

#11

Semester II: Week 1&2 – New Scope, Risk Management

We're Back

After a short break due to the scheduling of the DHBW, we're back and we redefined our scope for the upcoming time. Additionally we set us five Use Cases to work on.

To start off with, we sat together and had a look at our project, our current state and our code to recall our progress.

Our Scope

Below you can see our updated scope including the five Use Cases we will work on this semester.

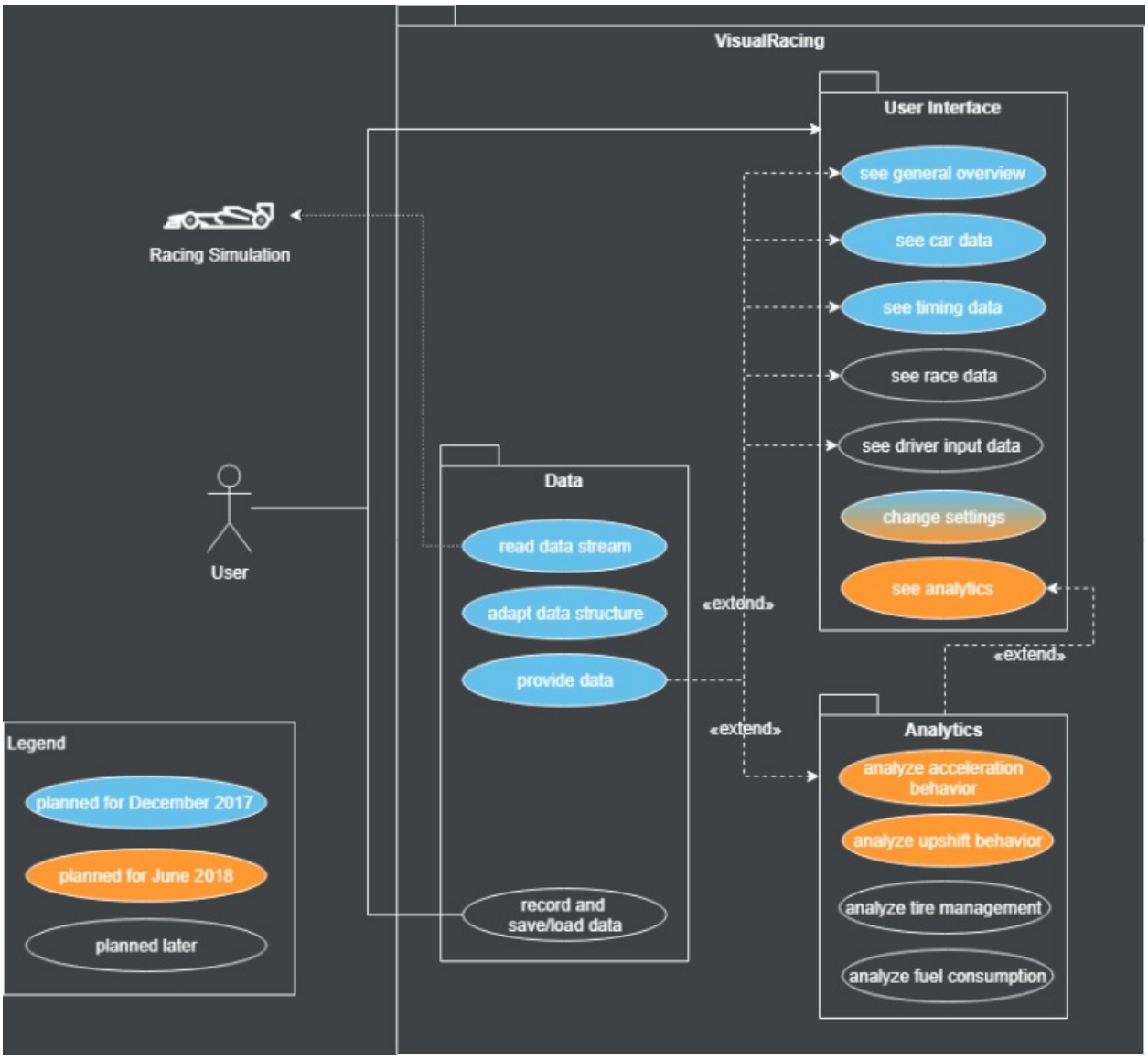
The Use Case Diagram can be found in the SRS.
The added specifications for the seleced UCs can be found on GitHub:

Risk Management

To be prepared for any potential risks we've done some risk management to be aware of unfortunate incidents which might occur. You can see a screenshot below as well as find it on GitHub.

Time Spent

To summarize the work we've done so far and the time spent, we created a time table, which should give you a good overview. You can see a screenshot below as well as find it on GitHub. Additionally you can find the generated overview as a YouTrack Report here.


VisualRacing
User Interface
see general overview
see car data
see timing data
see race data
see driver input data
change settings
see analytics
Racing Simulation
User
Data
read data stream
adapt data structure
provide data
record and save/load data
«extend»
«extend»
«extend»
Analytics
analyze acceleration behavior
analyze upshift behavior
analyze tire management
analyze fuel consumption
Legend
planned for December 2017
planned for June 2018
planned later

Risk Name	Risk Description	Risk Probability of Occurrence	Risk Impact	Risk Mitigation	Person in Charge of Tracking	Risk Factor
Trouble in feature implementation	wrong implementation, bugs, ...	80%	3-7	precise specification, read documentation	all	2.4 - 5.6
Communicational Problems	bad or wrong communication between the team members, the customer	30%	5-7	communication!	all	1.5 - 2.1
Hardware Problems	Simulator hardware fails	30%	3	replacement hardware is available	Felix	0.9
Loss of team member	Losing a valuable team member due to various reasons, including sickness, death, bad grades, fireing people, loss of Staatsbürgerschaft, ...	5%	8	live healthy	all	0.4
YouTrack Server	Server problems leed to system problems	10%	3	Continuous conversation with Server Responsible	Lars	0.3
YouTrack	Project Management tool changes terms of use	5%	3	read Youtrack News	Chris	0.15
Qt API/ABI	Proramming Interface/ Framework Interface changes	1%	1-10	CI service	Lars	0.01 - 0.1
Raceroom	Game Interface changes	1%	1-10	CI service	Felix	0.01 - 0.1
Assetto Corsa	Game Interface changes	1%	1-10	CI service	Felix	0.01 - 0.1

Use Case	Documentation	Coding	Testing	Warm-Up time	Total
General Tab	51min	8h57min	-	7h29min	17h17min
Data Mapping	31min	8h38min	-	3h35min	12h44min
Timing Tab	1h9min	7h	-	-	8h9min
Car Data Tab	1h32min	2h27min	-	-	3h59min
Settings Tab	23min	1h30min	-	-	1h53min
Sum	**4h26min**	**28h32min**	**-**	**11h4min**	**44h2min**

Permalink: https://visualracing.wordpress.com/2018/04/09/2week12/

#12

Semester II: Week 3 – Function Points

Hey everyone!

This week we worked on evaluating our work and the time it took, based on Function Points in order to estimate how long it will take to implement the Use Cases we set us as a task for this semester.

But what are Function Points?

This site, which also provides the tool we used for calculating the Function Points, explains the methodology as follows:

> "Function Point is a method of estimating software project costs. By making simple estimates of the software you or your team plan to develop, the number of function points for your project can be determined. Using these function points, software developers can estimate various costs such as the cost in time to develop the software, or estimate the number of lines of code it will take to develop the software in question."
>
> ~ http://groups.umd.umich.edu/cis/course.des/cis375/projects/fp

In general, the Functions Points for a project are calculated based on the number of User Inputs, Outputs, Inquiries, as well as the number of Files and External Interfaces. Moreover, these amounts are differently weighted depending on their complexity (simple, average or complex). Those complexities can be calculated based on the amount of DETs, RETs and FTRs per use case. As those metrics are not representative for our project, as the complexity for our use cases comes mainly from more complex internal calculations, mappings between different data structures and algorithms. We determined the complexity based on our

expert knowledge and insights into our own application. If you want to know how a specific complexity was determined, feel free to ask.

This is an example for the number of In- and Outputs etc. (on which the Function Point calculation is based) for the Use Case Timing Tab:

	Simple	Average	Complex
Number of User Inputs	0	0	0
Number of User Outputs	1	3	1
Number of User Inquiries	1	0	0
Number of Files	0	0	0
Number of External Interfaces	0	2	0

		0 1 2 3 4 5
1.	Does the system require reliable backup and recovery?	◉ ○ ○ ○ ○ ○
2.	Are data communications required?	○ ○ ○ ○ ○ ◉
3.	Are there distributed processing functions?	○ ○ ○ ○ ◉ ○
4.	Is performance critical?	○ ○ ○ ○ ◉ ○
5.	Will the system run in an existing, heavily utilized operational environment?	○ ○ ○ ○ ◉ ○
6.	Does the system require on-line data entry?	◉ ○ ○ ○ ○ ○
7.	Does the on-line data entry require the input transaction to be built over multiple screens or operations?	◉ ○ ○ ○ ○ ○

		0 1 2 3 4 5
8.	Are the master files updated on-line?	◉ ○ ○ ○ ○ ○
9.	Are the inputs, outputs, files, or inquiries complex?	○ ○ ○ ◉ ○ ○
10.	Is the internal processing complex?	○ ○ ○ ○ ◉ ○
11.	Is the code designed to be reusable?	○ ○ ○ ○ ◉ ○
12.	Are conversion and installation included in the design?	○ ○ ◉ ○ ○ ○
13.	Is the system designed for multiple installations in different organizations?	○ ○ ○ ○ ○ ◉
14.	Is the application designed to facilitate change and ease of use by the user?	○ ○ ○ ○ ◉ ○

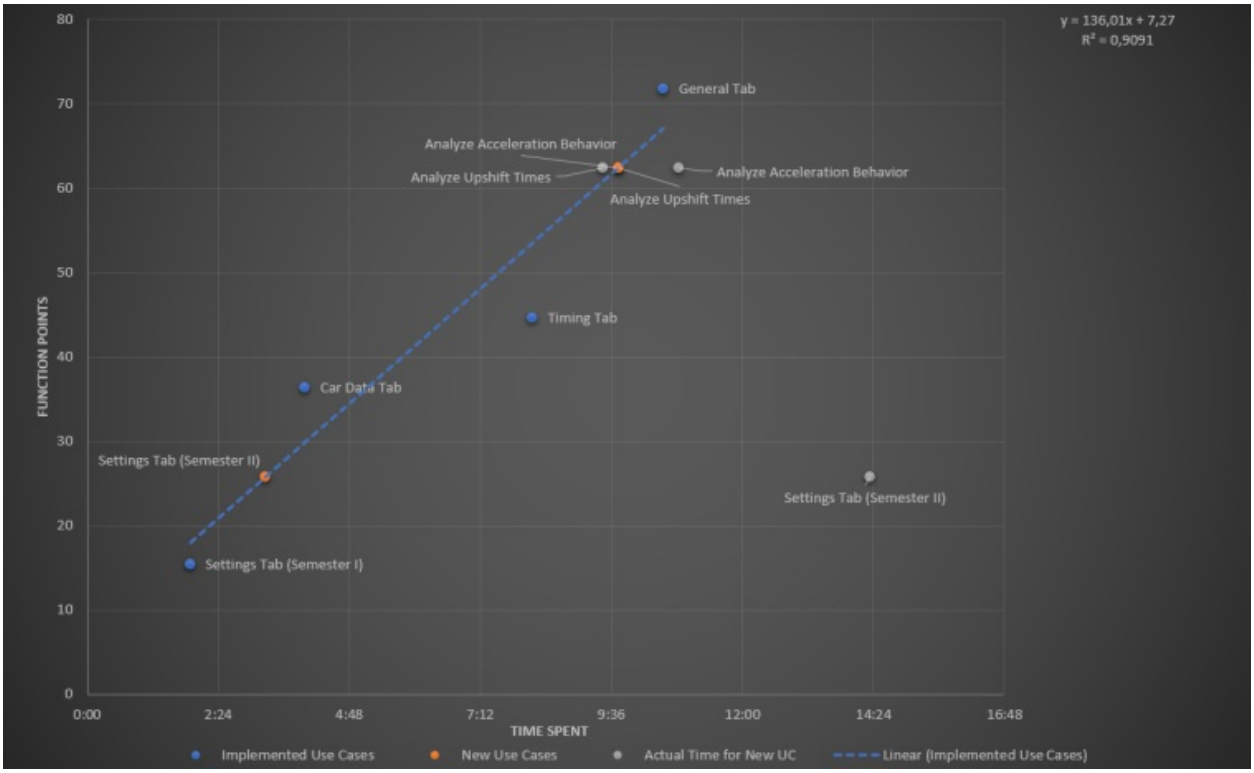
Using this tool and scheme we calculated the Function Points for the following Use Cases and with our spent time calculated an estimation function:

With this estimation function we were able to estimate the time needed for these Use Cases:

Here you can see an overview of our Use Cases and the estimated time for the Use Cases we are going to implement in the next few months.

Use Case	Time spent (h:mm)	Time spent (h:mm)	Function Points
General Tab	10:33	10:33	71,8
Timing Tab	8:09	8:09	44,7
Car Data Tab	3:59	3:59	36,4
Settings Tab (Semester I)	1:53	1:53	15,4
New Use Cases	**Time Estimation (h:mm)**	**Time spent (h:mm)**	**Function Points**
Analyze Acceleration Behavior	*9:43*	*10:50*	*62,4*
Analyze Upshift Times	*9:43*	*9:27*	*62,4*
Settings Tab (Semester II)	*3:16*	*14:20*	*25,8*
x = (y - 7,27) / 136,01			
x: time			
y: function points			

Additionally we generated a chart for better understanding of the time estimation:



You can find the Excel file containing all of this information here.
Permalink: https://visualracing.wordpress.com/2018/04/19/2week3/

#13

Semester II: Week 4 – Testing

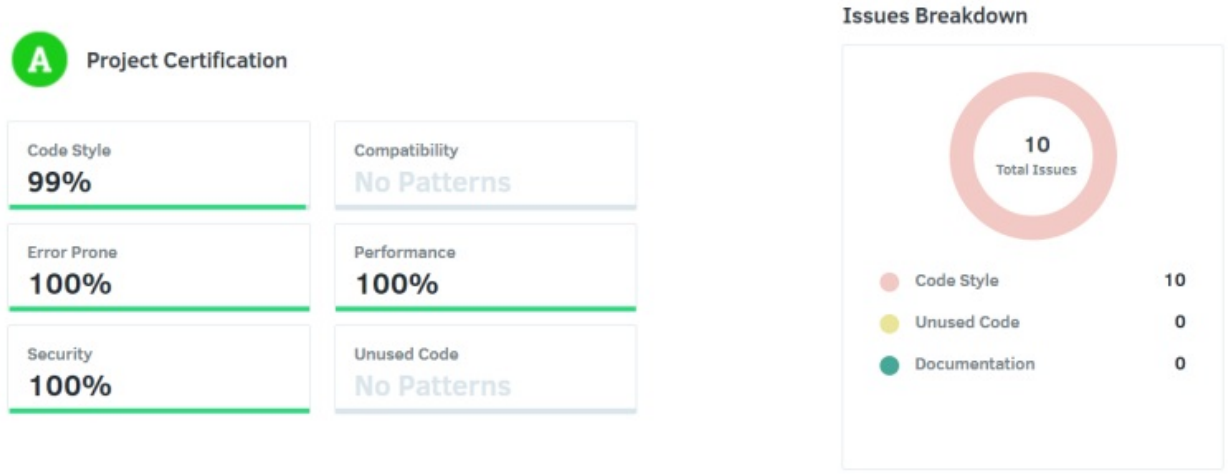
Hey everyone, we're back with a new blog post all about Testing!

During the last week we were busy writing our Test Plan containing all the information needed to plan and control the test workflow for our project. Feel free to have a look at it!

For our test workflow we setup Travis-CI (Linux) and AppVeyor (Windows) for automatically building and testing our latest commit on GitHub. You can find the build files integrating the tests here and here.

Additionally these tools are integrated into our GitHub workflow, as each commit is marked with a build status, to show whether the changes build correctly. The last build status is also displayed on the project starting page on GitHub,

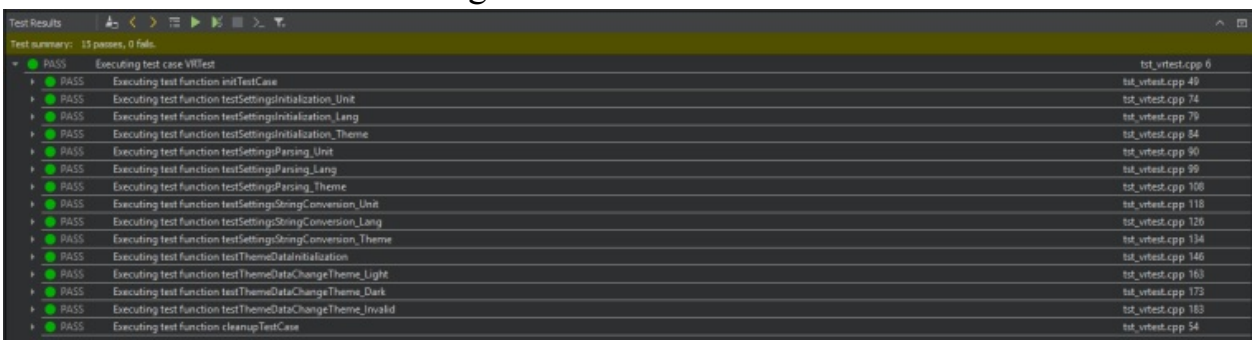
Furthermore we integrated codacy into our workflow. This tool is an indicator for the code quality and helps us – the developers – to find and fix issues.



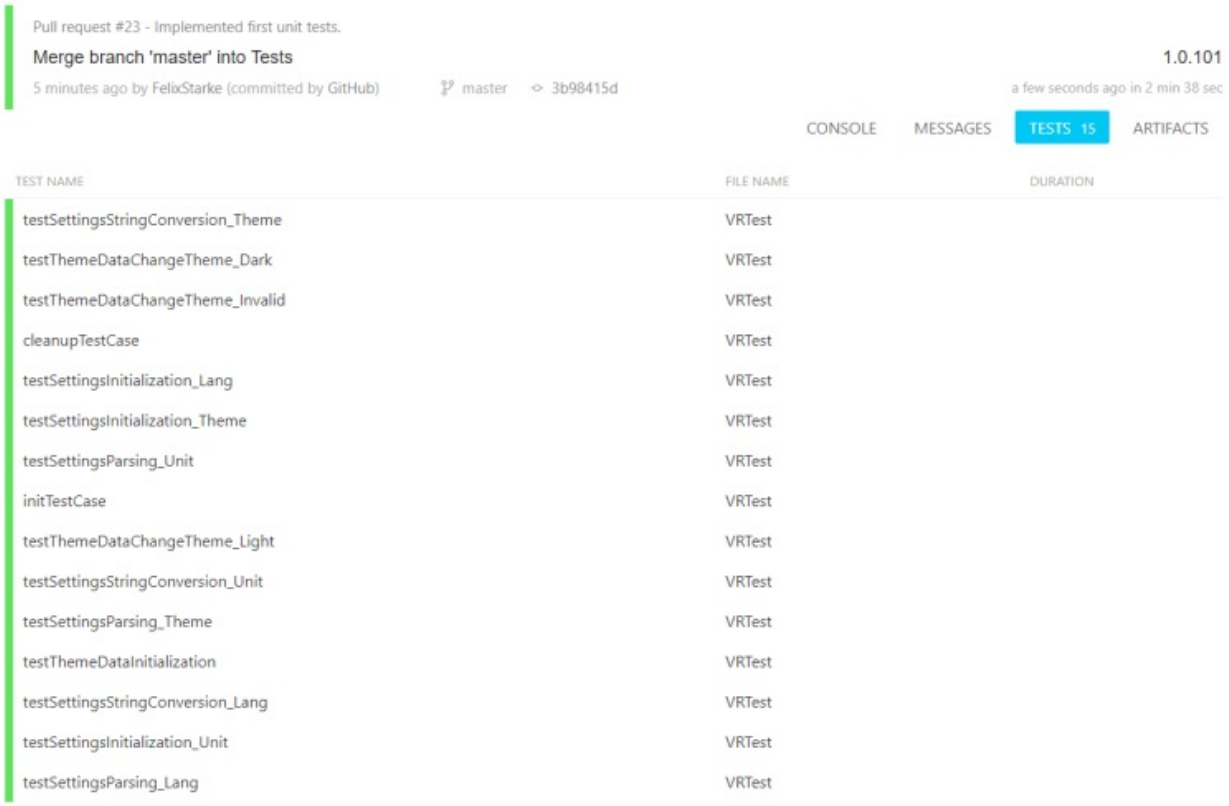
Codacy Overview

In order for getting the Tests to run in the Qt Creator we had to change our project structure a little bit. So here you can find our main application and here you can have a look at the subproject which contains our test code. You can take a look at the test source here.

You can see all tests running in the screenshots below.



All tests running in our IDE (higher resolution).



All tests running in AppVeyor automatically after each commit.

To get a better understanding of our tests and their coverage we integrated codecov into our testing workflow. After every build on AppVeyor and after the test run is complete, the code coverage results are calculated and sent to codecov. You can have a look at the dashboard here.

Additionally, we added a badge to our ReadMe on Github to display the current code coverage, which is at nearly 70% for the classes we are testing.

Codecov also gives us an insight into how the coverage changes with every commit and whether a pull request has a positive impact on the coverage.

Permalink: https://visualracing.wordpress.com/2018/05/03/2week4/

#14

Semester II: Week 5 – Refactoring

This week we learned some refactoring techniques based on *Martin Fowler's* book **Refactoring: Improving the Design of Existing Code**.

You can find links to the repositories containing the source code and some test code of each team member on GitHub, as well as direct links to the corresponding *Codacy* dashboard.

Lars Hübner: GitHub | Codacy

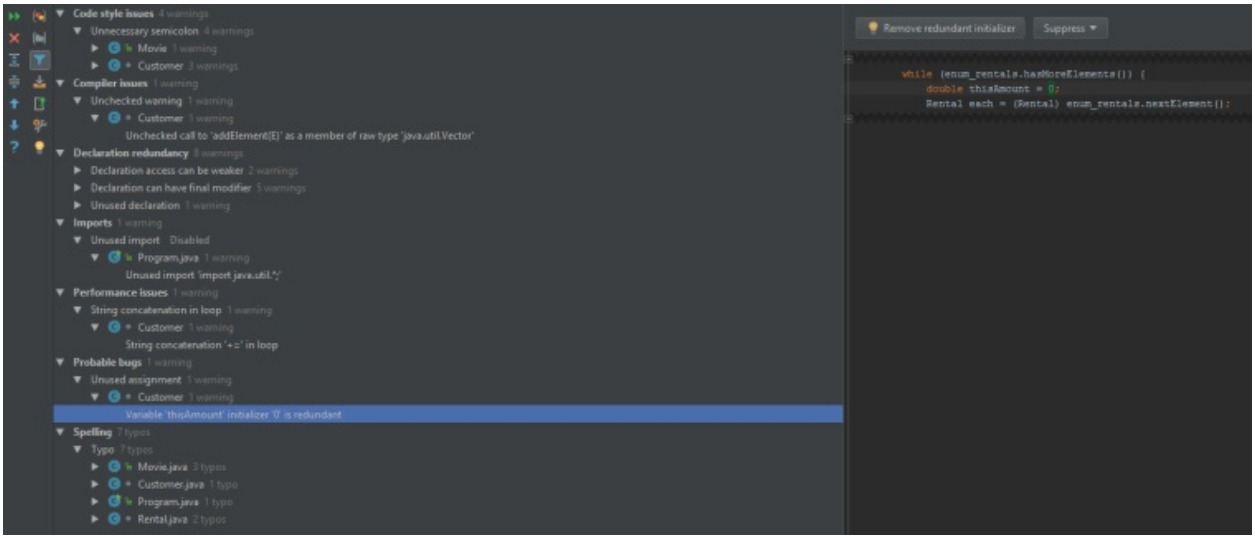
Christopher Klammt: GitHub | Codacy

Felix Starke: GitHub | Codacy

As all of us used Jetbrains' IntelliJ for the refactoring work, we all used IntelliJ's Code Inspection feature.

> "IntelliJ IDEA features robust, fast, and flexible static code analysis. It detects the language and runtime errors, suggests corrections and improvements before you even compile."
>
> ~ https://www.jetbrains.com/help/idea/code-inspection.html

This feature analyzes the source code and finds language and runtime errors, unused imports, typos and more. Additionally the code inspection suggests ways to correct those issues. It even detects performance critical issues and provides functionality to fix those issues with a click of a button.



IntelliJ’s Code Inspection (high resolution)

Permalink: https://visualracing.wordpress.com/2018/05/08/2week5/

#15

Semester II: Week 6 – Patterns

Hi guys,

in this weeks blog post we are going to talk about design patterns.

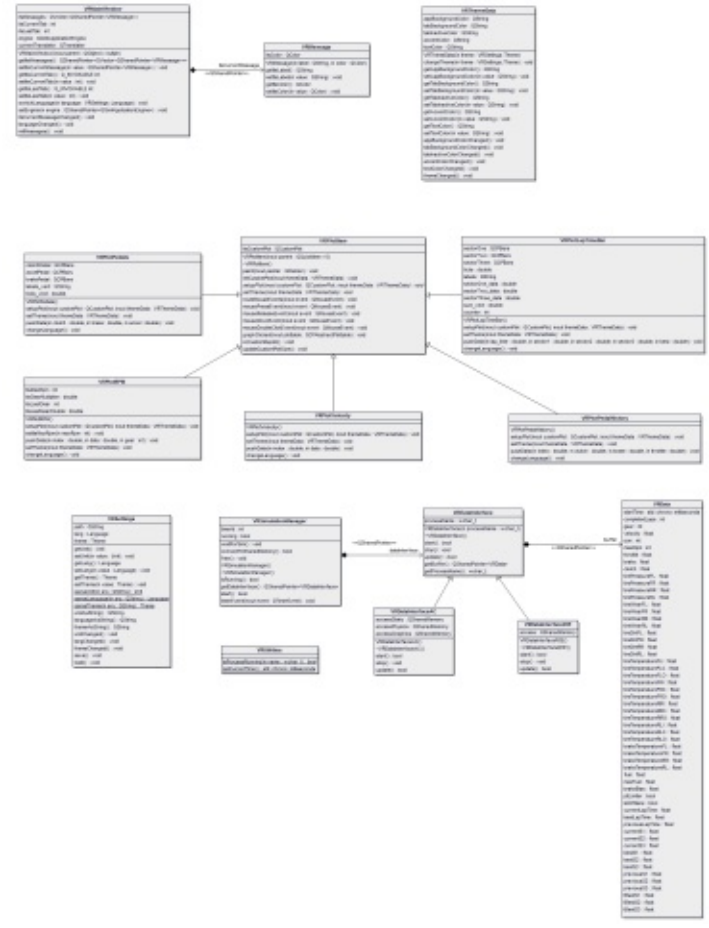
> “In software engineering, a software design pattern is a general, reusable solution to a commonly occurring problem within a given context in software design. It is not a finished design that can be transformed directly into source or machine code. It is a description or template for how to solve a problem that can be used in many different situations. Design patterns are formalized best practices that the programmer can use to solve common problems when designing an application or system.”
>
> ~
>
> https://en.wikipedia.org/wiki/Software_design_pattern

Up to this point we are already using design patterns in our project. On one side our project is based on the MVC-Pattern, which means our architecture is split into the three parts Model, View and Controller. Next to that our project heavily relies on the Observer Pattern. We are using it for notifying our UI elements when the corresponding data elements change.

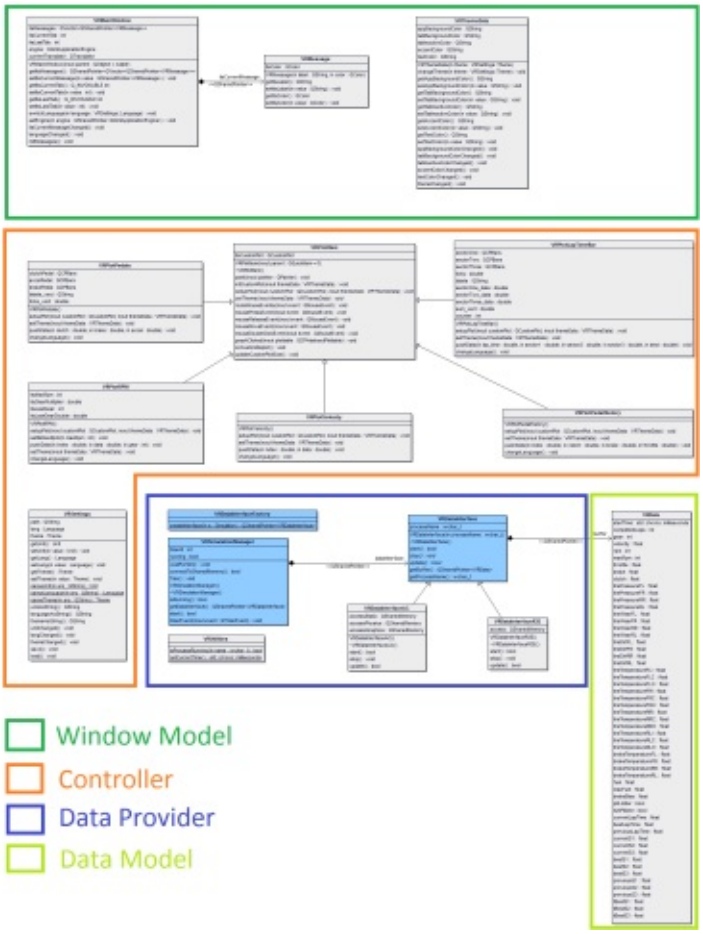
To fulfill the requirements for this weeks grading criteria, we’ve decided to implement a Factory Pattern for our data interface classes. This means, extracting the instantiation of our different types of data interfaces for the different racing simulations from our *SimulationManager*-class to a new factory class. We have decided to implement this pattern, as it simplifies data interface

instantiation for us as developers. We don’t have to think about how the specific name of a data interface class every time we want to create one. Instead, we just have to pass an enum value representing the corresponding racing simulation. Furthermore using the factory pattern cleans up source code in the *SimulationManager*.

You can see the things we’ve changed highlighted in blue in the UML class diagrams below. Additionally we have added our factory class *VRDataInterfaceFactory* in the code snippet at the end of this blog post. For even more insights in what has changed feel free to take a look at all of our code on GitHub or the Header– and Source-File of our factory class. You can find the commit containing the changes here.



Before



After

The following code snippet shows a slightly simplified version of our factory class *VRDataInterfaceFactory.* This factory enables us, to easily instantiate data interface objects by calling the static *createInterface*-method.

```
class VRDataInterfaceFactory
{
public:
    enum Simulation { AC, R3E };
    static QSharedPointer<VRDataInterface> createInterface(Simulatio
};

QSharedPointer<VRDataInterface> VRDataInterfaceFactory::createInterf
{
    if (s == Simulation::AC)
```

```
        return QSharedPointer<VRDataInterface>(new VRDataInterfaceAC
    else if (s == Simulation::R3E)
        return QSharedPointer<VRDataInterface>(new VRDataInterfaceR3
}
```

Permalink: https://visualracing.wordpress.com/2018/05/27/2week6/

#16

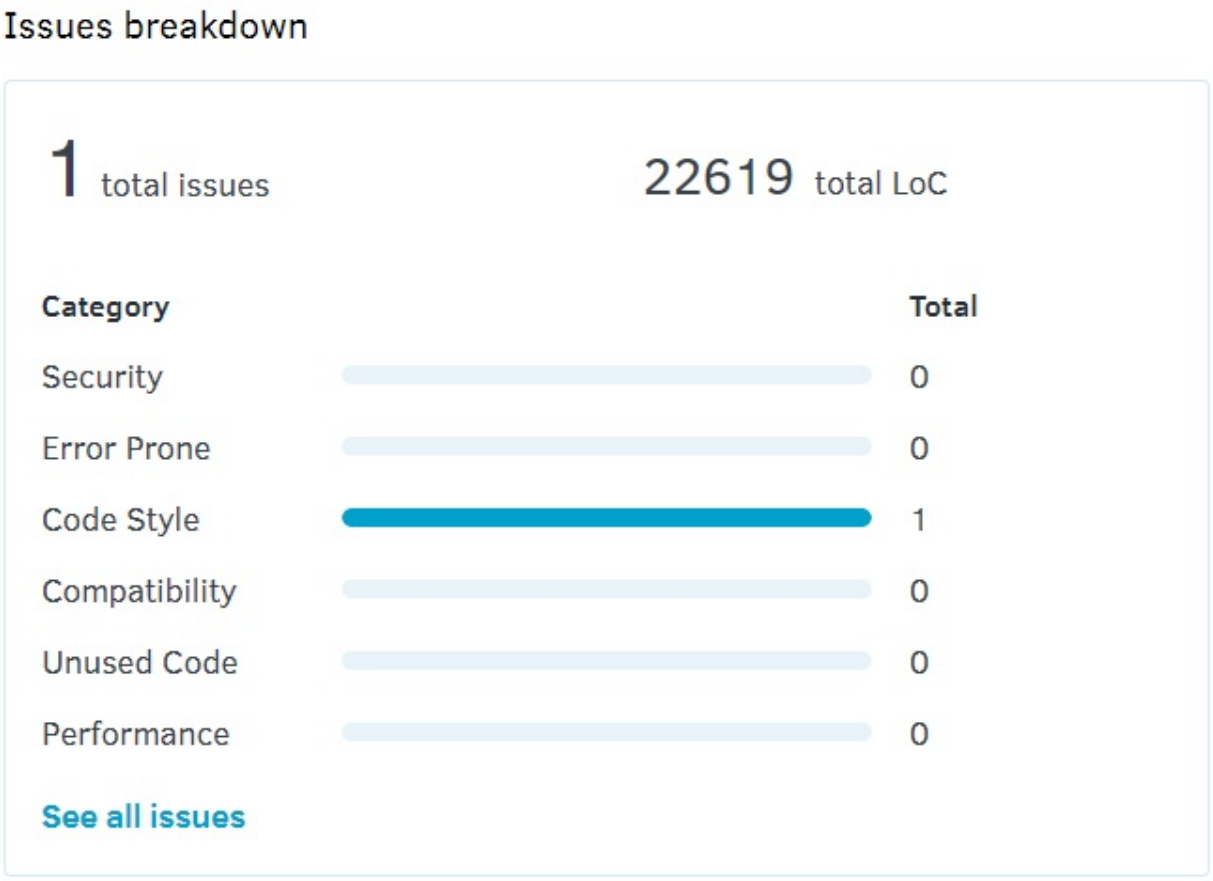
Semester II: Week 7 – Metrics

This week we've been working with Metrics to improve our code!

> "Metrics are an important component of quality assurance, management, debugging, performance, and estimating costs, and they're valuable for both developers and development team leaders.
>
> Software metrics offer an assessment of the impact of decisions made during software development projects. "
>
> ~ Stackify

To get a good overview of our code quality we setup Codacy to track issues automatically with each commit to the master branch. You can have a look at the Codacy dashboard here. Additionally Codacy calculates a rating for each project based on these issues, which we included in form of a badge in our GitHub ReadMe.

The problem is that Codacy doesn't really offer any helpful metrics for C++ except for pointing out some minor issues which we already resolved.



Codacy: Issue Overview

Codacy: Example Issue (Details)

CCCC

To get a deeper insight into our code by using metrics, we ran the program CCCC (C and C++ Code Counter). CCCC is a tool which analyzes C++ files and

generates an HTML report on various metrics of the code. Metrics supported include lines of code, McCabe's complexity and much more.

In the following you can see a few screenshots of this report before our refactoring:

Metric	Tag	Overall	Per Module
Number of modules	NOM	115	
Lines of Code	LOC	6267	54.496
McCabe's Cyclomatic Number	MVG	1169	10.165
Lines of Comment	COM	4235	36.826
LOC/COM	L_C	1.480	
MVG/COM	M_C	0.276	
Information Flow measure (inclusive)	IF4	7437	64.670
Information Flow measure (visible)	IF4v	7437	64.670
Information Flow measure (concrete)	IF4c	29	0.252
Lines of Code rejected by parser	REJ	431	

Project Summary

Module Name	LOC	MVG	COM	L_C	M_C
VRData	*969*	169	53	*18.283*	3.189
VRDataInterface	26	2	0	******	------
VRDataInterfaceAC	98	12	20	4.900	0.600
VRDataInterfaceR3E	86	8	8	*10.750*	1.000
VRMainWindow	90	6	29	3.103	0.207
VRMessage	35	2	8	4.375	------
VRPlotItem	92	3	3	**30.667**	------
VRPlotLapTimeBar	111	1	15	*7.400*	------
VRPlotPedalHistory	63	0	7	*9.000*	------
VRPlotPedals	95	0	10	*9.500*	------
VRPlotRPM	75	1	8	*9.375*	------
VRPlotVelocity	53	0	10	5.300	------
VRSettings	136	40	3	**45.333**	**13.333**
VRSimulationManager	91	17	6	*15.167*	2.833
VRThemeData	106	7	1	**106.000**	*7.000*
VRUtilities	33	7	4	*8.250*	1.750

Procedural Metrics Summary

CCCC helps us by providing procedural metrics and pointing out, which classes potentially could be a problem regarding these metrics.

For Example VRData has lots of Lines of Code, but we can't reduce these.

In general CCCC points out, that we didn't use much comments in our classes (as pointed out by *L_C: Lines of Code per Comment*). We don't think this is a big problem, but we will keep it in mind for our further implementations.

Module Name	WMC1	WMCv	DIT	NOC	CBO
VRData	114	0	1	0	2
VRDataInterface	7	7	0	2	4
VRDataInterfaceAC	5	2	1	0	2
VRDataInterfaceR3E	5	2	1	0	3
VRMainWindow	12	0	1	0	3
VRMessage	5	5	0	0	2
VRPlotItem	12	0	1	5	11
VRPlotLapTimeBar	5	0	2	0	3
VRPlotPedalHistory	5	0	2	0	3
VRPlotPedals	5	0	2	0	3
VRPlotRPM	6	0	2	0	3
VRPlotVelocity	5	0	2	0	3
VRSettings	15	0	1	0	5
VRSimulationManager	8	0	1	0	2
VRThemeData	12	0	1	0	9
VRUtilities	2	1	0	0	1

Object Oriented Design

WMC: Weighted methods per class DIT: Depth of inheritance tree NOC: Number of children CBO: Coupling between objects

VRData has a lot of functions as each data member we read from the simulation has a getter and setter. Nothing we can improve there.

VRPlotItem has 5 children, which is marked yellow as a warning, but it is necessary to do it that way.

Module Name	Fan-out			Fan-in			IF4		
	vis	con	inc	vis	con	incl	vis	con	inc
VRData	0	0	0	2	2	2	0	0	0
VRDataInterface	2	2	2	2	1	2	16	4	16
VRDataInterfaceAC	0	0	0	1	2	2	0	0	0
VRDataInterfaceR3E	0	0	0	1	2	3	0	0	0
VRMainWindow	0	0	0	3	3	3	0	0	0
VRMessage	0	0	0	2	2	2	0	0	0
VRPlotItem	5	5	5	6	1	6	900	25	*900*
VRPlotLapTimeBar	0	0	0	3	1	3	0	0	0
VRPlotPedalHistory	0	0	0	3	1	3	0	0	0
VRPlotPedals	0	0	0	3	1	3	0	0	0
VRPlotRPM	0	0	0	3	1	3	0	0	0
VRPlotVelocity	0	0	0	3	1	3	0	0	0
VRSettings	0	0	0	5	2	5	0	0	0
VRSimulationManager	0	0	0	2	1	2	0	0	0
VRThemeData	6	0	6	3	2	3	324	0	*324*
VRUtilities	0	0	0	1	0	1	0	0	0

Structural Metrics Summary

FI: Fan-in (number of modules which pass information into the current module) FO: Fan-out (number of modules into which the current module passes information) IF4: Information Flow measure

One thing to mention is that VRData has no Fan-out values and no Information Flow, which is weird, because this is where all of our data is read and passed on.

This probably is because we pass the information using Signals. Signals and Slots are a central feature of Qt and aren't recognized by CCCC.

Refactoring

One Problem we addressed is the McCabe complexity of VRData, which is rather high with 169. Furthermore we added a few comments in VRSettings and VRThemeData, as CCCC points out, that these classes have a rather high McCabe complexity in relation to the Number of Comments.

VRData	978	108	58	16.862	1.862
VRDataInterface	26	2	0	******	------
VRDataInterfaceAC	101	12	20	5.050	0.600
VRDataInterfaceFactory	17	4	0	------	------
VRDataInterfaceR3E	92	8	8	11.500	1.000
VRMainWindow	90	6	29	3.103	0.207
VRMessage	35	2	8	4.375	------
VRMetrics	30	1	0	******	------
VRMetricsManager	44	2	1	44.000	------
VRPlotItem	92	3	3	30.667	------
VRPlotLapTimeBar	115	1	15	7.667	------
VRPlotPedalHistory	63	0	7	9.000	------
VRPlotPedals	98	0	10	9.800	------
VRPlotRPM	79	1	8	9.875	------
VRPlotVelocity	53	0	10	5.300	------
VRSettings	136	40	11	12.364	3.636
VRSimulationManager	92	17	6	15.333	2.833
VRThemeData	106	7	4	26.500	1.750
VRUtilities	33	7	4	8.250	1.750

Procedural Metrics Summary – After Refactoring

As you can see, we improved the McCabe Complexity of the VRData class by reducing it from 169 to 108 (for further insights you can have a look at the commit). Furthermore we improved the Complexity per Comment for VRSettings and VRThemeData (here is the corresponding commit).

I hope we could give you a short introduction into the tool CCCC and show you how we improved our code with the help of these metrics.

Permalink: https://visualracing.wordpress.com/2018/05/29/2week7/

#17

Semester II: Week 8 – Installation

In order to spread our software, we integrated the automatic deployment of our application into our AppVeyor Build Process.

For each commit you can download the Artifact as a zip-file containing all necessary files to run the application (for example here). Furthermore, you can always download the latest build from this page or use the direct download link:

After downloading the build, you just have to unzip it and run the VRMain.exe.

Alternatively, we configured AppVeyor as such, that when tagging a commit on GitHub as a Release, the artifact will be added to this release. These releases can be found right here.

Permalink: https://visualracing.wordpress.com/2018/06/06/2week8/

#18

Final Summary

This last blog post is a summary of all the work we've done while following our Vision!

Requirements

- Software Requirements Specification (SRS)
  - Overall Use Case Diagram
- Use Cases
  - Display General Data (feature file)
  - Display Car Data (feature file)
  - Display Timing Data (feature file)
  - Change Settings (feature file)
  - Display Analytics
  - Read Data Stream
  - Analyze Acceleration Behavior
  - Analyze Upshift Behavior
- Testing according to Test Plan
  - Testing (Blog Post)
  - Gherkin (Blog Post)
  - Test Code
  - Test Log (latest Test Run on AppVeyor)

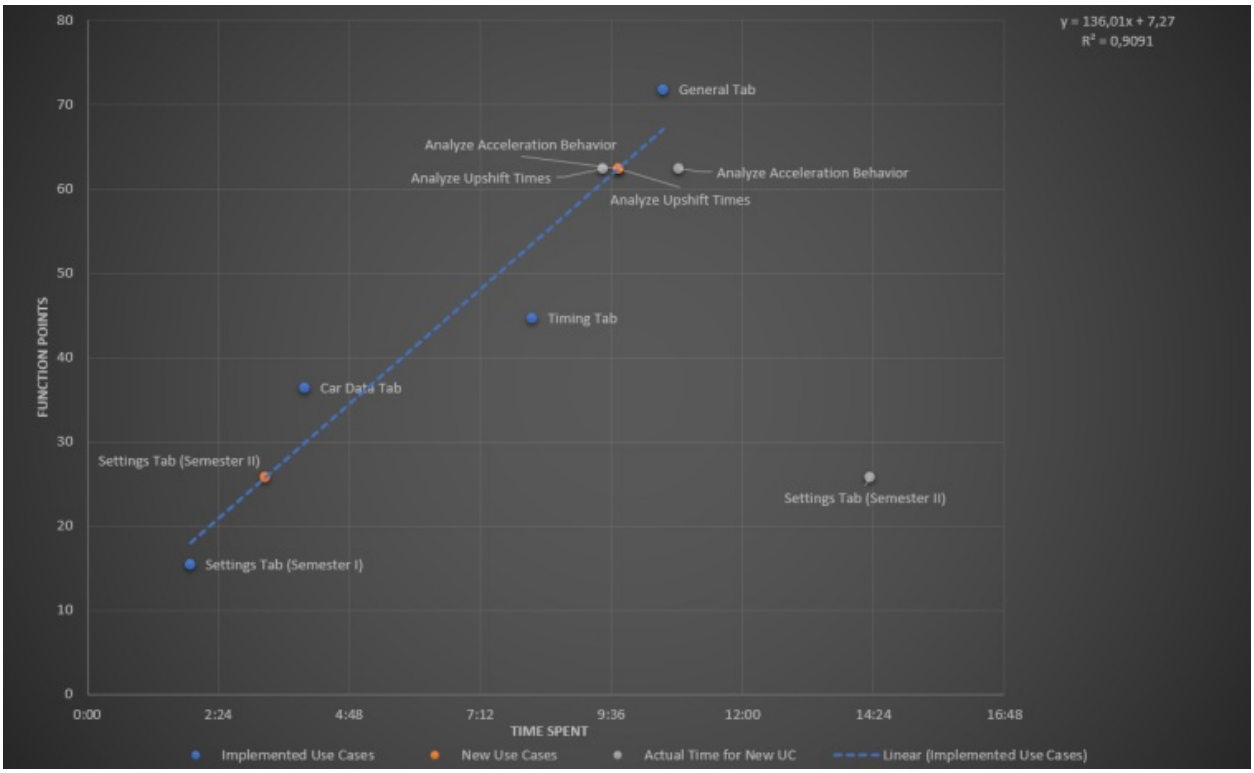
Project Management

- RUP (Blog Post)
  - Cumulative Flow (Semester I)
  - Cumulative Flow (Semester II)
- Time Spent per Person

- Scrum ([Blog Post])
  We organized our working with SCRUM using YouTrack. You can see two burndown charts here and more on our [YouTrack Dashboard].

- Function Points (FP)

- Blog Post explaining everything
- Time Estimation for the Use Cases
- Velocity Graph (Excel)



Ability to Execute

- Deployed Application on AppVeyor and GitHub Releases (explained in Installation Blog Post)
- Demo
  We showed the application in class and during our final presentation.
- Code
  We version controlled our project using GitHub

Architecture

- Software Architecture Document (SAD) including
  - Overall Class Diagram

        - Controller + Window Model
        - Data Provider + Model
    - Information on the Factory Pattern
- In Detail:
    - Week 2 – Team and Technology
    - Week 7 – Class Diagrams
- Patterns
    - Semester II: Week 5 – Fowler's Refactoring
    - Semester II: Week 6 – Patterns

Configuration Management

- Test Plan
- In Detail:
    - Semester II: Week 7 – Metrics
      -> Codacy and CCCC
    - Semester II: Week 1&2 – New Scope, Risk Management
      -> Risk Management Table
    - Automated Testing
    - Continuous Delivery

Presentations

- Midterm Presentation
- Presentation at DevCamp 2018
- Final Presentation

For the archive

- Blog as PDF
- Runnable Software as ZIP

- GitHub Repository as ZIP

Permalink: https://visualracing.wordpress.com/2018/06/18/final/