

# BEP Roparun

L. Overdevest  
M. van Meerten  
J. Nieuwdorp

Scheduling an international relay-run  
Bachelor thesis



# BEP Roparun

by

L. Overdevest  
M. van Meerten  
J. Nieuwdorp

in partial fulfillment of the requirements for the degree of

**Bachelor of Science**  
in Computer Science & Engineering

at the Delft University of Technology,  
To be presented publicly on Friday February 1, 2019.

Project duration: November 12, 2018 – February 1, 2019  
Thesis committee: Dr. Ir. H. Wang, TU Delft, Bachelor Project Coordinator  
Ir. O. Visser, TU Delft, Bachelor Project Coordinator  
Dr. Ir. M. T. J. Spaan, TU Delft, Coach  
W. Vloet, Roparun, Client

An electronic version of this thesis is available at <http://repository.tudelft.nl/>.

# Foreword

This report concludes the TI3806 Bachelorproject of the bachelor Computer Science & Engineering at the Delft University of Technology. Over the course of ten weeks, we, Martijn van Meerten, Jasper Nieuwdorp and Lennart Overdevest developed a scheduling application for the Roparun Foundation.

We would like to thank the people that supported us during the project. The assistance and guidance offered by Dr. Ir. M.T.J. Spaan was greatly appreciated. The clear and useful feedback received during our meetings were of great importance to our project. We would like to thank Erwin Walraven for sharing his knowledge with regards to Linear Programming algorithms and taking the time to guide us in that area. Finally, we would like to offer our special thanks to everyone of the Roparun Foundation who was involved during the project. Especially Wiljan Vloet, managing director of the Roparun Foundation and Max van de Ven, for meeting us every week and providing us with feedback on the product.

*L. Overdevest*

*M. van Meerten*

*J. Nieuwdorp*

*Delft, January 2019*

# Contents

<b>1</b>	<b>Summary</b>	<b>1</b>
<b>2</b>	<b>Introduction</b>	<b>2</b>
<b>3</b>	<b>Research</b>	<b>3</b>
3.1	Problem definition . . . . .	3
3.2	Current approach . . . . .	4
3.3	Input and output . . . . .	5
3.4	Formalization . . . . .	5
3.4.1	Input formalized. . . . .	5
3.4.2	Output formalized. . . . .	5
3.4.3	Problem formalized. . . . .	6
3.5	Data analysis . . . . .	6
3.5.1	Route . . . . .	6
3.5.2	Speed statistics. . . . .	7
3.5.3	Request data . . . . .	9
3.6	Existing solutions . . . . .	10
3.6.1	Linear Programming . . . . .	10
3.6.2	Brute force algorithm . . . . .	10
3.6.3	Greedy algorithm . . . . .	10
3.6.4	Genetic algorithm. . . . .	10
3.6.5	Neural network . . . . .	11
<b>4</b>	<b>Development Strategy</b>	<b>12</b>
4.1	User Stories. . . . .	12
4.2	Requirements. . . . .	12
4.2.1	Must have. . . . .	12
4.2.2	Should have . . . . .	13
4.2.3	Could have . . . . .	13
4.2.4	Would/Won't have . . . . .	13
4.3	Technical choices. . . . .	13
4.3.1	Language . . . . .	13
4.3.2	Graphical user interface . . . . .	13
4.3.3	Web application or local application . . . . .	14
4.4	Product plan . . . . .	14
<b>5</b>	<b>Code design</b>	<b>16</b>
5.1	Code architecture. . . . .	16
5.1.1	Packages . . . . .	16
5.1.2	UML . . . . .	16
5.2	Design patterns. . . . .	17
5.3	Input & Output . . . . .	18
<b>6</b>	<b>Interface Design</b>	<b>19</b>
6.1	Design approach . . . . .	19
6.2	Program flow . . . . .	20
6.3	Input specification . . . . .	21

---

<b>7</b>	<b>Implementation</b>	<b>22</b>
7.1	Greedy . . . . .	22
7.1.1	Linear programming . . . . .	23
7.1.2	Speed prediction . . . . .	23
7.1.3	Final algorithm . . . . .	24
<b>8</b>	<b>Process</b>	<b>25</b>
8.1	Methodology . . . . .	25
8.1.1	Code quality . . . . .	26
8.1.2	Testing . . . . .	26
8.1.3	Stakeholders . . . . .	27
8.1.4	Meetings . . . . .	27
<b>9</b>	<b>Product evaluation</b>	<b>28</b>
9.1	Code evaluation . . . . .	28
9.1.1	SIG . . . . .	28
9.1.2	Test evaluation . . . . .	29
9.2	Requirements evaluation. . . . .	29
9.3	Solution quality . . . . .	30
<b>10</b>	<b>Conclusion</b>	<b>33</b>
<b>11</b>	<b>Discussion &amp; Recommendations</b>	<b>34</b>
11.1	Reflection . . . . .	34
11.2	Recommendations . . . . .	34
11.2.1	Technical . . . . .	34
11.2.2	Non-Technical . . . . .	35
	<b>Bibliography</b>	<b>36</b>
<b>A</b>	<b>Info sheet</b>	<b>38</b>
<b>B</b>	<b>Original Project Description</b>	<b>40</b>
B.1	Dutch . . . . .	40
B.2	English . . . . .	41
<b>C</b>	<b>SIG Evaluation</b>	<b>42</b>
C.1	First feedback. . . . .	42
C.2	Second feedback. . . . .	43
<b>D</b>	<b>User stories</b>	<b>44</b>
<b>E</b>	<b>Speed data</b>	<b>46</b>
<b>F</b>	<b>Class Diagrams</b>	<b>47</b>

# 1

## Summary

Roparun, a Foundation that organizes a yearly relay race from Paris, Hamburg and Almelo to Rotterdam to raise money for a variety of causes related to palliative care, requested for an application that could create the start schedule of the race. The goal of the application is to create a schedule based on multiple parameters.

In the research phase, several algorithms were analyzed which would be of value to solving the problem. Furthermore, a speed prediction model was created to get a more precise estimation of team's speeds, and in turn, the course of the race.

During development, a greedy algorithm was chosen as the core of the solution. And an application was iteratively designed to encompass this solution. Extra features were added to increase the value of the program to the organization. Modern, proven development methodologies were used to ensure a reliable and maintainable product was delivered within the project duration. To ensure this, and that the solution offered meets the clients needs, the application is validated in two ways. Automatic testing and manual testing. A final manual verification is done, where the schedule generated by the application is compared to the schedule used in the previous year. The results of the comparison can be found in this report. This shows that a method was developed to create a schedule that outperforms the previous method on various levels. However there are more improvements possible, both within the software product context as in the problem space of the real world. These aspects are discussed and recommendations are offered in section 11.

# 2

## Introduction

The Roparun foundation organizes a yearly relay race called The Roparun. This race was held for the first time in 1991. It is a relay race with starting locations in Paris and Hamburg, stretching over more than 500 kilometers with its finish in Rotterdam. People take part as teams in an athletic event to raise money for a variety of causes related to palliative care. Roparun is also called an adventure for life. This also becomes clear from the motto, which for years has been: 'Adding life to days, when days often can't be added to life'.

To let the race run smoothly the organization needs to take many different aspects into account. One of the most important aspects is the schedule for the race. Our task was to develop an application which automates and assists in this scheduling process. Over the course of ten weeks, we have developed an application which creates a schedule that considers many different parameters. An overview of some of the functionality that was implemented includes: uploading teams and route information, speed prediction based on historical data, incorporating participants' specific requests and various export options of the schedule. The final product is measured against the current method by creating a schedule for the past year. This way the schedule produced by both methods can be compared and evaluated. The result of this evaluation favors the schedule created by the application.

In this report all information can be found about the process of creating the application and the application itself. The results of the two weeks research phase are presented in chapter 3. The development strategy is elaborated in chapter 4. Chapters 5 and 6 contain the design choices with their justification. Then, in chapter 7, the implementation of various algorithms are elaborated upon. Chapter 8 describes the process during the project, containing information about the frameworks and methods used. Then the product is evaluated, followed by our conclusion, and finally recommendations for the organization are given.

# 3

## Research

The research phase is of great importance for developing a solution to the given problem. In this chapter the results of the research phase are presented. First the problem is defined, followed by the current approach the organization uses to solve the problem. Next the requirements of the solution and a way to measure the quality of the given solution are proposed. The input and output of the algorithm, and the problem are defined in a formal way. Following this, the key concepts of the problem are laid out. Next the historical data from previous runs is analyzed. After that, the existing solutions are analyzed and the technical choices made during the project are substantiated. Finally, a product plan is given.

### 3.1. Problem definition

Since the birth of Roparun in 1991, scheduling the start times of the teams has been a matter of intuition and later, using experience from past runs. However, as Roparun has continued to grow over the years, this method is no longer sufficient for guaranteeing the safety of the participants and the efficient departure and arrival of the teams. Therefore, the development team is tasked with building a scheduling tool that will assist the organizers in creating the schedule. The main function of the tool is to generate a list of start times and predicted finish times for each team, which ensures a timely and efficient start for all teams, and a finish where all teams arrive in short succession, within the allotted time frame, and without compromising safety and throughput throughout the run. Spread across the route are checkpoints where teams register as they pass. Based on this information the organizers would like to be updated on whether the intended schedule is being followed, and where potential bottlenecks could occur. In 2018, 309 teams participated in the Roparun, and for 2019, 332 teams have already enrolled. Starting in Hamburg and Paris, the teams run to Rotterdam (see figure 3.1). In Barendrecht the two different routes meet and merge into one route to Rotterdam. From 2019 on, Roparun will organize a half run from Almelo along the Hamburg route.





Figure 3.1: The route as seen on the map

A team consists of 6-8 runners, and a number of people for support; e.g., cyclists, drivers, caterers and a team captain. The average size of a team is 25 people. Below are the variables that the tool should be able to take into account, weighing their importance:

- Average running speed of a team
- Different routes that finish at the same location (starting points in Paris, Hamburg and Almelo)
- Decreases in flow rate due to congestion and crowds
- Requests made by teams for specific start passage or finish times.
- Reduced speeds in certain cities being visited

### 3.2. Current approach

Currently the organization makes a starting schedule based on insight and experience. The teams provide their expected average speed, which according to the rules should be between 11 and 15 km/h (Roparun, 2018, p.8). For the half Roparun, teams are allowed to have an average speed between 10 and 15 km/h, since this is intended for less experienced runners. They will start ahead of the group coming from Hamburg. Since they run along the same route they are expected to be overtaken. The teams start in ascending order of speed. They try to let the teams from Paris and Hamburg meet around the same time in Barendrecht, so they will finish together. To spread out the teams evenly over the finish time frame, the organization plans for the opposite: congestion for runners at the beginning of the finish time frame. Experience taught the organization that if they plan for this congestion, it will not happen. So actually planning for chaos will result in an organized result in the current system. After this schedule is made, teams can submit their requests by email. Then the organization will try to fulfill

the requests by manually changing the starting order. Teams know whether their request is accepted by looking at the final starting schedule.

### 3.3. Input and output

The data used by the program needs to be uploaded by the user. To create the schedule, the application needs the following data: a list of teams, a list of routes and a list of requests. A team consists of a team number, their average running speed and the route they run. A route contains a distance, locations of the checkpoints and a flow rate due to congestion and crowds. The last input to our application is the submitted requests list. A request consists of a team number associated with the team making the request, the type of request and several optional values related to the type of request. These values are: requested start time, requested finish time, requested stop time frame, distance to stop location and combined stop/start team.

The output of the application contains the start time and the predicted finish time for each team.

### 3.4. Formalization

The input, output and problem are formalized in the following subsections.

#### 3.4.1. Input formalized

The input formalized:

- A set of teams  $T = \{t_1, \dots, t_n\}$
- A team can be formalized using:
  - *id* The team number
  - *name* The team name
  - *v* the specified average running speed
  - *r* the route this team follows
- A set of routes  $R = \{r_1, \dots, r_n\}$
- A route consists of:
  - An ordered set of starting points of intervals  $I = \{i_1, \dots, i_n\}$
- A set of requests  $Q = \{q_1, \dots, q_n\}$
- A request consists of:
  - *id* the team number that made the request
  - *type* the type of request
  - *start* the start time [optional]
  - *stop* the stop time frame [optional]
  - *distance* the distance until the planned stop [optional]

#### 3.4.2. Output formalized

The output formalized:

- A set of times  $E = \{e_1, \dots, e_n\}$ , where  $e_i = \langle start_i, end_i \rangle$  with  $start_i$  the start time of team  $i$  and  $end_i$  the predicted end time of team  $i$

### 3.4.3. Problem formalized

The problem can be described as a job shop scheduling problem, where a route is a list of  $n$  machines and a team traversing the route is a job. Each job consists of  $n$  operations, each team needs to traverse the  $n$  intervals of the route. Each operation  $O_i$  needs to be performed on machine  $M_i$  and the operations need to be completed in order; i.e.,  $O_{i+1}$  can only be started after  $O_i$  has been completed. A job can never wait for a machine. A machine  $i$  has a capacity  $c_i$ . The objective is to minimize the operational time of  $M_1$  and  $M_n$ .

The problem can be modeled as a linear programming problem. The decision variables are:

$$\begin{aligned}
 & t_{0,i} : \text{Start time of team } i \text{ in minutes} \\
 & t_{j+1,i} = t_{j,i} + \frac{d_{j,j+1}}{v_{j,i}}, \text{ the predicted time in minutes when team } i \text{ is at interval } j + 1 \\
 & T_i = t_{n,i} - t_{0,i}, \text{ where } n \text{ is the finish interval} \\
 & x_{j,i,k} = \begin{cases} 1 & \text{if } t_{j,i} \leq k \leq t_{j+1,i} \\ 0 & \text{otherwise} \end{cases} \\
 & L_{j,k} = \sum_i x_{j,i,k}
 \end{aligned} \tag{3.1}$$

The objective function:

$$\text{Minimize } \max(\{L_{n,tb}, \dots, L_{n,te}\}), \text{ where } n \text{ is the finish interval,} \tag{3.2}$$

The constraints are:

$$\begin{aligned}
 & C_j : \text{The capacity of interval } j \\
 & \forall_k \forall_j L_{j,k} \leq C_j \\
 & tb : \text{Opening time of the finish} \\
 & te : \text{Closing time of the finish} \\
 & L_{n,k} = 0, \text{ if } tb \geq k \geq te \\
 & tb = \max(T_i) \text{ for all } i
 \end{aligned} \tag{3.3}$$

## 3.5. Data analysis

Roparun has saved all data from the 2017 and 2018 runs, including but not limited to: starttime and finishtime of the teams, predicted finiahtime of the teams, the specified average speed of a team, at what time a team passed specific checkpoints. In this section a summary of the analyzed data is given.

### 3.5.1. Route

All information about the route is saved in multiple files. There is the route description, this is a spreadsheet where the route is divided into small intervals to assist the teams in navigating. Furthermore, it contains per interval information about: the distance traveled in meters since the last interval, the total distance traveled, checkpoints, what kinds of vehicles are allowed to drive at what location and information about toilets. This file does not have a visual representation of the route since only the distance between points is specified and not the location of the points. In Figure 3.2 a small part of the route description is shown.

335	43,6	b	C		FONTAINE-CHAALIS <b>D126</b> (RunBikeRun: 0.7 km)
195	43,8			↑	STOP -kruising <b>recht</b> door
245	44,1				ROPARUN Slaapplek tbv organisatie FONTAINE-CHAALIS
0	44,1				ROPARUN Checkpoint 02 FONTAINE-CHAALIS
175	44,2				Einde FONTAINE-CHAALIS <b>D126</b>
35	44,3				LET OP!!  Gevaarlijke kruising
5	44,3	b	C	↑	STOP -kruising <b>recht</b> door richting MONTEPILLOY
20	44,3		c		ROPARUN Toilet FONTAINE-CHAALIS

Figure 3.2: Snippet of the route description file

For this purpose other files are supplied. A KMZ file which is a (compressed) list of all coordinates that form the route (both for runners and for support vehicles). The KMZ file is intended to be loaded into Google Earth (Google, 2019a). There also is a URL available where the route is laid out in Google Maps (Google, 2019b), this includes not only the route but also markers for the checkpoints. Finally, the organization provides the route in a format usable on navigation devices by two major brands: TomTom and Garmin. These contain the same route data as the other files, enriched with various extra locations (points of interest). These will not be used for the product as they are heavily dependent on proprietary file systems and the added value of the data enrichment for our purposes is minimal.

The route description will be used to load the route into our tool as this is the leading document according to Roparun.

### 3.5.2. Speed statistics

In order for the tool to give a reliable estimation of the course of the race, it is important to get an accurate speed of the teams. Currently, the only available metric is the average speed of teams. This speed is specified by the team itself. Since it is an estimation, the reliability of this estimation has been an issue (rounding, over-estimation, inexperience, composition of teams, incidents). To improve this important information, using historic data will be looked into. Historic data is of great value. In the paper by (Chien and Kuchipudi, 2003) historic data is used to predict travel times in multiple transportation networks. Answering the following questions will be attempted by analyzing the data:

- Is there a general trend in deviation from the projected average speed?
- At what time was the team under / over the average speed?
- Are there other discernible trends that emerge from the data (e.g. do teams that specify a certain speed perform differently compared to teams that have other specified speeds)

Teams often participate multiple years in a row, but as teams are usually bound to an organization or community, these teams change composition every year. Therefore, data from a team that participated before cannot be used to more accurately predict that specific team's actual speed. Also the organization still sees a growth in participating teams, for which no information is present.

The data that will be attempted to analyze was gathered during the 2017 and 2018 runs. At every checkpoint, the time a team passed that checkpoint was recorded. From this data it will be possible to derive the average speed of a particular team between any two checkpoints (including start and finish). This in turn will allow creating a dataset containing, for each specified speed, at each checkpoint, a set of actual speeds. Table 3.1 shows a snippet of the original data.

The data from table 3.1 is first filtered by the columns 'FromAct' and 'Valid'. A '1' in the 'FromAct' column signifies the time is recorded by the organization, and not by the team itself. A '1' in the 'Valid' column signifies the recorded time has been verified as a correct time. This data is then used to calculate the average speed between checkpoints as shown in table 3.2

The data after filtering on the columns still contains impossible and implausible values such as negative speed, or speeds of over 20 km/h. This is most likely caused by human error, as the data was recorded manually. After filtering the data of these impossible values, the data is grouped into

ID	Positie	Tijdstip	FromAct	Valid	IsRegistered	Team
1273	0	5/19/18 19:54	1	1	0	1
1633	441	5/19/18 23:28	1	1	0	1
2202	863	5/20/18 3:03	1	1	0	1
2765	1307	5/20/18 6:35	1	1	0	1
3360	1744	5/20/18 10:07	1	1	0	1
3949	2156	5/20/18 13:33	1	1	0	1

Table 3.1: The original checkpoint-time data

ID	Positie	Tijdstip	FromAct	Valid	Registr.	Team	Distance	Speed
1273	0	5/19/18 19:54	1	1	0	1	0	12000
1633	441	5/19/18 23:28	1	1	0	1	44,1	12318,435
2202	863	5/20/18 3:03	1	1	0	1	86,3	11799,611
2765	1307	5/20/18 6:35	1	1	0	1	130,7	12537,459
3360	1744	5/20/18 10:07	1	1	0	1	174,4	12411,834
3949	2156	5/20/18 13:33	1	1	0	1	215,6	11992,238

Table 3.2: The calculated speed data

Paris and Hamburg runs according to checkpoint distances, and then grouped into average speed per specified speed, per distance as shown in figure E.1.

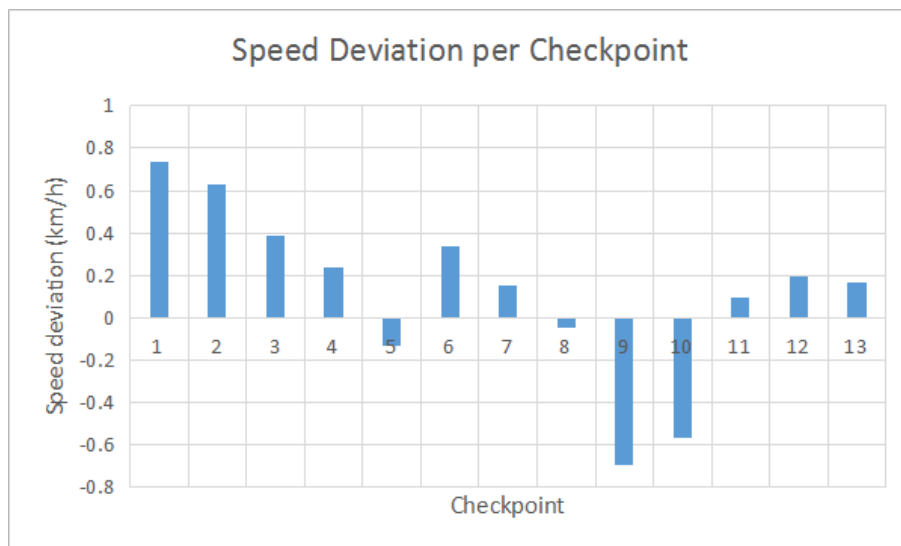


Figure 3.3: The average speed deviation per checkpoint from Paris 2018

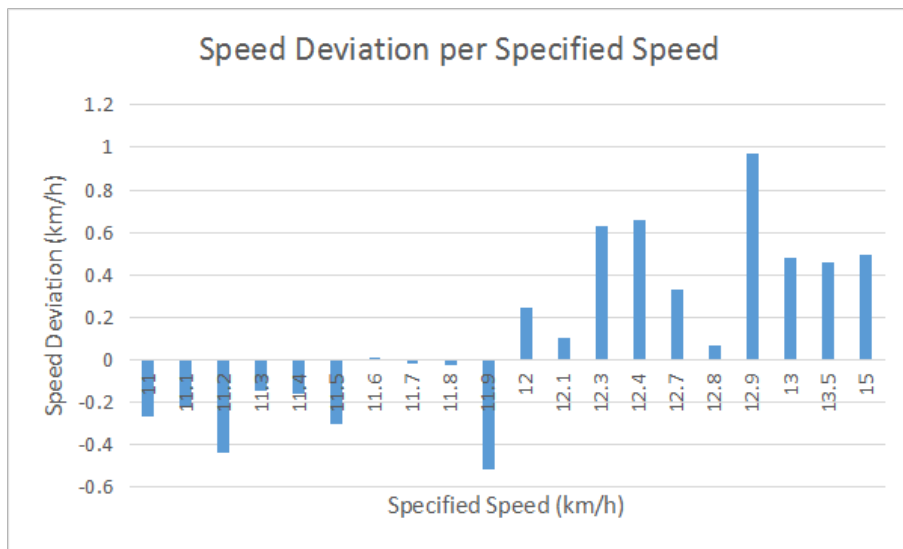


Figure 3.4: The average speed deviation per specified speed from Paris 2018

The speed data does not have a general trend in deviation from the projected average speed to which every team adheres. There is, however, a deviation dependent on time or distance. (See figure 3.3) Teams tend to start quickly, slow down during the night and speed up again in the morning. There is also a trend related to specified speeds. A team with low specified speed will likely have an even lower actual speed, while a team with a high specified speed will likely have an even higher actual speed. (See figure 3.4)

### 3.5.3. Request data

A large number of teams submit a request regarding the schedule. The requests vary from starting at a specific time to taking a break at particular location. Last year, 72 of the 309 teams proposed such a request. The tool must take these requests into account. In order to effectively process these, the requests of previous years are analyzed and categorized.

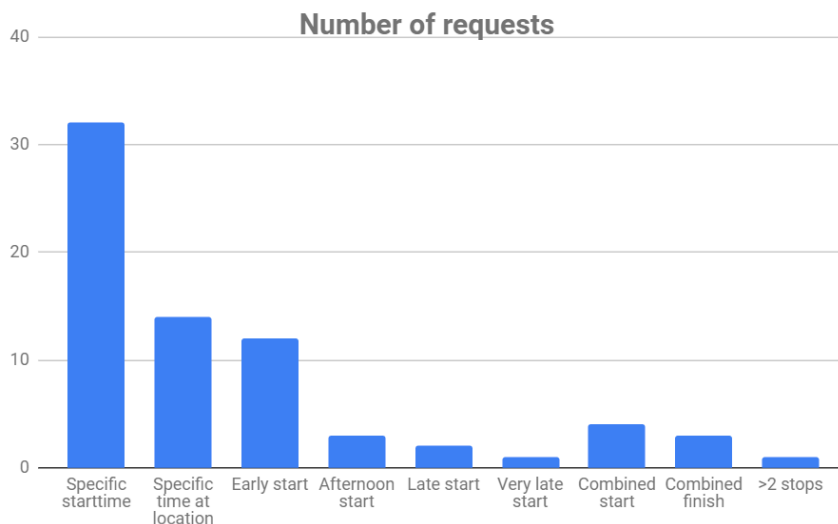


Figure 3.5: Statistics about the requests

In figure 3.5, a histogram of the categorized requests from 2018 is shown. Most teams want to start at a specific time and many teams want to stop at a specific time at a location. The "specific time

at location” requests can be translated to “specific start time” requests, by using the team’s predicted speed and the distance to the location. Similarly for the requests: early start, afternoon start and late start. Except that these start times lay within a specific range. The combined start request is more difficult as this introduces a dependency between two teams. A combined finish also causes this dependency.

## 3.6. Existing solutions

In this section, the approaches that could provide a solution to our problem will be analyzed. Starting with a subsection about the chosen solution and in the following subsections, elaborating on alternative solutions and why they do not solve our problem. The existing solutions found in literature regarding related problems were minimal. Information about general schedule systems were found in abundance. Our problem, however, does not apply to a general scheduling problem because of the constraints.

### 3.6.1. Linear Programming

It is expected that our problem is solvable using linear programming. M. Goemans from MIT defined linear programming as follows:

”Linear Programming deals with the problem of optimizing a linear objective function subject to linear equality and inequality constraints on the decision variables” (Goemans, 2015)

In section 3.4.3, the constraints and decision variables of our problem are defined. The output of the algorithm will be an optimal schedule which satisfies all the constraints. In the next subsections it will be discussed why other alternatives do not apply to our problem.

### 3.6.2. Brute force algorithm

One way of solving a schedule problem is looking at all possible schedule possibilities and choosing the optimal solution. This method runs in  $O(n!)$  time. Given the fact that there are 330 teams starting, there will be 330! possible solutions. This will obviously cause problems in terms of time complexity. (Kleinberg and Tardos, 2006, p.31-32)

### 3.6.3. Greedy algorithm

A well known algorithm to create optimal schedules is the greedy algorithm (Kleinberg and Tardos, 2006, p. 116-207). The basic concept of greedy algorithms is that the solution found by the algorithms is built up in small steps. At each step, the algorithm optimizes the solution for a specific criterion. Since all the intervals of the route must be traversed in the same order, the amount of choices the greedy algorithm has to make is minimal. Furthermore, the average speed of the teams has a huge impact on the final schedule. The optimal solution can be found by letting the teams start in increasing order of average speed. The reason a greedy algorithm is not considered optimal, is that the problem also specifies constraints that cannot be effectively taken into account by the algorithm, such as the prevention of bottlenecks in traffic flow. Limiting bottlenecks could lead to an endless switching around of starting times. A greedy algorithm is therefore not the optimal type of algorithm to solve the problem as defined.

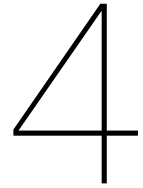
### 3.6.4. Genetic algorithm

As mentioned in the last paragraph of subsection 3.4.1 the problem can be described as the job shop scheduling problem. In (Pezzella et al., 2008) a method is discussed how the job shop scheduling problem can be solved using a genetic algorithm. The only major difference between our problem and the problems solvable by the genetic algorithm is that, in job shop, the jobs have to wait when the machine reached their capacity. In our problem, this is obviously not the case. If the maximum amount of runners at an interval is reached the average speed of a runner decreases rather than the runner coming to a complete stop until the amount of runners is below the capacity of that interval. This means an extra constraint is added: if the maximum capacity is reached the operation speed (average speed) of a job (runner) must decrease. This makes the job shop scheduling unsuitable to this genetic algorithm.

**3.6.5. Neural network**

In (Sabuncuoglu and Gurgun, 1996) a method is discussed how the job shop scheduling can be solved using neural networks. However, as elaborated in the subsection above, our problem does not fit with the job shop scheduling.





# Development Strategy

This section will highlight strategical choices that were made during the research phase of the project that provided the blueprint for the rest of the development process. First an outline will be given on how the requirements were formulated and prioritized, followed by a justification of technical design and environmental choices. Finally the initial product plan that was formulated at the end of the research phase is presented.

## 4.1. User Stories

In order to describe desired functionality in a way that the requirements of all different stakeholders will be thought out and visible from the start user stories will be utilized (Cohn, 2004).

In accordance with the model proposed by Ron Jeffries the user stories are written on cards. (Jeffries, 2001) For the card a standard template has been used:

As a <Role >I want to <Capability >so that <Benefit/Goal >

After a user story card is written, a conversation with the client follows. This includes all interactions the different stakeholders have concerning a specific user story and does not have a physical form. In order to document this step a brief summary of the conversation is written.

Finally every user story receives confirmation, which means it will be translated into functionality that is to be implemented.

For the sake of brevity all roles are reduced to keywords, with a longer description.

User – The roparun employee that will use the final product (end user).

Participant – A member of a participating team.

Roparun – The roparun organisation as a whole entity.

Developer – A member of the development team.

The user stories are defined in appendix D.

## 4.2. Requirements

During a project it is important to know what the requirements set by the customer are, and how they prioritize the requirements. The initial requirements came from the user stories (see appendix D) and where prioritized together with the client. A commonly used method that aides in prioritizing requirements is the MoSCoW method (Ghahrai, 2017). This method helps the client and project manager understand the importance of the requirements. If the importance is known, the requirements can be prioritized and developed in that specific order. In the following subsections the requirements are categorized according to the MoSCoW method.

### 4.2.1. Must have

- The tool must schedule a start time for each team.

- The tool must predict the finish time for each team.
- All predicted finish times must be within the time frame stated by the government permit.
- The tool must use historical data to adjust the specified speed to a more realistic speed.
- The tool must be effectively usable by the client.

#### **4.2.2. Should have**

- The tool should be able to handle proposed requests.
- The tool should give an insight in the optimal opening times of the checkpoints along the route.
- The tool should be able to limit the amount of traffic on the route based on known bottlenecks.
- The tool should allow the organization to manually change the schedule and adjust accordingly.
- The tool should be durable.
- The tool should be compatible with all operating systems.

#### **4.2.3. Could have**

- The tool could have a feature to visualize the density per segment per time frame of the race.
- The tool could update the predicted finish time by using the team's checkpoint times.
- The data of following years could be used to improve the performance of the tool.
- The tool could provide a list of fulfilled and denied requests.
- The tool could have a formatted pdf export for the schedule.

#### **4.2.4. Would/Won't have**

- Automation of checkpoints and start.

### **4.3. Technical choices**

Below various technical choices are substantiated with regards to choice of language, goals for the user experience and platform dependencies of the program.

#### **4.3.1. Language**

Java provides a flexible basis that allows for implementation of generally all concepts will be encountered in this project. Execution speed is not a direct requirement, so the ability of Java applications to run on a wide variety of hardware will offset the performance disadvantages Java has over over programming languages. The object oriented nature of Java is very suitable for the iterative work flow. Finally, Java is platform-independent (Arnold et al., 2005), so the Java application can be run on all operating systems.

#### **4.3.2. Graphical user interface**

In this section GUI libraries for Java are compared. The most used libraries to visualize the content of Java applications are Swing and JavaFX. The latter is seen as the successor of the outdated Swing (Meidinger et al., 2015, p.526). General major advantages of JavaFX are that it is has a powerful runtime and CSS styling can be used (Premkumar and Mohan, 2010). Compared to Swing it is also more consistent in event handling and code production (Lowe, 2016). For these reasons JavaFX was chosen to be used in our project.

### 4.3.3. Web application or local application

Currently the client has specified a preference for a local application, however, some lower priority requirements would benefit from a web application. If the project evolves to a web application with Java it is helpful to use a framework. This will abstract a lot of the details of building web applications and make developing more efficient. The framework then planned to use is Spring MVC. Spring MVC is chosen for a multiplicity of reasons; it is the most used framework, with a very complete and extensive documentation. Furthermore, it enforces a strict distinction of controllers, models and views and provides various templates to interact with databases (Pandey et al., 2018, p.4). Despite this, Spring MVC has a steep learning curve, so it takes some time to learn the basics of Spring MVC for team members not yet familiar with Spring MVC.

## 4.4. Product plan

- Week 1
  - Research report
    - ◊ Research problem definition
    - ◊ Request data analysis
- Week 2
  - Research report
    - ◊ Requirements specification
    - ◊ Existing solution analysis
    - ◊ Speed data analysis
  - UML design
- Week 3 (sprint 1)
  - Setup development tools
    - ◊ Define workflow
    - ◊ Configure version control system
    - ◊ Configure test environment
  - Create barebones application
- Week 4 (sprint 2)
  - Data import
  - First rudimentary schedule (greedy)
- Week 5 (sprint 3)
  - Model for speed prediction
  - First linear programming schedule model
- Week 6 (sprint 4)
  - Improve linear programming schedule model
  - improve Model for speed prediction
  - Handle proposed requests
  - Create model for testing solution quality
- Week 7 (sprint 5)
  - Implement SIG improvements
  - Improve linear programming schedule model

- 
- Insight in opening times of checkpoints
  - Manual adjustments to schedule
  - Week 8 (sprint 6)
    - Limit the amount of traffic on certain parts of the route
    - Update the predicted finish time by using the team's checkpoint time
  - Week 9 (sprint 7)
    - Extend the tool to update data for following years
  - Week 10 (sprint 8)
    - Export schedule as pdf
    - Preparing presentation

# 5

## Code design

In this section all code design choices are elaborated upon. First the general architecture of the code is given. Then the design patterns used are discussed.

### 5.1. Code architecture

In this section, the code architecture is elaborated. First the structure of the packages is explained. Then the dependencies between the packages are visualized using UML.

#### 5.1.1. Packages

The code base is divided into six packages: model, view, controller, core, validator and io. The model package contains the data classes (e.g. Route, Team, Request). The controller package contains the classes that interact between the model and the view package. Most of the program's logic is contained in the controller package. The view package contains the classes that produces the user interface of the application. The core package contains the classes that provide the logic not captured in the controller package. For example: the logic to generate the scheme, speed prediction over specific intervals, and sorting the teams and schedule based on specific elements. The validator package contains all classes that validate input. The final package, io, contains all classes that parse the input or export the result.

#### 5.1.2. UML

In this section the architecture of the code is visualized via UML diagrams, and the dependencies between packages are described and substantiated.

The RopaSchedule class is the primary class (See figure 5.1) that contains the main method of the application. From here an instance of the application is loaded, the JavaFX stage is created and the GUIController class in the controller package is created. This class is responsible for program flow and all other controllers are created from this class. Every screen has a unique controller which is responsible for showing a correct and updated view. The user input into the view is validated by a designated input validator class from the validator package, which is then returned to the controller. These controllers create and modify the required model objects, and call the required core classes for schedule creation, speed prediction and other processes that should be accessible to both the controller and model.

A complete overview of the classes can be found in appendix F.

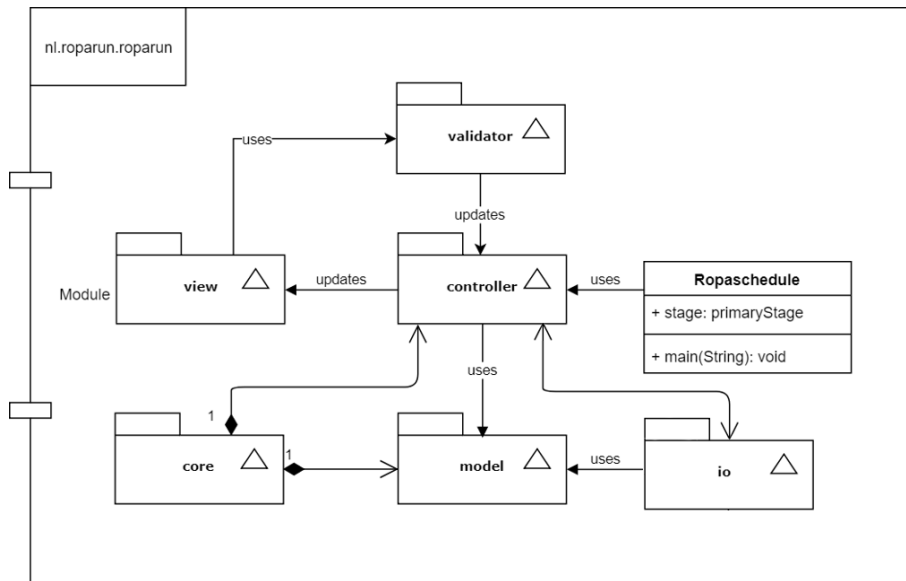


Figure 5.1: The package dependency diagram

## 5.2. Design patterns

Design patterns are used to solve common software engineering problems. (Gamma et al., 1994). This section will highlight the primary design patterns that have been used in the software.

### MVC

One of the design patterns used is the Model-View-Controller (MVC) pattern (Atwood, 2018). This design pattern is based on three other design patterns: Composite, Strategy and Model. MVC divides the software application into three interconnected parts: model, controller and view. The model specifies the structure of the data. The view shows the output to the user. To create the views, JavaFX is used (Clarke et al., 2009). CSS is used to style the graphical interface created by JavaFX. The controller acts as an interface between the model and the view. Most logic is included in the controller. The advantages of this design pattern are briefly discussed. Using this pattern, allows developers to program in parallel. This increases the speed of the development process. Furthermore, MVC enables developers to create multiple views for the same model. Duplication of code is therefore limited since the logic of the application is separate from the GUI.

### Singleton

Singleton is another design pattern that is implemented. The singleton design pattern restricts the instantiation of a class to at most one instance. (Gamma et al., 1994) The SpeedCalculator class is implemented in this way. This is done for the following reason: if there are multiple Speedcalculator instances in the program the calculations for the speed prediction could be mixed up. This would result in inconsistent and unreliable schedules.

### Observer

The observer pattern is a way for observer objects to observe the state of an observable object without being coupled. (Gamma et al., 1994) JavaFX uses this pattern in the form of observable lists that contain data displayed in a table. Updating the list will therefore also update the table. This pattern was also used in decoupling the GUIController class from the other controllers.

### Strategy

The strategy pattern is used to define a family of algorithms, encapsulate each one, and make them interchangeable. This lets the algorithm vary independently from clients that use it. (Gamma et al., 1994) This pattern was implemented in the Parser and Exporter classes. The Parser will be required to

call different methods for parsing route description files and team files. The Exporter classes differ in the way they fill the cells. The scheme, event info, and checkpoint exporter all implement this method differently.

## 5.3. Input & Output

This section describes the design of the input and output code: parsing, exporting and saving data.

### Read & write

The organization saves all data about the teams and routes in spreadsheets. Therefore Apache POI (Apache, 2018) is used. Apache POI is a Java library for reading and writing files in Microsoft Office formats, to which Excel belongs. The data used in our application, is read and written using Apache POI. In the previous years, the schedule was provided to the teams in a styled document with PDF format. Therefore, the same PDF files are able to be generated. The PDF contains the logo of the organization and the rows of the table are colored depending on the start bucket a team starts in. To export the schedule to PDF, itextpdf (Ittextpdf, 2019) is used. Ittextpdf is a Java library for generating PDFs.

### Saving

When the program is closed the user input is saved using Java serialization (Oracle, 2019). The serialized objects are written onto the disk in session files, which are read when the application is restarted to recreate the objects in memory. The main reason why saving functionality is implemented is usability of the application. The 2018 edition had 73 requests. Being able to close the program and take a break is a substantial improvement to the usability of the application. Moreover, this also enables the user to make last minute changes.

### Logging

Loggers are a fundamental part of software applications. Loggers are used to log the actions performed during run-time. This allows for quicker debugging, as the location of errors or failures is also logged. It could also provide information about the application (e.g. amount of teams parsed). An example log by the application is: "The schedule is successfully exported".

# 6

## Interface Design

One of the requirements discussed in 4.2 states that the tool must be effectively usable by the client. Therefore, operating the tool should be as simple and straightforward as possible. To achieve this, the user interface is built so that the user always knows what to do. The objective is that a user, with minimal prior knowledge of the task they would like to complete, is able to do so without additional instructions. How the user interface achieves this objective is described in section 6.1. Next, different views of the tool are discussed and the associated functionality of the associated steps are elaborated.

### 6.1. Design approach

Initially a broad idea of the general structure of the user interface was required in order to develop a consistent and smooth look and feel for the application that meets the prior given criteria. To achieve this early on in the process each team member created a mock up of their ideas for the design. These drafts were discussed and the best aspects of each were combined in a final product vision. Consistency and clarity were the most important factors when making design choices.

In figure 6.1 a first version of the user interface is shown.

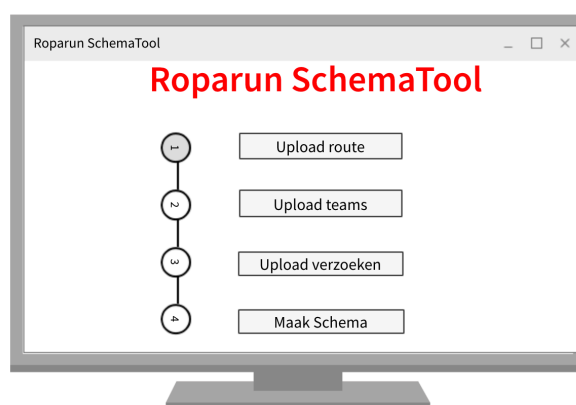


Figure 6.1: Draft of the GUI of the start screen

Stylistically the goal was to apply the corporate identity of the Roparun and make the application look modern. Most notably instances of the corporate identity are the colors used in the color scheme, logos through the application and font selection.



## 6.2. Program flow

When the application is started it checks to see if previous sessions are available. If that is the case the user is prompted to either continue with their previous session or create a new schedule. If the user selects the option, new schedule, a new screen is shown to the user. This screen can be seen as the main application, since it displays all elements that are consistent across the process. The style of the application is the same for each screen.

The screenshot shows the 'Roparun Scheduler' application window. It is divided into two main sections: '1. Vul evenement gegevens in' and '2. Voeg runs toe aan evenement'. The first section contains input fields for 'Selecteer een finish datum:' (23-1-2019), 'Selecteer openingstijd finish:' (14:19), 'Selecteer sluitingstijd finish:' (17:19), and 'Tijdsduur van tijdregistratie tot finish:' (30). The second section contains input fields for 'Titel:' (Hamburg), 'Selecteer een teamlijst:' (RrPa18 - Routerol (Internationaal).xlsx), 'Selecteer een routerol:' (empty), and 'Selecteer snelheidsmodel:' (Parijs). Below these sections is a table with columns: ID, Titel, Teamlijst, Routerol, Snelheidsmodel, and Verwijder. The table contains one row with ID 0, Titel 'parijs', Teamlijst 'RrPa18 - Passage St...', Routerol 'RrPa18 - Routerol.xlsx', and Snelheidsmodel 'Parijs'. At the bottom of the window, there is a step navigator with four steps: 'Bestanden Laden', 'Verzoeken Toevoegen', 'Wijzigen', and 'Exporteren'. A 'Volgende' button is located to the right of the step navigator.

Figure 6.2: Final version of the first stage

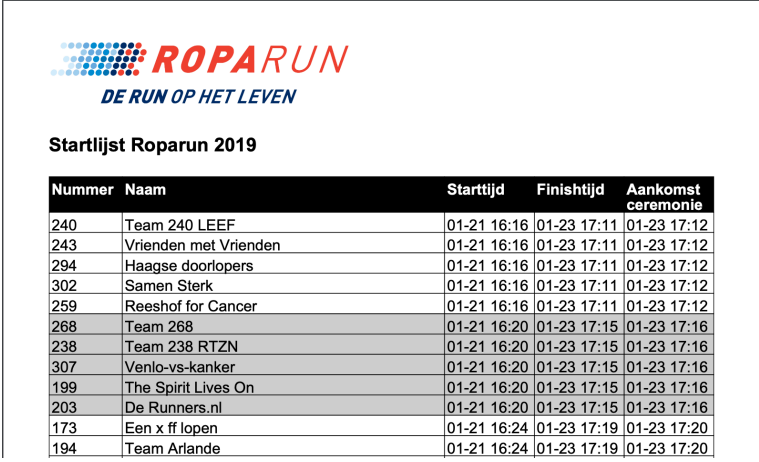
On each screen, the user is guided by a step navigator which can be seen on the bottom in figure 6.2. This ensures that the user always executes the required steps in the correct order. The user always knows in what phase of the application they are in and what is still to come. Every screen also has, where applicable, a 'volgende' and 'vorige' button that allows the user to progress, to the next or previous screen respectively. This creates a procedural approach and allows the program to perform tasks in the background while the user supplies more input. This greatly improves the perceived performance. When the user goes to the next page the input given is validated. If information is missing or incorrect, the user is notified. The input element's border will color red, and red text will appear to the right side of the element explaining what went wrong. When the fields are validated correctly the program will advance to the next screen. When the user presses the previous button they will receive an alert warning them that going to an earlier stage will discard any changes made in the current stage, when this is applicable. If the user confirms their decision, or no changes would be discarded the program goes back to the previous screen.

The decision to discard changes when navigating to a previous stage in the program was made to ensure consistency of the data. For example, if the user adds two runs, then adds a request for a team of run 1, and then removes this run. There will now be a request for a non-existent team. This is prevented by discarding the requests after going to the previous stage.

When the user arrives at the final screen all previous steps have been completed correctly and the generated scheme will be displayed in a table. The user has various options for exporting the schedule and related event information.

There are two types of documents that can be exported from the program, spreadsheets and PDF files. The former is unstyled and the latter conforms to the corporate identity of Roparun. The PDF

export is available for both the start and finish schedules of the organized event. An example of such a document can be seen in figure 6.3. It is formatted in a way that is immediately usable by the Roparun to redistribute to the teams and combines clarity and a high information density with the font, logo and styling in line with other documents used in the organization.



Nummer	Naam	Starttijd	Finishtijd	Aankomst ceremonie
240	Team 240 LEEF	01-21 16:16	01-23 17:11	01-23 17:12
243	Vrienden met Vrienden	01-21 16:16	01-23 17:11	01-23 17:12
294	Haagse doorlopers	01-21 16:16	01-23 17:11	01-23 17:12
302	Samen Sterk	01-21 16:16	01-23 17:11	01-23 17:12
259	Reeshof for Cancer	01-21 16:16	01-23 17:11	01-23 17:12
268	Team 268	01-21 16:20	01-23 17:15	01-23 17:16
238	Team 238 RTZN	01-21 16:20	01-23 17:15	01-23 17:16
307	Vento-vs-kanker	01-21 16:20	01-23 17:15	01-23 17:16
199	The Spirit Lives On	01-21 16:20	01-23 17:15	01-23 17:16
203	De Runners.nl	01-21 16:20	01-23 17:15	01-23 17:16
173	Een x ff lopen	01-21 16:24	01-23 17:19	01-23 17:20
194	Team Arlande	01-21 16:24	01-23 17:19	01-23 17:20

Figure 6.3: The schedule exported as a pdf file

## 6.3. Input specification

In the program, the input required from the user is mostly obtained from GUI elements, which have validation on the values to ensure the correct values are entered. During the creation of a schedule, the program also requires two files as input. These files contain the description of the route and the list of teams that participate. Both files are required for every run, in the following paragraphs the specifications for both files are given.

### Route file

The specifications for the route are based on a file structure that is already present within the roparun, the so called "Routerol" (see section 3.5.1). This means that the input file should be a Microsoft Excel Workbook file with file extension ".xlsx". The first row is reserved for headers and will not be read by the program. Every following row is interpreted as a point on the route, an entry. Column "A" should contain the distance to the previous entry in meters, formatted as a whole number. Column "B" should contain the distance from the start in kilometers formatted as a number with one decimal place. Column "G" represents a checkpoint, it should start with "ROPARUN Checkpoint " followed by the number of the checkpoint and a description of the checkpoint in one word separated by a white space. The format of the column should be general, any styling of the document is not used.

### Team file

The input file should be a Microsoft Excel Workbook file with file extension ".xlsx". The first row is reserved for headers and will not be read by the program. Every following row represents a team. Column "B" should contain the number of a team formatted as a whole number. Column "C" should contain the name of a team formatted as text. Column "D" should contain the specified speed of a team formatted as a number with one decimal place. Any styling of the document is not used.

# 7

## Implementation

In this section, the process of the implementation of the scheduling and speed prediction algorithms is elaborated. The scheduling is performed by a greedy algorithm.

### 7.1. Greedy

To be able to show the client a valid schedule as soon as possible, a simple greedy algorithm was implemented. Below, the pseudocode of the algorithm can be found. The algorithm only satisfies the following requirements: all predicted finish times must be within the time frame stated by the government permit and the teams must finish evenly spread over the time frame. The algorithm does not take any other requirements into account. For example: the maximum amount of runners allowed per segment and the requests.

The algorithm works as follows, first the teams are sorted in decreasing order of specified speed. The amount of time between the finish time of the teams is computed by dividing the amount of hours between the opening and closing time of the finish by the amount of teams, say `timeBetweenTeams`. This results in an even spread at the finish, each team finishes a constant and identical time after the last team. The starttime of a team is computed using the distance of the race and the specified speed. In this schedule, the fastest team is planned to finish at the opening time of the finish. The starttime is computed by subtracting the amount of time a team needs to run the race from the opening time. The next team needs to finish at the opening time plus the `timeBetweenTeams` such that a spread at the finish is achieved. This process is executed until a starttime is assigned to each team.

---

**Algorithm 1** Greedy schedule algorithm

---

```
Let teams be a List of teams sorted on decreasing specified speed, so  $v_1 \geq v_2 \geq \dots \geq v_n$ 
schedule  $\leftarrow$  Map that contains as key, a team, and as value, a Pair of start and finish time
finishOpeningTime  $\leftarrow$  Opening time of the finish
finishClosingTime  $\leftarrow$  Closing time of the finish
timeBetweenTeams  $\leftarrow$  (finishClosingTime - finishOpeningTime) / teams.size()
distance  $\leftarrow$  distance of the route
for (Team curTeam : teams) do
    timeRan  $\leftarrow$  distance /  $v_{curTeam}$ 
    startTime  $\leftarrow$  finishOpeningTime - timeRan
    add curTeam and Pair(startTime, finishOpeningTime) to schedule
    finishOpeningtime  $\leftarrow$  finishOpeningtime + timeBetweenTeams
end for
return schedule
```

---

### 7.1.1. Linear programming

Contrary to our expectation at the beginning of the project, a linear programming algorithm was not implemented. There are several reasons for this. First and foremost, during the course of the project, the client reported a change in priorities on their side. After signals received from stakeholders, the client decided that avoiding bottlenecks due to high density traffic was no longer an important concern. This in turn altered the problem to where a linear programming algorithm would not give a better solution than an advanced greedy algorithm, since the optimization of the traffic along the route was taken out of the equation. Secondly, implementing the LP algorithm took more time than anticipated. The amount of work required to get a simplified version of the problem to yield a solution greatly exceeded the estimation of the entire problem. This was partly due to the inexperience with the technology. When the priorities changed the best approach was reassessed and the linear programming approach was not only essential anymore, it was also unsubstantiated whether it would yield the best results at that point.

### 7.1.2. Speed prediction

As analyzed in 3.5.2, the speed specified by the teams is not a very accurate representations of their actual speed. The historical data analysis shows that teams with a lower specified speed are more likely to run slower than their specified speed, and teams with a higher specified speed are more likely to run faster than their specified speed. In order to utilize this information in the tool, a spreadsheet containing the actual mean speeds per distance per specified speed as shown in figure E.1 is used as a lookup table in the application.

The method `findLookupSpeed(oldSpeed)` returns the closest speed to `oldSpeed` for which there are entries in the table. `findLookupDistance(oldDistance)` returns the first distance rounded up from `oldDistance` for which there are entries in the table. `lookup(ISpeed, IDistance)` returns the speed value found in the table at `ISpeed` and `IDistance`. The speed prediction algorithm can be found in algorithm 2.

---

#### Algorithm 2 Speed prediction algorithm

---

```

1: specifiedSpeed ← the speed specified by the team
2: fromDistance ← the distance the team will travel from
3: toDistance ← the distance the team will travel to
4: totalDistance ← toDistance – fromDistance
5: lookupSpeed ← findLookupSpeed(specifiedSpeed)
6: lookupFromDistance ← findLookupDistance(fromDistance)
7: lookupToDistance ← findLookupDistance(toDistance)
8: while lookupFromDistance != lookupToDistance do
9:   distance ← lookupFromDistance – fromDistance
10:  predictedSpeed += distance * lookup(lookupSpeed, lookupFromDistance)
11:  fromDistance ← findLookupDistance(fromDistance)
12:  lookupFromDistance ← findLookupDistance(lookupFromDistance)
13: end while
14: predictedSpeed += (toDistance – fromDistance) * lookup(lookupSpeed, lookupToDistance)
15: predictedSpeed += (specifiedSpeed – lookupSpeed)
16: predictedSpeed /= totalDistance
17: return predictedSpeed

```

---

The data that is used for the speed prediction depends on the data label given to the route a team follows. If a team runs on the Paris route, the Paris data label will be used, and only Paris data files will be used in the prediction of the speed. If no data label is assigned, a combined data file will be used in the prediction. Furthermore, data from different years is weighted by geometric progression, devaluing older data over more recent data. In order to keep the speed prediction data accurate, the user has the ability to add new data files to the system.

### 7.1.3. Final algorithm

In section 7.1.1 several reasons are provided for why the linear programming formulation of 3.4.3 was not implemented. Instead, it was decided to improve the greedy algorithm.

#### Start buckets

The first improvement made to the algorithm was the creation of starttime buckets. This was implemented to minimize the number of teams that start separately. The main reason is safety. The Paris route starts by passing through a Banlieue, which are known for their high crime rates. Running in bigger groups is therefore safer. Furthermore, the teams find starting together more enjoyable. This results in competition between the teams. Thirdly, having fewer starting moments reduces the workload of the organization.

After the creation of the schedule, a method `createBuckets` is called. This method creates the start buckets for each route. First the teams are filtered by route to prevent mixing teams from different routes in the same bucket. Then the teams are sorted on starttime. Next the buckets are created and filled based on two conditions. If the difference between the starttime of a team and the bucket starttime is greater than 5 minutes or the maximum bucket size is reached, a new bucket is created. If this condition is not met, the team is added to the current bucket. This process is executed until each team is assigned to a bucket.

#### Requests

In section 3.5.3 an overview is given of the amount and types of requests proposed by the teams. In the previous year's event, approximately 23% of teams proposed such a request. Therefore, functionality was added to automate this. The user is able to add the most common types of requests. The first type is a request to start at a specific time, the second type is a request to finish at a specific finish time, and the last type is a request to stop at a location at a specific time for a particular duration. Before the request is incorporated in the schedule, the feasibility of the request is verified.

For the specific starttime request, the predicted finish time is computed. If the computed finish time is within the permitted time frame, the request is deemed feasible. If not, the request is denied and the user is informed of this decision. For the specific finish time request, the finish time is once again compared to the permitted time frame, and handled like the starttime request.

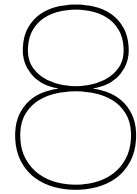
The last type of request, stop request, has two different checks. First the program requires as input the stop date, arrival time, distance and duration of the stop. These inputs are then used to verify whether the team is able to finish within the permitted time frame. By adding the stop duration and the time the team needs to run from the given location to the finish, the new finish time is computed. The second check validates whether the computed starttime,  $arrivalTime - (distanceToStop/speed)$ , is greater than the minimal starttime (slowest team finishes first). If both checks pass, the request is added to the list of requests.

If the organization is not allowed to add the request for the reasons mentioned above and the organization wants to add the request regardless the organization is able to manually change the start or finish time in the edit screen.

Before the the schedule is generated, the requests are handled. Each handled request has a computed start and finish time. The request information is anchored in the schedule. When the schedule is generated, the teams for which a request is added are skipped such that the request information is not overwritten by the generation of the schedule.

The edit screen comes after the schedule is created and allows the user to edit the schedule without restriction. This is the first time the user is presented with the schedule.

All computed times for verification and handling of the requests use predicted speeds by the speed calculation algorithm found in algorithm 2



# Process

This section describes the process that has been used to arrive at the solution. The process provides the framework to support the applications' development and ensure effectiveness of the team. The process should also make the project transparent and reproducible.

## 8.1. Methodology

Over the course of the project various methodologies have been applied in order to facilitate the development. In the selection of methodologies various aspects have been considered, such as size of the team, scope of the assignment and duration of the project.

### Scrum

The scrum framework was utilized as it is focused on maximizing work efficiency and has a clear division of responsibility (Maximini, 2015). Based on the duration of the project, weekly sprints were chosen. As a consequence of the sprint duration the frequency of various meetings was determined. Scrum meetings were held daily, and sprint meetings were held weekly. In the latter the sprint review, retrospective and planning of the next sprint were combined.

Each sprint started with the sprint planning. This entails assigning issues to team members, as they are prioritized on the issue backlog. It is worth noting that estimation was done purely on the teams' notion of manageable workload. The short duration of the project meant that there was not enough time to measure an accurate velocity, which is required for most common estimation methods. In order to keep track of progress on the issues, daily scrum meetings were held where each team member could raise issues they were facing. At the end of each sprint a new version of the program was released. This was followed by a sprint review where the team looked back on the issues they closed and which remained open. Finally the sprint is concluded by a sprint retrospective. The purpose of the retrospective is to identify organizational, personal and teamwork issues the team might face. Consequently, possible resolutions for these issues are discussed.

Within the scrum team multiple roles are to be fulfilled, since the agile/scrum methodology is utilized and the team consists of 3 people, only the vital roles required for scrum are assigned.

- Scrum master - Jasper Nieuwdorp
- Product Owner - Martijn van Meerten
- Scrum Team - Everybody

### CI/CD

As the size of the team is relatively small and the development rate is high it is preferable to automate as much as possible of the development process. With continuous integration the building, packaging and testing of an application is automated. This was implemented Using Github (Github, 2019) with Travis CI (CI, 2019) This also meant there always was a working version ready and a consistent, reproducible

environment was available. In order to locally build, package and test the application frequently Maven (Maven, 2019) was used. Since there was only one real delivery moment at the end of the project, it was decided to not utilize continuous delivery.

### **Agile**

Agile was used in order to fully benefit from incremental development and iterative life cycles. ("Project Management Institute", 2017). This allowed the team to adapt to changes in priority or demands as well as to technical limitations that occurred. In the research phase of the project a plethora of user stories were written down and later translated to requirements, both functional and non-functional.

### **MoSCoW**

In order to prioritize the backlog of requirements and to make sure the most important demands by the client were fulfilled the MoSCoW methodology was used. All requirements were classified within the following four categories: Must have, Should have, Could have and Would/Won't have. During the project certain priorities and requirements would shift which lead to placing it in a different category. At the start of the project a project plan was created mapping out when certain requirements were expected to be incorporated in the final solution.

### **UML Diagrams**

To visualize the program in its current state and the envisioned final solution, various UML Diagrams were made. For the former, package/class diagrams were maintained in order to give a clear view of the programs' architecture. For the final product, and to create an idea how the actors would use the program, use case diagrams were made.

#### **8.1.1. Code quality**

To keep the code understandable and maintainable, Javadoc (OracleCorporation, 2018) was added to the codebase. This provides an explanation about the input, output and functionality of every component. If the level of complexity of the method is high, regular comments can be added to increase the understandability of the method to the developers, although this should be an indication that perhaps refactoring the code is on order.

Different static code analysis tools were used to increase the code quality, namely Checkstyle (Sourceforge, 2018), PMD (& CPD) (PMD, 2018) and SonarLint (SonarLint, 2019). These plugins report on a range of potential issues with your code, such as unnecessary spaces, missing break statements in switches, but also information about infinite loops. Moreover, these tools also gives an indication of the complexity of the code in the application, which is very useful for developers.

To ensure a high code quality was maintained, code would not be allowed into the final product if any of the static analysis tools reported issues (as per the configuration). In addition, at least one manual review was required. A second team member would read and understand the code, and if applicable, request changes where deemed necessary. The reviewer would also confirm that the code was tested adequately (line and branch coverage shouldn't drop, and the quality of tests should be sufficient).

#### **8.1.2. Testing**

To make sure the program works as intended (verification), but also to make sure the program delivers what the client expected (validation), different methods of testing were utilized.

##### **Unit Testing**

To verify the software functions as specified, unit testing was utilized. sections of the code were tested individually by writing test cases in JUnit (JUnit, 2019). As a way to quantify the amount of tests, line coverage and branch coverage were measured by the tool JaCoCo (EclEmma, 2019). This tool only reports the quantity of test code in relation to the functional code, not the exhaustiveness or quality of the test code. To combat this, code reviews were carried out to judge the quality of the tests.

##### **Functional Testing**

To validate that the software performs as expected and desired by the client, weekly acceptance testing meetings with the client were held. During these meetings, the progress on the application was showcased, a potential end-user was asked to try out the application, and all stakeholders on the client

side were asked for their opinion. These meetings were also used to make notes about how the end user interacted with the application in order to test the usability of the software. Based on the feedback received from the client, requirements were added, removed or adjusted to reflect the new insights.

### 8.1.3. Stakeholders

This project had several stakeholders. These can be individuals representing either themselves or (groups of) other people. The following people were the project's stakeholders:

- Managing Director of the Roparun Foundation, representing the entirety of the organization.
- Employee of the Roparun Foundation, representing the end-user.
- Volunteer, In charge of the the schedule in previous years.
- Volunteer, In charge of the route and safety, representing the runners.

It is interesting to note that the various stakeholders each had their own demands and priorities. As developers it was our task to weigh those and make sure the most optimal overall solution would be delivered for all parties involved.

The stakeholder representing the Roparun foundation as a whole had a more long term view of the product and concerned itself less with details of the program that the end-user stakeholder found to be important. The person that made the schedule in previous years would prefer a fully automated implementation of their own method and experience most. The stakeholder that represented the runners was very important for the overall acceptance of the final product. Trust and acceptance are key aspects to the smooth introduction of any product.

### 8.1.4. Meetings

Over the course of the project there was a variety of meetings. Some focused completely on the development, some focused on the overall progress or on the client.

- Scrum meeting: Review, retrospective and planning (Weekly)
- Stand up (Daily)
- Client meeting (Weekly)
- Client acceptance testing (Weekly)
- Coach meeting (as needed)



# 9

## Product evaluation

In this chapter, the product is evaluated on the grounds of different aspects. First the software aspect is analyzed. Then the final product is compared to the requirements of the product set at the start of the project. Finally, the quality of the solution is compared to schedules of previous years.

### 9.1. Code evaluation

Software Improvement Group (SIG) is the company that measured the quality of the code base. SIG uses several metrics to give the maintainability of code a rating between one and five stars. Some of the metrics used to rate the maintainability of our code are: module coupling, code duplication, quantity of code, unit interfacing, unit size, unit complexity and test coverage. In week six and nine of the project the code base was sent to SIG. The feedback provided by SIG after the first upload will be elaborated in the following subsection.

#### 9.1.1. SIG

The code scored four out of five stars for the maintainability of the code. An above average grade, which means that the code was built and designed in such a way that it is easily extendable in the future. Nevertheless, some improvements could be made. SIG points out that the reason why the code did not get five out of five stars was because of unit interfacing. The score of unit interfacing depends on the percentage code in units with an above average amount of parameters. Methods with many parameters could lead to confusing method calls and frequently results in complex and long methods. Furthermore, it indicates a lack of abstraction. Several reasons are given why unit interfacing in the code needs to improve.

In our code two different constructors were implemented with an above average amount of parameters (five), relative to our code base. The constructor for the SchemeData class and the constructor for the Route class. SchemeData is a data class for one row in the schedule table. The Route class is a data class representing a route.

The problem has two possible solutions. One solution is to convert the class to a bean. A bean has an empty constructor and all fields have a get and a set method. The other solution is to make the class immutable. An immutable object is an object that will not change after it is created. An immutable class does not contain any set methods for its fields, only get methods. The advice from SIG was to transform the classes into beans, since the set methods were already implemented. It was, however, decided to convert the classes to immutable classes, since objects of these classes were not meant to change after creation.

The feedback on the quantity of test code was positive. At the time of the first submission to SIG, the branch test coverage was 66%. The goal was to maintain a level of test coverage between 65-70%.

To conclude, the code base scored above average at the time of the first submission. For the remainder of the project, the code base was checked for unit interfacing in the hopes of improving the code quality. In the feedback on the second submission, it was noted that the issue with unit interfacing was resolved, and that the overall code quality had improved. Furthermore, the amount of duplicate

code had decreased, and the test coverage remained at a sufficient level. The full SIG feedback can be found in appendix C.

### 9.1.2. Test evaluation

In figure 9.1 the statistics about the test coverage are shown. As can be seen the io, core, model, validator and view packages are extensively tested (using Mockito). However, the controller package has a branch coverage of 21%. The reason why this package has a relative low test coverage, is that these classes contain a lot of methods which use EventHandlers (e.g. button click actions) and methods related to the user interface (JavaFX). To increase the test coverage TestFX (TestFX) can be used. TestFX is able to simulate user interactions which can be used to test the methods which require user interaction. Due to time limitation, TestFX is not used to test these classes.

Element	Missed Instructions	Cov.	Missed Branches	Cov.
<a href="#">nl.roparun.roparun.controller</a>		29%		21%
<a href="#">nl.roparun.roparun.io</a>		86%		76%
<a href="#">nl.roparun.roparun</a>		9%		0%
<a href="#">nl.roparun.roparun.core</a>		91%		88%
<a href="#">nl.roparun.roparun.model</a>		100%		99%
<a href="#">nl.roparun.roparun.validator</a>		100%		97%
<a href="#">nl.roparun.roparun.view</a>		100%		100%
Total	3,388 of 10,031	66%	216 of 601	64%

Figure 9.1: Test coverage

## 9.2. Requirements evaluation

At the beginning of the project, the requirements were prioritized according to the MoSCoW method (see 8.1). These requirements can be found in section 4.2. In this section, The product is evaluated by whether these requirements have or have not been met.

### Must haves

The product meets all of the Must haves. Four out of the five Must haves had been implemented by then end of sprint three. The last Must have: "The application must be effectively usable by the client", was a very broad requirement. Initially this requirement would have been met by a clear and intuitive GUI, but later on, saving functionality was added, which caused a delay in meeting this requirement. This last Must have was implemented after sprint five. In the following sprints, the functionality satisfying the Must haves was improved upon and implementation of the Should and Could haves was started.

### Should haves

The product meets five out of the six Should haves. The requirement: "The tool should be able to limit the amount of traffic on the route based on known bottlenecks", has not been met, because this requirement became invalid. The bottlenecks along the route were deemed no longer important by the client (see subsection 7.1.1).

### Could haves

Our product meets three out of the five Could haves. The requirement: "The tool could have a feature to visualize the density per segment per time frame of the race", has not been met. The density along the route can be found in the logger of the program. Due to time limitation, the density is only textually visualized. A graph in the GUI would be a more suitable way to visualize the density. This could be implemented relatively easily in next versions of our application. The other requirement not met is, "The tool could update the predicted finish time by using the team's checkpoint times.". One way to meet this requirement, is to rewrite the application to a web application, where predicted finish times are automatically updated when a team passes a checkpoint. Another way would be to add functionality for uploading a list of checkpoint passage times. Then, using the speed calculator, the new finish times

can be calculated. Due to time constraints, this was not implemented.

### Would/Won't haves

Our product fulfills none of the Would haves. The requirement, "Automation of checkpoints and start.", is one for a different product. Not only limited time but, also the equipment needed to realize this requirement are the reason this requirement is not met. Nevertheless, meetings with the stakeholders have shown that the Roparun is very interested in automating the checkpoints and the start of the race.

## 9.3. Solution quality

One way of checking the quality is to use the data of previous years' runs and generate the schedule with exactly the same data as what was present at the time and compare this to the schedule used that year. However, it should be noted that schedules are always a prediction. Different properties can be used to compare schedules. When comparing the schedule created by the application and the schedule that was created for the race in 2018, two different aspects will be compared.

### Formula

Following from the problem specification, a solution that is considered good will have the following aspects:

- Ensured safety of all teams
- Minimized time frame for the start of the run (teams leave in quick succession)
- Ensure a good spread of teams at the finish, within the permit
- Adherence to team requests.

A scoring system of these aspects is introduced below. A solution is considered invalid if the government permit times are predicted to be violated. In order to weigh all aspects of a good solution, the following parameters should be minimized. To ensure a start in quick succession is the duration of the start ( $hs$ ); the time from the start of the first team until the last team in hours. To make sure there is a good spread at the finish the maximum of the number of people finishing every hour ( $nf$ ) should be as low as possible. The number of requests fulfilled ( $nr$ ) should be as high as possible, and will therefore be subtracted from the result. Finally two of the three objectives get their own weight so that the importance of the aspects can be adjusted, relative to each other, as to better represent the real-world significance of those objectives.

**minimize**  $(hs + (w_f * nf) - (w_r * nr))$

**such that** The finish time frame does not exceed the time stated by the government permit

**given**

$hs$  = start time frame in hours

$nf$  = max(number of people finishing every hour)

$nr$  = number of requests honored

$w_f$  = the weight assigned to the finish spread

$w_r$  = the weight assigned to the number of requests honored

(9.1)

When using this formula the schedule used in 2018 has parameters:  $hs = 14.4$   $nf = 68$   $nr = 72$  (not known, so it is assumed that all request have been fulfilled) and the schedule created with the application has parameters:  $hs = 13.6$   $nf = 56$   $nr = 69$ . Naturally the weights of both calculations should be equal, since the number of request fulfilled in 2018 is unknown this was given a relatively low weight of 0,25 compared to 1 for the weight of the finish spread. Since the weights are relative the weight for the duration of the start is fixed at 1. This gives the 2018 schedule a score of 64.4 and the schedule created by the program scores 52.35. The goal was to minimize the outcome so it can be said that the software solution performs significantly better on this aspect.

**Finish spread**

The spread of the teams arriving at the finish is already incorporated in the formula presented in section 9.3. It is, however, still noteworthy to visualize this spread. This has been done by creating a histogram of the number of teams arriving every hour for the schedule that has been used and the schedule created by the application. This yields the following results:

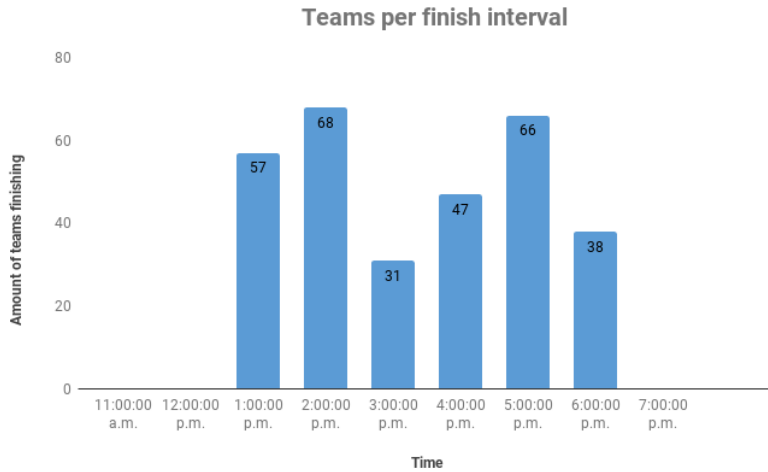


Figure 9.2: A histogram representing the number of teams finishing every hour, for the schedule used in 2018

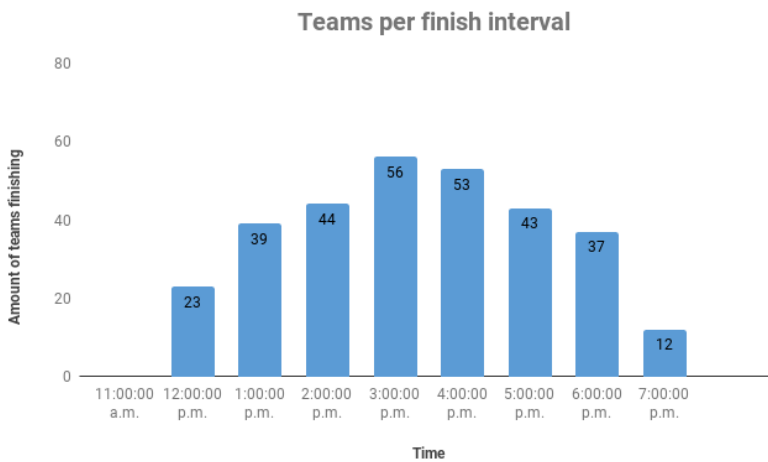


Figure 9.3: A histogram representing the number of teams finishing every hour, for the generated schedule

It becomes apparent that the schedule that the application created has a more consistent distribution that has a much lower maximum than the schedule used in 2018. This is also because in previous years the first hour and last two hours of the finish time frame were not used in order to account for uncertainty. Since this uncertainty has been reduced by the speed prediction model it is possible to utilize more of the finish time frame (but still with a safety margin).

**Checkpoint opening hours**

Another factor that would constitute a good schedule in the eyes of the Roparun is one where the checkpoints have to be open for a short amount of time. Again the schedule used in 2018 is compared to the schedule created by our program with the same parameters. This results in the following table:

In the generated schedule every checkpoint has to be open significantly shorter, so this metric also indicates that the program creates a better schedule.

<b>Checkpoint</b>	<b>Hours open generated schedule</b>	<b>Hours open schedule 2018</b>
CH01 Start Parijs	13:41:00	14:40:00
CH02 FONTAINE-CHAALIS	12:33:00	14:40:00
CH03 COUDUN	11:26:00	14:40:00
CH04 HOMBLEUX	10:05:00	14:00:00
CH05 RAMICOURT	8:52:00	13:30:00
CH06 VERTAIN	7:34:00	12:50:00
CH07 THULIN	6:31:00	12:10:00
CH08 SILLYCH08 SILLY	5:52:00	11:40:00
CH09 OPWIJK	5:42:00	11:00:00
CH10 TEMSE	5:25:00	10:30:00
CH11 OSSENDRECHT	5:05:00	9:40:00
CH12 DINTELOORD	5:07:00	9:00:00
CH13 BARENDRECHT	5:49:00	8:20:00
CH14 WILLEMSBRUG	6:07:00	8:00:00

Table 9.1: Amount of hours opened per checkpoint

# 10

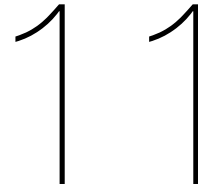
## Conclusion

The project began with a clear objective: create a scheduling tool that automates and assists the Roparun foundation in creating the schedule. During the research phase, the problem was formulated as a job-shop scheduling problem, existing solutions to scheduling problems were studied, and at that point a linear programming solution was thought of as the best option. During the project the benefits of the iterative approach to software development were discovered, as the carefully formulated requirements changed and a linear programming solution was no longer deemed to be the best option. Based on the research there was an immediate alternative approach that would meet the new requirements. A successful project would mean that the organization of the Roparun receives a functional application that allows them to create a schedule for their relay-run event, that outperforms the current method of scheduling.

The application is considered to be a good solution to the problem if it is able to generate a list of starttimes and predicted finishtimes for each team, which ensures a timely and efficient start for all teams, and a predicted finish where all teams arrive in short succession, within the allotted time frame. The created application contains this functionality and more, and is therefore accepted as a good solution.

This software development project was very interesting as the initial problem seemed very straight forward, but learning more about the context, problem space and requirements there were various obstacles to overcome. It was also made clear that not every aspect of the organizations wishes would be solvable by a better schedule. There were certain teams that consistently ran faster than the allowed speed and the organization wanted to accommodate this rather than restrict it by rules, as that is not within the spirit of the event. However, without changing the event itself there is a limit to accommodating this as there are legal constraints such as permits that force the race to be within a certain time frame. Various solutions for this were thought of, that were not necessarily part of the software development project, but with the goal of offering the best possible solution these were suggested to the organization for their consideration.

Delivering a complete solution that will actually be used to contribute to a charity and make an impact that matters was very motivating. In the end it can be concluded that meaningful software has been delivered that meets the current requirements but also is extendable and maintainable for the future.



# Discussion & Recommendations

In this section, the obstacles we ran into are discussed and an overall reflection on our work is given. Furthermore recommendations are given to the organization of the Roparun on what could be implemented or improved in later iterations of our application.

## 11.1. Reflection

During the first two weeks of the project, we searched for a lot of papers to use in our report. However, no problems were found in the literature that could be considered similar enough. We formalized the problem and created a linear programming formalization to solve our problem. In week three we wanted to show something to the client as soon as possible. Therefore we decided to implement a basic greedy algorithm that creates a schedule. After sprint one, the application was able to import the teams and route and create a basic schedule. The first three sprints all progressed according to our expectations. In sprint four, we planned to start improving the scheduling algorithm by converting the greedy algorithm to a linear programming algorithm. This was the main issue we encountered during the project. Linear programming was new to all members of the team, as it is not part of the curriculum of the Bachelor of Computer Science. We reached out to Erwin Walraven, PhD candidate within the Algorithmic group of the TU Delft, to help us move in the right direction. While making slow progress for two weeks on the linear programming solution, the requirements changed which made the linear programming approach no longer required. The last three weeks there were no major setbacks. The saving functionality was more complex to implement than expected, which caused it to take longer. However, since we accounted for some delays in the planning this didn't cause issues.

## 11.2. Recommendations

In this section technical and non technical recommendations are given to increase the quality and pleasantness of the race.

### 11.2.1. Technical

Although most of the requirements were met, and some of the "Should-" and "Could Haves" were implemented, the product can always be improved and extended upon. An extra feature which makes it possible to update the predicted finish times based on the passage times of teams at checkpoints would be of great value to the organization. It would provide a more accurate planning of the finish during the race. This gives the organization a change to anticipate, if needed. If this feature is implemented and the application would most likely be converted to a Web Application where it would be convenient for supporters of the teams to see when "their" team is going to finish. However, security problems might arise when the application is rewritten to a web application. As a benefit, a connection between a database used by the Roparun to store their data and the application would make the manual uploading of the teams unnecessary.

### **11.2.2. Non-Technical**

Some problems discovered during the project can not be solved by adjusting the schedule. Teams who run above the maximum allowed speed are one of these problems. These teams run 5-7 km/h faster than the teams who run at the minimum speed. On a race of 530km, this has a very large impact. The run duration of the teams could reach a difference of 20 hours. This is a burden on the entire organization of the race, as time is an important factor when requesting permits and looking for volunteers. During meetings with the client, this problem was discussed multiple times. The managing director of the Roparun was very curious if we had any creative ideas on how to increase the duration of the race for the fastest teams. Therefore the solutions we proposed are discussed below.

Since increasing the velocity of the slowest teams is not achievable without creating unfairness, increasing the run duration of the fastest teams is the best option. There are two approaches to do this are: decreasing the velocity and increasing the distance. Below, several examples are given to achieve this.

#### **Decrease velocity**

This could be done by holding up the teams. This could be achieved by creating an activity where the teams need to do a specific exercise(e.g. helping the organization along the route). Another approach to reach this result would be to slow down the teams by increasing the intensity, for example giving them extra baggage to carry with them. This is not favourable, since this introduces a different kind of challenge that is not necessarily in line with the physical strains of long distance running.

#### **Increase distance**

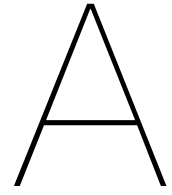
Another way to increase the run duration is to increase the length of the route. The fastest team could start at a location further back from the start. A requirement for this is that it should be possible for teams to register themselves at automated checkpoints which don't require a physical set-up. Finally, a competitive event could be added to the race for the teams who run above a specified speed. Examples of such an event could be: an optional extra segment of the route (detour). Furthermore, treadmills could be placed at the finish. The team that runs the highest distance on the treadmill wins the competition. This would also increase the audiences interest at the finish, as it is an extra activity taking place. The treadmills have the benefit of combining the time increase of an extra activity (since the team won't progress along the route), and a distance increase (competition for highest distance), which can be easily put into context in relation to the original achievement of running a long distance.



# Bibliography

- Apache. Apache poi, 2018. [Retrieved from: <https://poi.apache.org/> Accessed: 22/11/2018].
- Ken Arnold, James Gosling, and David Holmes. *The Java programming language*. Addison Wesley Professional, 2005.
- Jeff Atwood. Understanding model-view-controller, 2018. [Retrieved from: <https://blog.codinghorror.com/understanding-model-view-controller/>].
- Steven I-Jy Chien and Chandra Mouly Kuchipudi. Dynamic travel time prediction with real-time and historic data. *Journal of transportation engineering*, 129(6):608–616, 2003.
- Travis CI. Travis ci, 2019. [Retrieved from: <https://travis-ci.org/> Accessed: 22/11/2018].
- Jim Clarke, Jim Connors, and Eric J Bruno. *JavaFX: Developing Rich Internet Applications*. Pearson Education, 2009.
- Mike Cohn. *User stories applied : for agile software development*. Addison-Wesley, Boston, 2004.
- EclEmma. EclEmma, 2019. [Retrieved from: <https://www.eclEmma.org/jacoco/> Accessed: 22/11/2018].
- Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994. ISBN 0-201-63361-2.
- Amir Ghahrai. Moscow prioritization in software testing, Jul 2017. URL <https://www.testingexcellence.com/moscow-prioritization/>. [Accessed: 14/11/2018].
- Github. Github, inc, 2019. [Retrieved from: <https://github.com/> Accessed: 10/01/2019].
- Michel Goemans. *Lecture notes in linear programming, 18.310*. Massachusetts Institute of Technology, march 2015.
- Google. Google earth, 2019a. [Retrieved from: <https://www.google.com/earth> Accessed: 22/01/2019].
- Google. Google maps, 2019b. [Retrieved from: <https://www.maps.google.com/> Accessed: 22/01/2019].
- Itextpdf. Itext pdf, 2019. [Retrieved from: <https://itextpdf.com/en> Accessed: 22/11/2018].
- Ron Jeffries. Essential xp: Card, conversation, confirmation, 2001. [Retrieved from: <https://xprogramming.com/articles/expcardconversationconfirmation/>].
- JUnit. Junit, 2019. [Retrieved from: <https://junit.org/junit5/> Accessed: 22/11/2018].
- Jon Kleinberg and Eva Tardos. *Algorithm design*. Pearson Education India, 2006.
- Doug Lowe. *10 Differences between JavaFX and Swing*, 2016. <https://www.dummies.com/programming/java/10-differences-between-javafx-and-swing/> [Accessed: 19/11/2018].
- Maven. Apache maven, 2019. [Retrieved from: <https://maven.apache.org/> Accessed: 22/11/2018].
- Dominik Maximini. *Why a Scrum Culture Is Important*, pages 3–7. Springer International Publishing, Cham, 2015. ISBN 978-3-319-11827-7. doi: 10.1007/978-3-319-11827-7-1.

- Matthias Meidinger, Hendrik Ebbers, and Christian Reimann. Aerofx - native themes for javafx. In Giedre Dregvaite and Robertas Damasevicius, editors, *Information and Software Technologies*, pages 526–536, Cham, 2015. Springer International Publishing. ISBN 978-3-319-24770-0.
- Oracle. Java serializable, 2019. [Retrieved from: <https://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html> Accessed: 22/11/2018].
- OracleCorporation. Javadoc, 2018. [Retrieved from: <https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html> Accessed: 21/11/2018].
- H. Pandey, H. Rastogi, C. Gupta, and Anuja. Web-based network management system implemented using hibernate, jboss and spring framework. In *2nd International Conference on Telecommunication and Networks, TEL-NET 2017*, volume 2018-January, pages 1–5, 2018. doi: 10.1109/TEL-NET.2017.8343499. URL <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85049206450&doi=10.1109%2fTEL-NET.2017.8343499&partnerID=40&md5=2a7b6b9386b1971836c2258abee83324>. [Accessed: 15/11/2018].
- F Pezzella, G Morganti, and G Ciaschetti. A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research*, 35(10):3202–3212, 2008.
- PMD. Pmd, 2018. [Retrieved from: <https://pmd.github.io/pmd-6.10.0/>].
- Lawrence Premkumar and Praveen Mohan. Introduction to javafx. *Beginning JavaFX™*, pages 9–31, 2010.
- ”Agile Alliance” ”Project Management Institute”. Agile practice guide., 2017.
- Stichting Roparun. Roparun reglement 2018, 2018. [Retrieved from: <https://www.roparun.nl/wp-content/uploads/2018/04/Roparunreglement-2018-v-1.0-def.pdf> Accessed: 15/11/2018].
- Ihsan Sabuncuoglu and Burckaan Gurgun. A neural network model for scheduling problems. *European Journal of Operational Research*, 93(2):288–299, 1996.
- SonarLint. Sonarlint, 2019. [Retrieved from: <https://www.sonarlint.org/> Accessed: 22/11/2018].
- Sourceforge. Checkstyle, 2018. [Retrieved from: <http://checkstyle.sourceforge.net/> Accessed: 22/11/2018].
- TestFX. Testfx. [Retrieved from: <https://github.com/TestFX/TestFX> Accessed: 25/01/2019].



## Info sheet

This page is intentionally left blank.

**Title:** Roparun: 8000 people travel more than 500 kilometers for a good cause  
**Client:** Roparun  
**Final presentation Date:** 1 February 2019

## Description

**Short description:** Roparun is a foundation that organizes a yearly relay race from Paris, Hamburg and Almelo to Rotterdam to raise money for a variety of causes related to palliative care. The Roparun wanted an application that would help them with creating a starting schedule for the event, that will ensure a safe race and a smooth finish.

**Challenge:** Our challenge was to create an optimal schedule which takes several parameters into account: spread between the teams at the finish, incorporating participants' specific requests and accurate predictions, and combine these parameters in an accessible application.

**Research:** In our research phase, we considered different ways to solve our problem (brute force, greedy algorithm, linear programming, etc.). We analyzed the data of previous years and found valuable results about the specified speed by the teams. This resulted in ideas on how to improve upon the current method of scheduling.

**Process:** Our goal was to show a working version as early as possible to the client, so an agile approach was taken. Each week we planned to improve the quality of the algorithm. A linear programming algorithm was planned as the core mechanism, however due a change in requirements on the clients' side a greedy algorithm was deemed more suitable.

**Product:** The final product is a java application capable of creating schedules based on the uploaded team lists, routes and requests. The organization is able to change the schedule manually if needed. The schedules generated by the application can be exported in different formats. Also other valuable information (e.g. checkpoint opening and closing times) can be exported.

**Outlook:** The Roparun looks forward to use the application for the race of 2020. User guides are written such that data of coming years can be used as input to the program which increases the precision of the speed prediction, and in turn the schedule. When new routes are added to the event, it can easily be added to the program which makes the application future-orientated. This year the tool will be used together with the current method as a pilot.

## Team Members

**Name:** Jasper Nieuwdorp  
**Interests:** Methodology, Bridging the gap between computer science and real world problems  
**Contributions:** Scrum master, GUI, tooling, parsing and session storage.

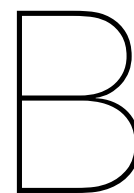
**Name:** Martijn van Meerten  
**Interests:** Statistics, Software Engineering Methods  
**Contributions:** Product owner, speed prediction, data analysis, GUI, editing functionality

**Name:** Lennart Overdeest  
**Interests:** Algorithm Design, Big Data  
**Contributions:** Schedule algorithm, request handling and exporting schedules

## Project Affiliates

**Client:** Wiljan Vloet  
**TU Coach:** Matthijs Spaan  
**Contact Person:** Lennart Overdeest [Lennart.overdeest@gmail.com](mailto:Lennart.overdeest@gmail.com)

The final report for this project can be found at: <http://repository.tudelft.nl>



# Original Project Description

## B.1. Dutch

De organisatie van de Roparun is op zoek naar een routetool die gaat helpen om een juiste planning te maken van de run. Met de routetool die ontwikkeld wordt, wil Stichting Roparun de veiligheid van de lopers en fietsers kunnen blijven garanderen. Dus door middel van het werken met deze routetool moet de organisatie de vertrek- en finishtijden van alle teams zodanig kunnen plannen, dat er afhankelijk van het parcours (de ene plek kan meer of minder mensen tegelijk behelzen) geen opeenhopingen ontstaan en er een gestroomlijnde run plaatsvindt. Daarbij rekening houdend met de nodige variabelen afgestemd op de wensen van ieder afzonderlijk team.

De variabelen waar aan gedacht moet worden, kunnen onder andere zijn;

- Gemiddelde loopsnelheid
- Afstand
  - Parijs, Hamburg, Almelo (halve)
  - Bij de grotere doorkomststeden ontstaat oponthoud c.q. vertraging op de loopsnelheid
- Individuele pauze momenten (door de teams zelf aan te geven)
  - Extra lange tijd in een bepaalde doorkomststad
  - Gezamenlijke koffie- of eetpauze
  - Komst van familie of sponsor op een bepaalde plek op de route

Door gebruik van deze routetool wil Stichting Roparun een lijst kunnen genereren die aangeeft welk team op welke tijd start en ook op welke tijd finisht, rekening houdend met vertrektijden die niet te ver uit elkaar liggen en een binnenkomst in Rotterdam van alle teams gezamenlijk, die veilig en soepel moet verlopen.

Op de route zijn diverse checkpoints (cp's). Het is wenselijk via die cp's te kunnen registreren en controleren of het beoogde schema wordt gevolgd en/of waar er knelpunten verwacht worden. Op die manier heeft de organisatie zicht op waar de teams zich bevinden en kan zij proactief acteren op mogelijke onveilige situaties. Na afloop van de run kan de organisatie dan ook per team de beoogde en daadwerkelijke route naast elkaar leggen en er lering uit trekken voor een volgende run.

Een bijkomende vraagstelling is de mogelijkheid van het toevoegen van een wedstrijdelement op een gedeelte van de route. Dit kan bijvoorbeeld voor een deel van de teams gelden, maar dan wel op een zodanige manier dat het passend is in de totale run. Verder moet de routetool door de organisatie zelf te hanteren zijn en moet worden voldaan aan alle hedendaagse eisen en moderne technologie.

## B.2. English

The organization of the Roparun is looking for a scheduling tool that will assist the organizers in creating the planning of the run. With the scheduling tool that will be developed the organization can continue to guarantee the safety of the participants (i.e. runners and cyclist). The tool should allow for coordinating the start and finish times of all teams in a manner that allows for the smoothest flow of traffic (since some parts of the route allow for more activity than others). There are many variables originating from the personal preferences of every team to take into account. Examples of such variables include:

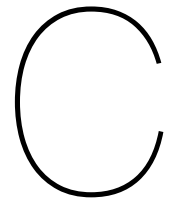
- Average running speed
- Distance
  - Paris, hamburg, Almelo (halve)
  - There is congestion in the larger cities, in which case the running speed decreases
- Individual breaks (determined by the teams)
  - Extended stay in a particular city
  - A joint coffee/tea break
  - Family members visiting on a particular part of the route

The main goal of the tool is to generate a list of starting times for each team (with predicted finish times) which ensures a timely and efficient start for all teams, and a finish of all teams close to each other in Rotterdam that should go over smooth and safe.

Along the route there are various checkpoints (cp's). It might be desirable to register, verify and adjust the schedule based on the check-ins at those checkpoints. This could give the organization an insight in projected bottlenecks and makes it possible to act proactively on dangerous situations. After the run the organization can compare the prediction for the run to the actual times to study and improve for future runs.

A possible addition to the project could be a competition-element into (parts of) the run. This doesn't have to include all teams, but it should match the character of the event.

The tool should be easily usable for the organization and should utilize modern standards and demands of modern technology



# SIG Evaluation

In this chapter the feedback that was received from the Software Improvement Group after submitting our code in week six and nine can be found.

## C.1. First feedback

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

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

In jullie geval zitten de meeste parameters in een paar constructors. Dat is een patroon dat we vaker tegenkomen. Let op dat in jullie geval "veel parameters" relatief is, omdat jullie qua score al behoorlijk hoog zitten gaat het hier om constructors met 5 parameters, namelijk die van SchemeData en Route. Bij dit soort classes zijn er in Java grofweg twee patronen:

- De class is een "bean", en alle velden hebben getters en setters
- De class is immutable, alle velden worden in de constructor meegegeven, en er zijn alleen getters (en dus geen setters)

In de twee genoemde voorbeelden doen jullie een soort combinatie van allebei. Het is beter om expliciet een keuze te maken. Gezien het aantal velden en het feit dat jullie toch al setters hebben zou ik adviseren om voor de eerste aanpak te gaan, en dus een constructor zonder parameters te gebruiken en alle velden met de setters een waarde te geven.

De aanwezigheid van testcode is in ieder geval veelbelovend. De hoeveelheid testcode ziet er ook goed uit, hopelijk lukt het om naast toevoegen van nieuwe productiecode ook nieuwe tests te blijven schrijven.

Over het algemeen scoort de code dus bovengemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase.

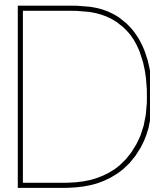
## C.2. Second feedback

In de tweede upload zien we dat het codevolume is gegroeid, terwijl de score voor onderhoudbaarheid met 4 sterren ongeveer gelijk is gebleven. Wel zien we een duidelijke verbetering op het gebied van Unit Interfacing, dat in de feedback op de eerste upload nog als verbeterpunt werd aangemerkt. Ook is het goed om te zien dat er andere verbeteringen in de onderhoudbaarheid zijn doorgevoerd, zonder dat deze expliciet genoemd werden in de feedback. Zo is er bijvoorbeeld wat geduplicateerde code opgeruimd. Deze verbeteringen zijn echter niet groot genoeg om de totaalscore naar de 5 sterren te laten stijgen.

Naast nieuwe productiecode is er ook nieuwe testcode toegevoegd. Het is positief dat het gelukt is om de verhouding tussen productie- en testcode stabiel te houden, vaak lukt dit minder op het moment dat de deadline in zicht komt.

Uit deze observaties kunnen we concluderen dat de aanbevelingen uit de feedback zijn meegenomen in het vervolg van het ontwikkeltraject.





# User stories

**User story 1** As the Roparun I want a scheduling tool so that I can plan for a safe and well organized event.

**User story 2** As a user I want to add teams to the program so that they can be included in the schedule.

**User story 3** As a user I want to add the route to the program so that it can be taken into account in the schedule

**User story 4** As a participant I want to add a request so that it can be taken into account in the schedule

**User story 5** As a user I want to be able to input request so that they can be incorporated in the schedule if possible

**User story 6** As a user I want to be able to decide exactly between what times the finish should be so that we can comply with the government permit.

**User story 7** As a user I want to have a (visual) prediction of the entire race so that I can get an idea where certain risks will occur and I can react to them.

**User story 8** As a participant I want to know if my request has been fulfilled so that it can plan accordingly.

**User story 9** As a user I want to have the option to save the schedule so that I can distribute it to the participants.

**User story 10** As a user I want to be able to change the schedule manually. So that special requests can be incorporated in the schedule.

**User story 11** As a participant I want to know my start time so that I can plan accordingly.

**User story 12** As a participant I want to know my predicted finish time so that I can communicate this with people who want to see me finish.

**User story 13** As a participant I want to know at what time I will be where along the route so that I can plan accordingly.

**User story 14** As a user I want to have insight in how the schedule came to be so that I can explain this to the participants and gain a better understanding.

**User story 15** As a user I want to be able to prioritize certain requests so that the importance of a request can be decided in accordance with the view of the organization.

**User story 16** As a user I want to see the impact a request has on the schedule so that I can weigh it with the priority and decide if it should be fulfilled.

**User story 17** As a user I want to have a schedule that includes the start times of all teams so that I can communicate this with the participants.

**User story 18** As a user I want to have a schedule that includes the predicted finish times of all teams so that I know how busy the finish will be at a certain time.

**User story 19** As the Roparun I want to have an optimal spread at the finish line so that crowds will be limited and the event will proceed smoothly.

**User story 20** As a user I want to make a schedule for all three races we organize so that they can be incorporated with each other.

**User story 21** As a developer I want to be able to compare my solution with previous years so that I can prove the benefits of our approach.

**User story 22** As the Roparun I want a scheduling tool that produces a better schedule than our current approach so that we can improve our organization.

**User story 23** As the Roparun I want a scheduling tool that will still work accurately in the coming years.

**User story 24** As the Roparun I want to be able to update the prediction of the race, during the race so that I can have a more accurate prediction of the traffic at the finish.

**User story 25** As a user I want to make sure the schedule tool takes known bottlenecks on the route into account so that they can be managed safely.

**User story 26** As the Roparun I want a solution for teams that are faster than 15 km/h so that they can still participate without causing the organization too much extra work.



## Speed data

The first column contains the specified speeds in meters per hour. The first row contains the distance from the start in kilometers. These distances are to checkpoints. The following columns (130.70, 215.60, 296.40, 342.30, 425.80, 518.40) have been left out for formatting reasons.

	<b>44.1</b>	<b>86.3</b>	<b>174.4</b>	<b>257.1</b>	<b>381.9</b>	<b>465</b>	<b>529.4</b>
11000	11507.57	11100.96	10413.82	9601.26	9965.55	10618.46	10519.74
11100	11618.61	11454.29	10672.82	9426.43	10056.74	10735.88	10466.01
11200	11664.20	11435.54	10518.15	9395.21	9982.26	10420.77	10729.28
11300	12005.70	11691.88	11152.85	9888.374	10421.54	10561.49	11128.58
11400	12258.71	11832.00	10544.56	10274.00	10359.47	10663.75	11178.39
11500	12204.34	11501.39	11083.29	10014.35	10296.71	10943.52	10872.99
11600	12398.38	11810.19	12178.01	10105.76	10597.84	11255.72	11658.14
11700	12269.18	11642.67	12143.15	10965.18	11017.74	11895.06	11573.54
11800	12196.31	12205.57	12285.08	10268.81	11121.34	11680.18	11093.56
11900	12411.85	11316.90	9716.344	11382.94	11379.60	11247.31	12064.49
12000	12920.91	12380.14	12433.02	10866.61	11558.24	12289.48	12052.87
12100	12905.20	12599.70	12311.17	11374.65	11188.00	12064.00	12275.43
12300	13487.30	13120.41	12965.66	12120.15	12079.90	12424.74	12599.29
12400	13996.29	11233.39	12549.28	13029.88	12703.15	13377.57	15100.91
12700	12502.86	13630.48	12839.93	12792.73	12575.69	13498.02	12979.43
12800	13203.59	12745.39	9902.526	13623.85	13093.62	14363.35	12274.04
12900	13798.01	14314.88	14269.77	13034.65	12989.23	14237.28	13801.08
13000	14164.91	13827.68	13461.47	12893.78	12737.97	13889.11	13686.21
13500	14057.02	14536.19	13908.47	13446.51	13152.46	14394.12	13911.77
15000	16318.34	16514.76	15215.66	14756.94	14587.28	14542.57	16627.05

Table E.1: Speed analysis Paris, 2018

# F

## Class Diagrams

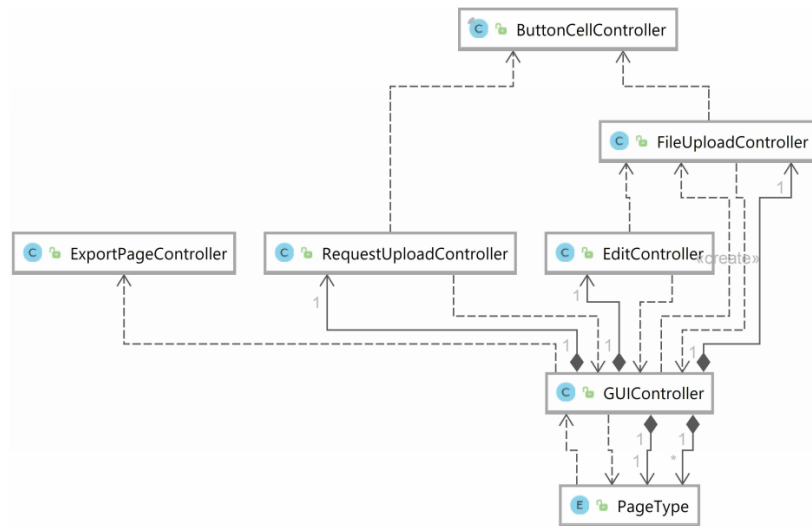


Figure F.1: Class diagram of the controller package

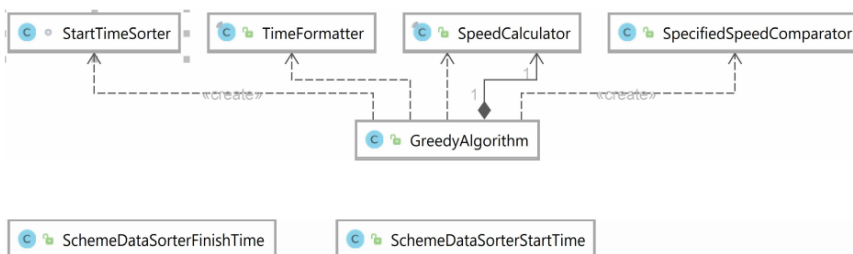


Figure F.2: Class diagram of the core package

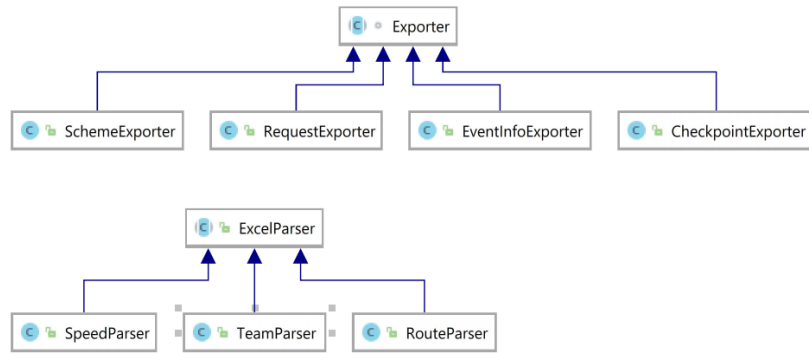


Figure F.3: Class diagram of the io package

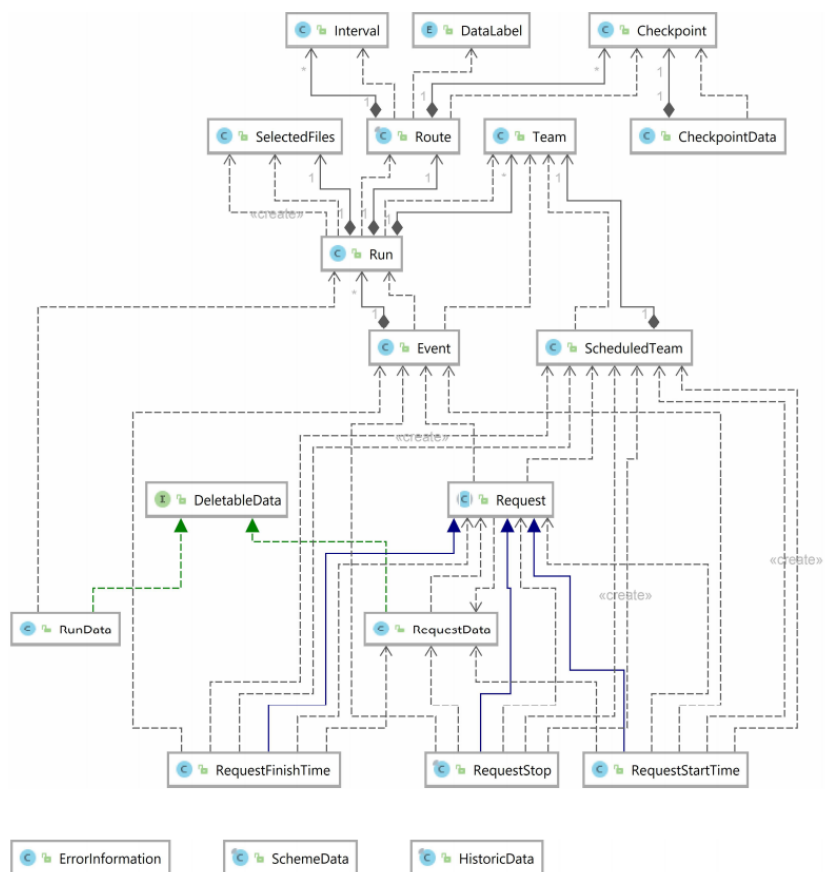


Figure F.4: Class diagram of the model package

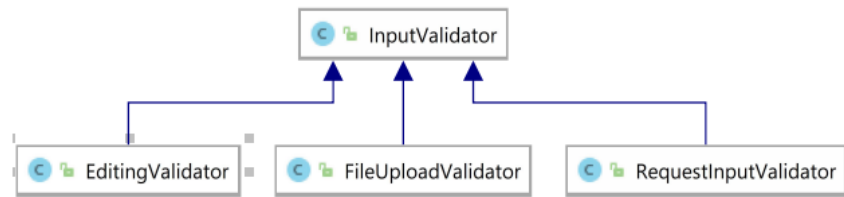


Figure F.5: Class diagram of the validator package