Code for Success
Software Development for Robotics Competitions

by **Sammy Pfeiffer**

Thesis submitted in fulfilment of the requirements for the degree of

Doctor of Philosophy
in
Software Engineering

under the supervision of

Mary-Anne Williams
and
Benjamin Johnston

University of Technology Sydney
Faculty of Engineering
and Information Technology

September 2020

AUTHOR'S DECLARATION

I, *Sammy Pfeiffer* declare that this thesis, submitted in fulfilment of the requirements for the award of Doctor of Philosophy, in the *School of Software*, *Faculty of Engineering & Information Technology* at the University of Technology Sydney. This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis. In addition, I certify that all information sources and literature used are indicated in the thesis. This document has not been submitted for qualifications at any other academic institution. This research is supported by the Australian Government Research Training Program.

Production Note:
Signature removed prior to publication.

SIGNATURE: _____

[Sammy Pfeiffer]

DATE: 7th September, 2020

PLACE: Sydney, Australia

Abstract

Robotics technologies have the potential to change the way we live for the better by reducing the difficulty of, helping with, or completely automating tasks. Robotics competitions such as RoboCup aim to push the field forward while providing an environment for participants to acquire important skills and knowledge. Most participants in these competitions are university teams with members from different backgrounds and levels of expertise, using different types of robots. These diverse teams must develop large and complex software stacks to accomplish their respective competitions' objectives.

This thesis aims to improve the software development process for these teams in regards to the development experience and competition outcomes. This will help push forward the robotics field and, consequently, our quality of life.

The available literature about software development methodologies for non-professional teams in robotics competitions is currently limited. The objectives of this thesis include enlarging the available knowledge in this domain and creating a practical set of guidelines that improve the software development experience and outcomes for robotics competitions. In order to do this, the software development methodology of the *UTS Unleashed!* team was analyzed over three consecutive years of participation in the RoboCup@Home Social Standard Platform League from the point of view of the development lead. Additionally, expert feedback was gathered to analyze, discuss, and compare the software development methodology of other teams and experts in the RoboCup League.

The research methodologies used in this thesis are Action Research, to explore *UTS Unleashed!*'s case study, and Grounded Theory, to analyze expert feedback gathered from a workshop and survey of members of the RoboCup community.

To the author's knowledge, this thesis presents the first longitudinal case study on a competitive team participating over multiple years in a robotics competition. Moreover, with the team under study achieving victory in their third year of participation. Furthermore, it is the first work showcasing expert feedback on a RoboCup teams' software development process from the RoboCup community.

This thesis concludes with a set of guidelines for software development practices for teams participating in robotic competitions. These guidelines offer insights and advice to improve competition team software development experiences and outcomes.

Dedication

To myself and my mental health. I never thought I would ever get this far.

To all the people that have supported me in one way or another up to getting here. At this point, I sweetly remember the teachers in Sa Colomina who pushed me to go to university and helped me get my first scholarship.

To all the people that believed in me, especially when I did not know any better.

To my friends who have always shown me that they are my family. I cannot express this in the same way in English, so here it is in Spanish:

Aunque la lista de nombres de todos los que me habéis inspirado y habéis estado ahí en los buenos y malos momentos, antes y durante este doctorado, es probablemente demasiado larga, he de dar las gracias a algunos de vosotros directamente y dedicaros esta tesis doctoral.

Irene, has sido mi constante desde siempre. Gracias.

Mi gente de Barcelona y de Ibiza. Jordi, Javi, Sebas, Toni, Jonathan, Hilario, Bence, Quique, Pol, la gente de AESS... Gracias.

Mi gente de Sídney, mis compañeros de casa, mis compañeros de escalada, mis amigos. Jenifer, Pedro, Rohan, Carlos, Gosia, Henry, Fiona, Callum, Mauricio, Vass, Clara, Jorge, Anton, Dani, Iris... Gracias.

Joan Aranda, Ricardo Tellez, Cecilio Angulo, gracias por creer en mí.

La gente de PAL que me ayudó a crecer, Luca, Jordi, Victor, Francesco... Gracias.

Acknowledgments

I acknowledge my supervisor Mary-Anne Williams for making me an offer I could not refuse, followed by a great trust in me and my work. And, obviously, for all your support and help during this PhD adventure.

I acknowledge my co-supervisor Benjamin Johnston, for all the support during these years. Your ways of thinking always opened my mind even though I may have seemed resistant. Moreover, I believe you made the completion of this thesis possible.

I acknowledge Jonathan Vitale and Meg Tonkin for their support and sportsmanship. Without you in the lab, I doubt I would have gotten this far.

I acknowledge Neil McLaren for his help during the RoboCup preparation and competition and his dedication in proofreading my thesis.

Finally, I acknowledge every member of The Magic Lab that was part of this journey. At different points in time each one of you did or said something or just were there when it was most needed.

Personal Motivation

Understanding my story with regards to robotics competitions helps frame the context of this work. Here I will describe it in first person.

I have been taking part in robotic competitions and challenges since my early university days in Spain in 2007. I attended a robotics course where we built a sumo fighting two-wheeled robot, which also could do line following, and at the end of it, there was a competition between the participants.

I joined the robotics club[1] that offered the course. The club organized the largest robot sumo fighting competition nationally, I became part of the organization and helped members develop platforms to participate. I participated with other team members in competitions such as the *Lunabotics NASA robot design competition* and a variety of local hackathons in Barcelona.

In 2012 the robotics company PAL Robotics came to our club to look for talented students interested in robotics to create a team to participate in RoboCup@Home with one of their new robots *REEM*, a real-size humanoid robot as can be seen in Figure 1.

[1]The club was called AESS Estudiants, from Aerospace & Electronics Systems Society Students, and welcomed anyone with interest in robotics and new technologies.

Figure 1: The robot REEM at RoboCup@Home in Eindhoven in 2013. I'm on the left with another former member of the club, Jonathan Gonzalez. Great times.

I joined the project, and it changed my life. We did not make it for the 2012 edition of RoboCup@Home, but we tried again in 2013. We successfully qualified and participated. We put tremendous effort in, the team was talented and motivated (Figure 2), but it was not enough. The bar was set extremely high, and we did not manage to focus on the correct directions to maximize the output of our efforts.

Figure 2: The 2013 team for RoboCup@Home, called REEM@IRI.

Thanks to this project, I joined PAL Robotics as an intern and afterward as an employee. I got motivated to get robots to **actually** become a reality, and to be part of a team that effectively, and in an enjoyable journey, enables robots to work in the real world.

In the following years, I have participated in RoboCup@Rescue, NASA Space Robotics Challenge, Move-it/Moving self-driving car hackathons, among others. I have learned and performed better individually but also as a team. My results have been positive, achieving podiums and awards in these competitions.

There is something victorious teams do that makes them successful. In this work, I want to take the unique opportunity to use the participation in Robocup@Home Social Standard Platform League (SSPL) over three consecutive years to uncover insights and strategies that help teams achieve exceptional outcomes.

List of Publications

1. Pfeiffer S., Ebrahimian D., Herse S., Le T. N., Leong S., Lu B., Powell K., Raza S. A., Sang T., Sawant I., Tonkin M., Vinaviles C., Vu T. D., Yang Q., Billingsley R., Clark J., Johnston B., Madhisetty S., McLaren N., Peppas P., Vitale J., Williams M.A. (2019, July). *UTS Unleashed! RoboCup@ Home SSPL Champions 2019*. In Robot World Cup (pp. 603-615). Springer, Cham.

2. Magyar B., Tsiogkas N., Deray J., Pfeiffer S., Lane D. (2019). *Timed-Elastic Bands for Manipulation Motion Planning*. In IEEE Robotics and Automation Letters, 4(4) (pp. 3513-3520).

3. Tonkin M., Vitale J., Ojha S., Clark J., Pfeiffer S., Judge W., Wang X., Williams M.A. (2017, November). *Embodiment, privacy and social robots: May I remember you?*. In International Conference on Social Robotics (pp. 506-515). Springer, Cham.

IMPACT

This thesis presents further contributions to society alongside the scientific publications directly related to it. Namely:

- Under the development of this thesis the *UTS Unleashed!* RoboCup@Home SSPL team achieved second place and best Human Robot Interface award in 2017 in Nagoya, Japan, second place in 2018 in Montreal, Canada, and won in 2019 in Sydney, Australia (the team's hometown).

- The outcomes of the *UTS Unleashed!* team were communicated in the media multiple times [2].

- The software stack developed for the RoboCup@Home SSPL participation powered social robotics experiments in hospitality and hospital scenarios[3].

- Parts of the software stack[4] were made open source for the benefit of the robotics community and beyond. Other parts of the system will be open sourced in the near future.

- The rulebook for the RoboCup@Home competition was improved as part of this work.

[2]For example, in Gizmodo: https://www.gizmodo.com.au/2017/07/meet-australias-newest-robocup-team/, IoTHub: https://www.iothub.com.au/news/uts-researchers-to-develop-ai-for-robot-waiter-466625, InsideRobotics: https://www.insiderobotics.com.au/robotics/personal-robots/Pepper-scores-a-world-class-goal-for-Sydney-team-in-RoboCup/, UTS media: https://www.uts.edu.au/about/faculty-engineering-and-information-technology/news/uts-brings-home-gold-home-robocup2019.

[3]The publications related to this work are pending at the time of writing this thesis.

[4]Pepper robot simulation: https://github.com/awesomebytes/pepper_virtual, Continuous Integration for the Gentoo Prefix Operating System: https://github.com/awesomebytes/gentoo_prefix_ci, pre-compiled Robotics Operating System (ROS) for unsupported platforms: https://github.com/awesomebytes/ros_overlay_on_gentoo_prefix, pre-compiled Operating System, libraries and applications for the Pepper robot to participate in RoboCup@Home SSPL: https://github.com/awesomebytes/pepper_os.

- Improvements in the popular open source robotics framework ROS were submitted during the work of this thesis.
- This work presents the first detailed description and evolution of the software development methodology and technical approaches of a competitive team for the RoboCup@Home SSPL competition during three consecutive years.
- This work presents a set of guidelines to help new and existing teams to improve their experience and outcomes when participating in robotics competitions.

Table of Contents

CHAPTER 1

INTRODUCTION

1.1 Motivation

Challenge projects help to drive human understanding and push the frontiers of science, engineering, and technology. They focus the attention of researchers on a specific hard problem, build resources, fuel innovation, and help to direct critical mass and momentum; and, importantly, provide benchmarks for completely new advances.

For many decades NASA, DARPA, and other organizations have used challenge projects to advance their knowledge of science, engineering, and technology [1]. Well known examples include the Apollo Missions, Moon Landing, Self-Driving Cars in desert [2] and urban terrains [3], and more recently, robotics in disaster scenarios [4]. For example, Thrun [5] described how in a year, from the first edition of the DARPA Grand Challenge to the second in 2005, the participants went from having all their autonomous cars stuck at the start of the race to five of them finishing the race driving autonomously for 131 miles for 7 hours.

The RoboCup initiative, commencing in 1997, was inspired by the earlier challenge projects to dramatically advance the field of robotics [6]. Over the ensuing 21 years RoboCup has grown and the first challenge of robot soccer extended to include robot rescue [7], robots in the home [8], and in the workplace [9].

RoboCup has had a dramatic impact on the field of robotics by providing benchmark challenges for roboticists used to measure scientific and engineering progress

[10]. RoboCup's success and fast technological advancements show that the concept of competition and science is effective and relevant [11].

One of the key problems in a field like robotics, that is constantly pushing the boundaries, is the difficulty in demonstrating and evaluating advancements. The use of benchmarks to measure progress is also found in many other areas, such as the use of specific datasets, like ImageNet [12], in Computer Vision and Machine Learning in particular. Benchmarking advancements in robotics is one of the most challenging because it requires the development of fully operational robotic systems, as benchmarks and advancements.

This dissertation develops and evaluates a set of guidelines for developing innovative robotics systems able to perform well on robotics benchmark challenges, i.e. robotics competitions. The case study is about the RoboCup@Home SSPL and includes feedback from experts. This league has been designed to advance the emerging field of social robotics.

The SSPL uses SoftBank's 'Pepper' robot. All teams competing in this league are restricted to using this platform, which is the current state-of-the-art social robot. The SSPL is a robotics contest focused on deploying social robots in a home-like environment and benchmarking their performance using a number of complex challenges. Hence, preparing the Pepper robot for participating in the RoboCup@Home SSPL implies having a robot ready for real life social robotics applications.

This dissertation analyzes the design, evolution, and performance of a RoboCup@Home SSPL team over three years of competition (2017, 2018, and 2019), from the decision making perspective of the software development process and its natural surrounding factors. Furthermore, discussion and analysis with other experts in the field through conversations, a workshop, and a survey ground this work.

1.2 Research Questions

This dissertation aims to answer the following research questions:

- How does a new RoboCup team successfully develop an effective software system for the RoboCup@Home Social Standard Platform League?
- How does the approach of a single team relate to other teams?
- What are common insights to improve the development process and its outcomes?

1.3 Research Objectives

The following objectives, which stem from the research questions, structure the dissertation:

- To identify the available practices from software development methodologies to improve robotics competitions' software development processes.
- To design, evaluate, and iterate the software development methodology for the three years of competition for our RoboCup@Home SSPL team.
- To gather insights from experts about the software development process for RoboCup.
- To analyze, discuss, and compare the software development methodology followed by UTS Unleashed! to the approach from other experts.
- To create a practical set of guidelines that improve the software development experience and outcomes for robotics competitions.

1.4 Significance

This work will showcase a real example of the evolution of practices in regards to software development in a distinctive context, a robotics competition case study over three years. Furthermore, this work develops a set of novel guidelines to provide advice to teams working in related situations, such as small teams working on technological challenges or competitions that have team-based software development in their requirements.

The literature review in chapter 2 revealed that the application of software development methodologies, such as Scrum or Extreme Programming, over different sized teams of employees in business organizations, is well understood. However, case studies on competitive teams lack representation. The few relevant publications are discussed in chapter 2, but they neither follow a project with a span of multiple years nor engage in obtaining the input of other experts in the same domain.

To the author's knowledge, this work is the first longitudinal case study on a competitive team participating during multiple years. Comprehensive advice or instructions for competitors in robotics competitions is considered by the author as urgently needed to advance the field, particularly social robotics. These competitors may use this guidance to improve their processes and outcomes, hence, improving the profile and outcomes of

future researchers and professionals. Furthermore, this work may increase focus and attention on the improvement of these processes for the benefit of all involved parties.

1.5 Scope and Limitations

This dissertation will focus on developing a successful software development methodology and software stack for the UTS Unleashed! team during the three years of participation in the RoboCup@Home SSPL competition with a qualitative scope. Further aims are: to achieve the best possible outcomes, including winning the competition; the well-being, experience, and learning of the team members; and the research outputs from participation in the competition.

A workshop and survey on software development methodologies for RoboCup were held to improve the methodology and give back to the robotics community. The results from these will be discussed within the results of UTS Unleashed! team's three-year journey. The author will create a set of guidelines, meaningful immediately for the RoboCup community, based on both.

This dissertation will not delve into the management of individuals or specific micro-management practices unless it is noted as an essential part of the analysis of the trajectory of the team. This dissertation will attempt to look from a higher-level perspective that focuses on the critical aspects that most influence team learning and performance.

While the author developed formal contributions during his PhD, as noted in the publication list, such theory is beyond the scope of this work.

1.6 Epistemology / Theoretical basis

Action Research will be used as the research methodology for exploring the three years of leading the software development for the UTS Unleashed! team. Grounded Theory will be used to analyze the information and feedback from experts in the field.

The characteristics of these methods are explained briefly in the Literature Review (chapter 2) and then discussed further throughout this dissertation in the Research Methods chapter (chapter 4).

CHAPTER 2

LITERATURE REVIEW

This chapter provides a literature review to establish the background behind the context and methodology of this dissertation. Furthermore, it reviews software development methodologies to showcase the research gap in relation to methodologies for robotics competitions.

This chapter begins by explaining the RoboCup event and, more closely, RoboCup@Home Social Standard Platform League (SSPL) as it is the focus of this work. The chapter continues with a review of Action Research (AR) and Grounded Theory (GT), the two methodologies used in this dissertation. The history of Software Development Methodologies (SDMs) and a note on the concept of proactivity in regards to software development follows. Subsequently, related works are discussed. Moreover, a review of the closest works to this research is presented and discussed. Finally, the background context for the rest of the dissertation is explained.

2.1 RoboCup

RoboCup, a contraction of 'Robot Soccer World Cup', is an annual international robotics competition founded in 1996. It was launched as "an attempt to foster AI and intelligent robotics research by providing a standard problem where a wide range of technologies can be integrated and examined" [13].

RoboCup was born differentiating itself from the previous AAAI Mobile Robot Competition and Exhibition, "the oldest AI-centric robotics competition in the world" [14],

by focusing on a task for a team of fast-moving robots under a dynamic environment instead of a single slow-moving robot. The RoboCup federation states the ultimate goal of this event to be that "by the middle of the 21st century, a team of fully autonomous humanoid robot soccer players shall win a soccer game, complying with the official rules of FIFA, against the winner of the most recent World Cup" [15].

Visser and Burkhard [11] explained how RoboCup, combining competition and science, is a successful, effective, and relevant platform for researchers. Furthermore, Maurice Pagnucco, Head of the School of Computer Science and Engineering at the University of New South Wales suggests that "Competition pushes advances in technologies. What we learn from robots playing soccer or navigating a maze can be applied to industry and help us solve difficult real-world problems" [16]. Additionally, Ferrein and Steinbauer [10] argue that besides technical skills, such as programming robots, students participating in RoboCup acquire other important skills like: interdisciplinary work to run and maintain a team, organizational skills, deadline-driven work, team-driven development, and community building.

Prior to RoboCup, the AAAI Mobile Robot Competition and Exhibition, had similar comments on the relevance of AI-centric robotics competitions. For example, Sebastian Thrun said "our tour-guide work was clearly strongly motivated by similar tasks in the mobile robot competition –in fact, without the competition, we would never have had the resources to do the museum tour-guide work" [14]. RoboCup not only helps to directly advance science in itself but also creates unforeseen opportunities for innovation.

RoboCup has been driving research results since its creation. One can find about 39,900 results on Google Scholar with the keyword 'RoboCup' on diverse topics, including: reinforcement learning, autonomous agents research, Simultaneous Localization And Mapping (SLAM), physical simulation, reasoning, and computer vision.

The event has grown significantly, with additional competitions having been added to the RoboCup event over the years. The contest currently has six major leagues, each with their own sub-leagues:

1. **RoboCup Soccer**, where teams of robots play soccer.

    - **Standard Platform League**, formerly known as Four Legged League when the robot dog Aibo was used, now using the 58cm tall Nao humanoid robot.
    - **Small Size League**, wheeled robots are used of up to 18cm of diameter and 15cm of height.

- **Middle Size League**, wheeled robots are used of up to 52cm * 52cm * 80cm of size and up to 40kg.
- **Simulation League**: 2D Soccer Simulation, 3D Soccer Simulation based on simulated environments.
- **Humanoid League**, non standard humanoid robots of three sizes: KidSize (40-90cm height), TeenSize (80-140cm) and AdultSize (130-180cm) are used to play soccer.

2. **RoboCup Rescue League**, which debuted in 2001, where participants face challenges involved in search and rescue applications.
   - **Rescue Robot League**, using real robots.
   - **Rescue Simulation League**, on a simulated environment.
   - **Rapidly Manufactured Robot Challenge**, where low cost, rapidly manufacturable small robots and robotic components that enable responders to more safely and effectively perform hazardous mission tasks are developed.
3. **RoboCup@Home**, which debuted in 2006, aims to develop service and assistive robot technology with high relevance for future personal domestic applications.
   - **RoboCup@Home Open Platform League**, formerly just RoboCup@Home, which accepts any robot that follows the rulebook specifications.
   - **Robocup@Home Domestic Standard Platform League**, using the Toyota HSR robot as standard platform.
   - **RoboCup@Home Social Standard Platform League**, using SoftBank's Pepper robot as standard platform.
4. **RoboCup Logistics League**, which debuted in 2012, is an application-driven league inspired by the industrial scenario of a smart factory.
5. **RoboCup@Work** which debuted in 2016, which targets the use of robots in work-related scenarios.
6. **RoboCupJunior**, which debuted as a league in 2000, contains leagues for students up to 19 years old to participate in.
   - **Soccer League**, similar to Small Size League.

- **OnStage League**, formerly Dance League, where teams of students are challenged to design, build and program a robot or robots to perform in musical and theatrical presentations.
- **Rescue League**, similar to the Rescue Robot League.

2.1.1 RoboCup@Home

The founders and active members of the competition define RoboCup@Home as:

> "RoboCup@Home is a competition where domestic and service robots perform several tasks in a home environment, interacting with people and with the environment in a natural way. Natural interaction means that a robot is expected to interact with the environment and with other people, as any person would do. So natural forms of human-robot interaction include speech and gestures, but not joysticks or keyboards.
>
> During the competition, the teams are required to perform several tests. Since their total score is the sum of the scores obtained in each test. Each test requires a combination of different functionalities (including navigation, object perception and manipulation, person detection, and tracking, etc.) and the score is related to the accomplishment of the task.
>
> RoboCup@Home started in 2006, and its main characteristic is that it changes tests every year while maintaining the same basic functionalities. " [17]

RoboCup@Home is a great background initiative for research, as can be observed by the over 10,000 results on academic search engines searching for the keywords similar to "RoboCup@Home", highlighting the amount of research done in reference to it. Further reasons to be interested in this competition will be discussed in chapter 8.

The RoboCup@Home competition runs over a 6 days period, with 1.5 days of setup, in a test arena similar to a home environment. Some tests are also executed in real environments like restaurants or shopping malls. The exact distribution of the arena is not announced beforehand so teams need to use the setup days to prepare. Furthermore, the real environments to be used are only announced hours before the competition.

Holz, Iocchi and Van der Zant [18] state that good results are obtained only when a complete system, implementing all the desired abilities, is successfully demonstrated during the competition. Furthermore, they state that by requiring teams to participate in many different tests, the development of general solutions is forced as hacking specific

solutions for many different problems would be ineffective. The competition tries to enforce general solutions and reward them. The goals of competing teams may or may not align with this vision as the balance between achieving research outputs and a good competition score is hard to maintain as we will see throughout this work.

The rulebook spans over 100 pages, including descriptions of the philosophy of the competition, scenarios, tests, and scoring systems. To avoid reading the whole rulebook and still provide a reasonable understanding of how the competition works, an example of a RoboCup@Home Social Standard Platform League (SSPL) test is shown and commented here.

2.1.1.1 Rulebook Test Example: Cocktail Party

The description and scoring for the test "Cocktail Party" from the rulebook of 2018 [19], used in Montreal, Canada is shown in the following pages. Afterwards, a detailed description of how the test looks and what it involves is provided. [1]

[1] For an even better understanding the reader can watch a video hosted on YouTube. A run of the cocktail party test of UTS Unleashed! team in RoboCup 2017 in Nagoya, Japan, with partial subtitles is showcased.

5.1. Cocktail Party [SSPL only]

The robot has to learn and recognize previously unknown people, and fetch orders.

5.1.1. Focus

This test focuses on human detection and recognition, safe navigation and human-robot interaction with unknown people.

5.1.2. Setup

- **Party room**: any (large) room inside the apartment when normally a party would be held.
- **Guests:** At least five people are distributed in a predefined *party room* either sitting or standing. Three of the guests have drink orders.
- **Bar:** The bar is any flat surface where objects can be placed, in a room other than the *party room*. All available beverages are on top of the bar.
- **Bartender:** The Bartender may be standing either behind the bar or next to it, depending on the arena setup.

5.1.3. Task

1. **Entering:** The robot enters the arena and navigates to the *party room*.
2. **Getting called:** The standing guests with an order call the robot simultaneously, either rising an arm, waving, or shouting. The robot has to approach one of them. Optionally, the robot can skip the call detection and ask for a person to step in front of it (the referees determine who approaches to the robot).
   The calling person introduces themself by name before giving the order of a drink. The robot asks for the person's name and obtains their drink order.

3. **Taking the order:** After the robot has taken the order of the first guest, it can either take more orders or proceed to place the order.
4. **Placing the orders:** The robot has to navigate to the *Bar*. The robot must repeat each order to the *Bartender*, clearly stating:

   4.1. The person's name
   4.2. The person's chosen drink
   4.3. A description of unique characteristics of that person that allow the *Bartender* to find them (e.g. gender, hair colour, how they are dressed, etc)

   While the robot places the orders, the people in the *party room* change their places within the *party room* (on request of the referees).
5. **Missing beverage:** One of the ordered drinks is not available and therefore missing from the bar. The robot should realize this inconvenience and tell the *Bartender*, providing a list of 3 viable alternatives. If the robot cannot detect which drink is missing, the *Bartender* can be asked to state which of the beverages is not available and provide a list of 3 alternatives.

6. **Correcting an order:** The robot should navigate back to the *party room*, find the person whose drink is missing and provide the alternatives for them to choose from.

   If the robot returns to find a person and the person is not there, it should call that person loudly and the person should respond (either through sound or by waving their hand). The robot should go to the person who is speaking and waving their hand to check their identity.
7. **Placing the corrected order:** The robot should navigate to the bar and inform the bartender of the change to the guest's order.

5.1.4. Additional rules and remarks

1. **Repeating names:** The robot may ask to repeat the name if it has not understood it.
2. **Misunderstood names:** If the robot misunderstands the name, the understood (wrong) name is used in the remainder of this test.
3. **Misunderstood order:** If the robot does not understand the order, it can continue with an own assignment of drinks to people or with a wrong, misunderstood assignment.
4. **Approaching non-people:** If the robot approaches a person that is not calling and asks for an order, the person indicates that they does not want to order anything. No points can be scored for understanding names or orders, or for grasping or delivery for a non-calling person.
5. **Guest description:** The guest's description must be unique inside the scenario. For instance, it make no sense to state that a person is wearing a red T-shirt if two people are wearing them. In the same sense, stating that the ordering guest is *tall* can lead to confusion, but stating that is the *tallest* does not.
6. **Changing places:** After giving the order (when the robot is not in the *party room*), the referees may re-arrange the people including their body posture. That is, a sitting person may change to a standing posture and vice versa.
7. **Positions and orientations:** All test participants roughly stay where they are (if not asked to move by the referees), but they are allowed to move in certain limits (e.g. turn around, make a step aside). They do not need to look at the robot, but are requested to do so, when instructed by the robot.
8. **Empty arena:** During the test, only the robot, the guest, and the Bartender are in the arena. The door opener, the referees and other personnel that is not assigned as test people will be outside the scenario.
9. **Calling instruction:** The team needs to specify before the test which ways of getting the attention of the robot are allowed for standing persons. This can be waving, calling, or both of them. The robot can also decide to skip this part, by asking for people to get close to it.
10. **Sitting persons:** Sitting persons might have an order but are not actively calling the robot.

5.1.5. Referee instructions

The referees need to

- Select at least 5 volunteers and assign names from the list of person names (see Section 3.3.8)
- Arrange (and re-arrange) people in the textitparty room. At least one is sitting
- Assign orders to two standing persons
- Assign an order to a sitting person
- Select the person (bartender) who will serve the drinks
- Place drinks at the bar while one drink is missing
- In case the robot skips the calling detection, select the ordering person to approach the robot
- Write down the understood names and drinks during an order and update the order accordingly

5.1.6. OC instructions

2h before test:

- Specify and announce the rooms where the test takes place
- Specify and announce the location where the drinks are served (i.e. bar location)

5.1.7. Score sheet

The maximum time for this test is 5 minutes.

Action	Score
Taking the orders	
Detecting calling person	*2 × 15*
Finding sitting & distracted person	*30*
Understanding and repeating the correct person's name	*3 × 5*
Understanding and repeating the correct drink's name	*3 × 5*
Placing orders	
Repeat the correct name & drink to the Barman	*3 × 5*
Provide an accurate description of the guest to the Barman	*3 × 30*
Missing beverage	
Realize the missing drink	*20*
Provide 3 available alternatives to the Barman	*20*
Understanding and repeating the alternatives to the Barman	*5*
Correcting the order	
Find the guest without calling them	*20*
Find the guest by calling them	*10*
Repeat the correct list of alternate drinks to the guest	*5*
Understanding and repeating the corrected order	*5*
Place the corrected order	*5*
Penalties	
Talk to something that is not a human	*-1 × 20*
Special penalties & standard bonuses	
Not attending (see sec. 3.10.1)	*-50*
Outstanding performance (see sec. 3.10.3)	*27*
Total score (excluding penalties and bonuses)	*270*

The test is described first by the typical flow of the challenge, then additional information about how it is scored is provided, and finally, how strategic decisions may be made by teams in order to achieve the best results are showcased.

The test is about the robot taking care of drink orders of attendees of a party in the living room. The test starts with the robot outside of the arena, which is shaped as an apartment, in front of the main door. The robot must wait for the door to open which is the signal that starts the five minute timer for the test. When a referee opens the door, the robot must navigate autonomously and safely to the living room. The robot has a previously built map of the arena in its system, usually constructed by the team during the setup days, with rooms and key elements tagged in it in order to navigate to them.

Once the robot reaches the living room, the people in the party, distributed randomly around the room and with some of them sitting and others standing, start grabbing the attention of the robot by calling it using their voice and with waving gestures. By this point any team would be at the one minute mark into the test. Then the robot detects a calling person (either via sound source localization or visual recognition of gestures, or a combination of both) and approaches them, making it clear who they detected calling.

Here, the robot is navigating towards the calling person in a crowded area with the possibility of the furniture having moved as guests are using it, especially in the case of chairs, so safe navigation is important. Once the robot arrives and confirms the person did call it, it asks for the person's name, making it clear it understood correctly by repeating the name and then asks for the drink this person wants, and it confirms in the same way. The robot will memorize how the person looks, as it is expected to provide an accurate description of the guest to the barman, and to bring the drink back to the same person, noting that this person may not be in the same place when the robot returns with the order.

At this point, the robot could either go to the barman in another room to get the order, or try to get another attendant of the party to provide the order. Both approaches are valid but have different implications in scoring, timing, and risks associated with something going wrong.

In the competition in 2018, no team got to the point of getting to the barman, let alone returning to the guests within the five minutes of the total time available for this test. This showcases the spirit of the competition where the tests are complex to master and a variety of approaches are possible.

Analyzing the test with the focus on scoring, we note that navigating to the living room as fast as possible implies having more time later on to perform the rest of the

tasks that actually score points, as this is just a requirement to get started and it does not score. If the robot was to collide with something while navigating to the living room, the test would be aborted and the team would score 0 for this run of the test.

Furthermore, when the robot detects a calling person, the team will be awarded 15 points if it is clear it detected the correct person calling. If the robot talks to something that is not human, for example, by navigating to the wrong place, facing the wrong direction, or having a false detection, a penalty of -20 points is applied. Moreover, when the robot arrives in front of the calling person, it must confirm that this person did indeed call it, because otherwise the robot would be losing time by interacting with the wrong guest.

Additionally, when asking for the person's name and drink, both need to be confirmed to make clear they were correctly understood by the robot so the referee can award 5 points for each of them.

The teams can engage in different strategies in order to ensure robust scoring, e.g. make sure the team scores over 0, or aiming for higher scoring with riskier strategies. These strategies depend on the robot skills that the team perceive as being more robust. For example, time is passing as the robot scans the room for a calling guest so UTS Unleashed! took care in choosing an optimal placement in the living room to navigate to and commence scanning for calling guests. This placement would minimize the need of the robot to navigate again to find guests and maximize the amount of guests it could potentially see without moving.

Furthermore, once a guest has been found and their name and drink has been taken, the robot must ensure they can recognize them again and describe them to the barman. An example approach for this is to use face recognition technologies to ensure who the person is while also memorizing how this person is clothed. This would help finding them from far away, as face recognition may not perform satisfactorily if the faces found in the image are too little. In the case of UTS Unleashed!, this strategy is used. This strategy, however, takes some time as it implies asking the person to stand in front of the robot to robustly learn the color of its clothing by looking at them closely. This can take up to two and a half minutes of the total five minutes allowed for this test.

Subsequently, after having taken the order from one guest, the team must decide what the robot should do next. If the team decides to program the robot to go to the barman, autonomous navigation to another room must be performed, potentially taking longer than approaching a new guest in the same room. Telling the barman the order of one guest would provide 5 points with additional 30 points for an accurate description

of the guest. However, if the robot was to detect a new guest first instead, providing 15 points, and understand its name, 5 points, and a drink, another 5 points, then it would score a further 25 points. Additionally, it would be able to provide the barman two orders potentially doubling the score at that point. Which strategy to use could be based on the time remaining for the test, which the robot could keep track of and decide dynamically what to do based on that. Or, the team could hardcode it based on the belief of which robot skills are more likely to perform satisfactorily.

2.1.2 RoboCup@Home Social Standard Platform League

The author of this dissertation and its team participated in one of the three leagues found in RoboCup@Home, namely the Social Standard Platform League. As the name suggests, this league uses a standard platform and tries to emphasize the social aspects of domestic robots. The league was established in 2017 with its first edition in Nagoya, Japan, and has been running since then. This thesis follows the editions of the years 2017, 2018, and 2019.

In the editions 2017 and 2018, this league had specific tests adapted to be solved by a social platform, the Pepper robot, i.e., making use of Human Robot Interaction (HRI) for some tasks. An example would be manipulation, as robots are expected to pick up objects, but this is challenging for this league's standard platform as the robot was not designed for this practice.

2.1.2.1 Pepper robot

All competitors in the RoboCup@Home SSPL league are mandated to use the 'Pepper' robot manufactured by Softbank robotics. This provides an even playing field and potential for collaboration, where teams must differentiate based on software rather than the hardware that they can buy or assemble. Softbank robotics describes their 'Pepper' robot [20] as follows [21]:

> "Pepper is the world's first social humanoid robot able to recognize faces and basic human emotions. Pepper was optimized for human interaction and is able to engage with people through conversation and his touch screen. "

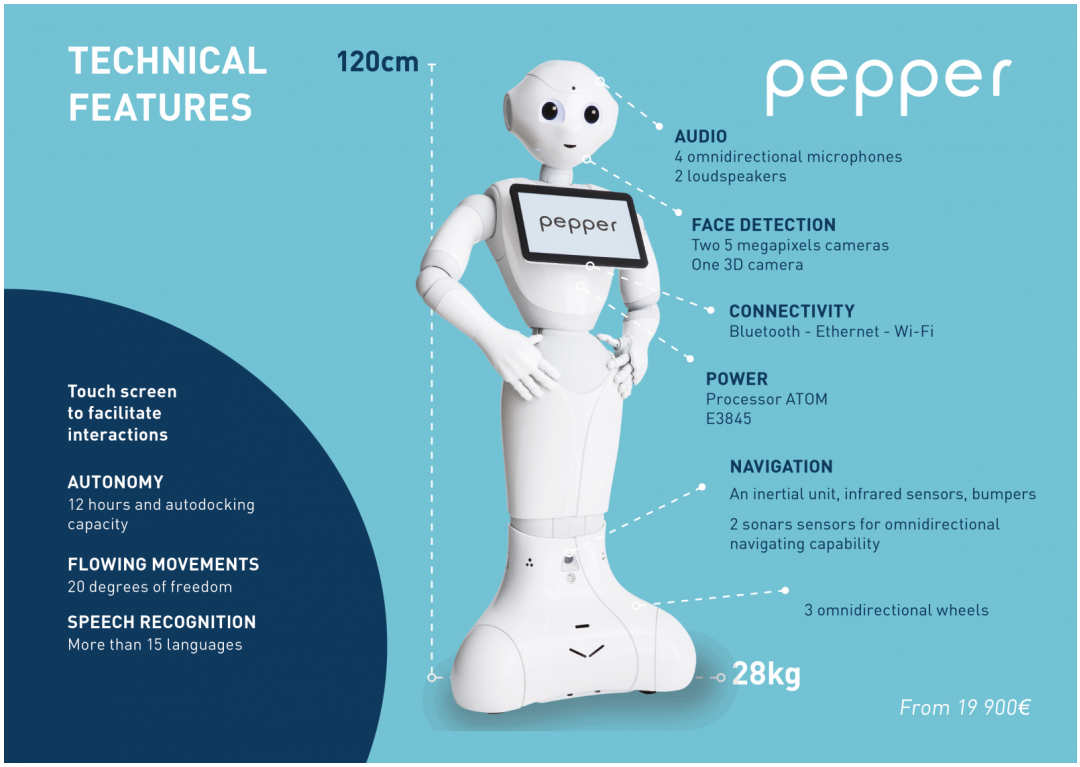


Figure 2.1: Brochure of the Pepper robot by SoftBank Robotics.

2.1.3 UTS Unleashed! RoboCup Project

The University of Technology Sydney (UTS) created the UTS Unleashed! RoboCup@Home SSPL team to engage in making safe and intelligent artificial intelligence for social robots a reality, to make significant scientific contributions in artificial intelligence and social robotics, to build national capability in these fields, and to provide transformational leadership opportunities and transdisciplinary learning experiences for UTS students in a practical manner [22].

The project was set to last three years, including the editions of 2017, 2018, and 2019 of the RoboCup event. The team was part of the Innovation and Enterprise Research Laboratory (The Magic Lab), established in 2002. Its laboratory members focused its interests on disruptive technologies (including social robotics), data science, explainable artificial intelligence, artificial intelligence policy, strategic innovation, entrepreneurship, and design thinking, in between other fields [23]. Researchers in the lab have competed at RoboCup since 2002.

2.2 Action Research

The research methodology known as Action Research (AR) is used in this thesis to investigate the process of software development for a robotics competition. Checkland and Holwell [24] offer a careful definition and discussion of AR, which is briefly summarized below.

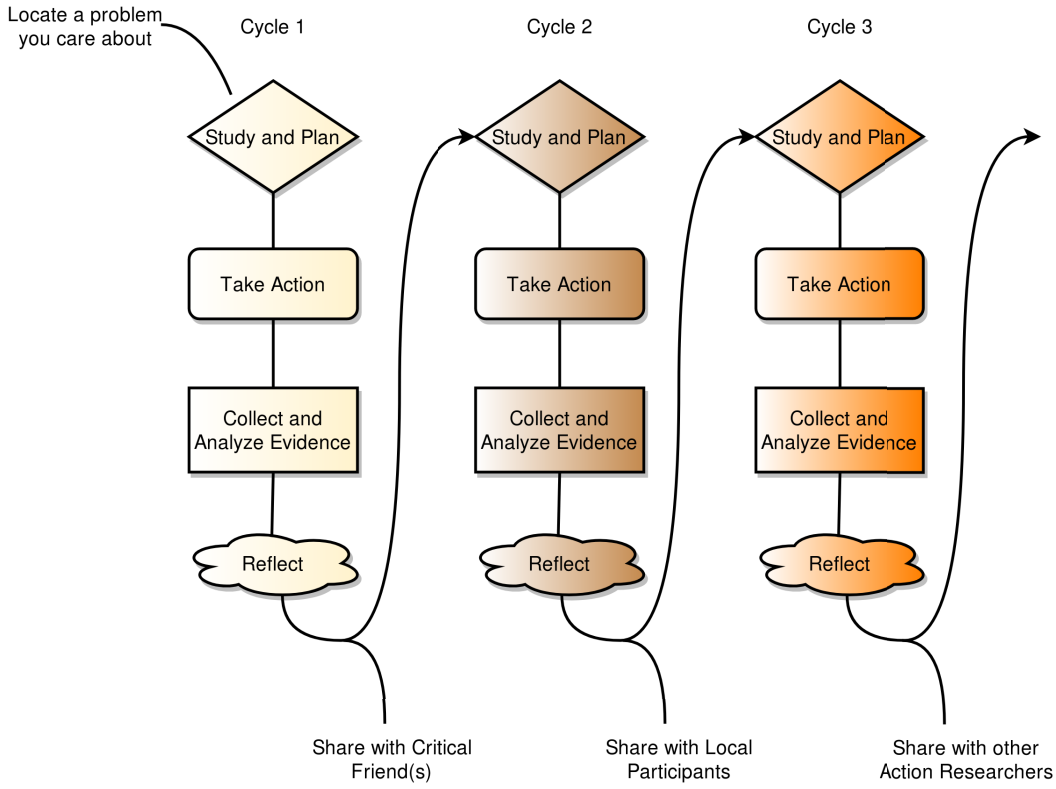


Figure 2.2: Model of Action Research from from Margaret Riel [25]

On a high-level view, cycles of Action Research can be understood as seen in Figure 2.2. The process starts with a specific theme of interest where we research and plan a strategy. Then we take action on it while collecting and analyzing evidence. At the end of the cycle, we reflect on the knowledge found. From this knowledge, we start another cycle where we repeat the steps for as many iterations as possible, depending on our resources.

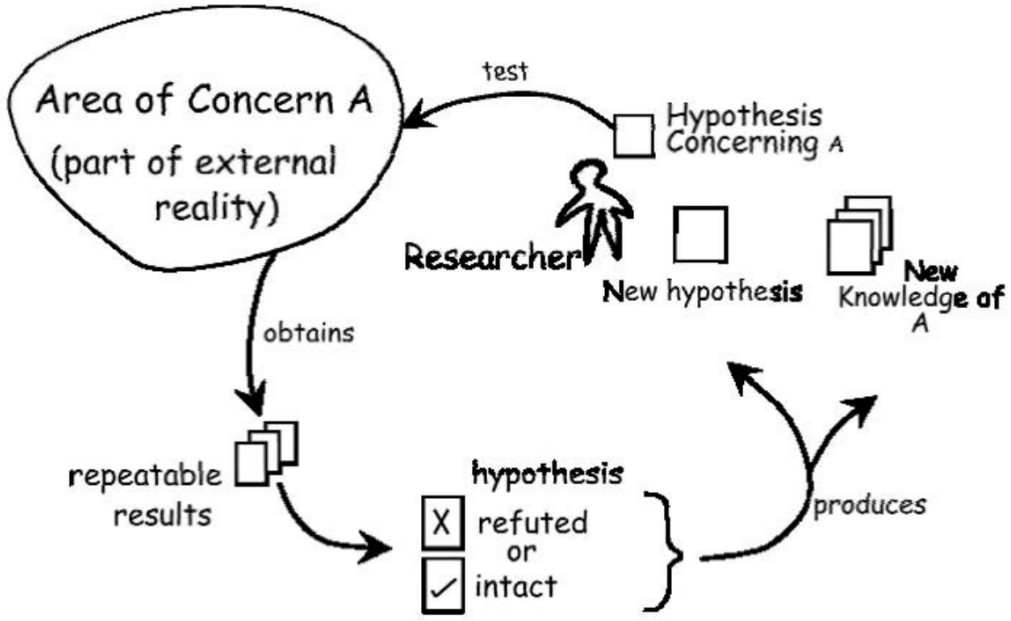


Figure 2.3: The hypothesis-testing research process of natural science, from [24].

To understand a specific action research cycle, we start with Figure 2.3. This figure represents the usual hypothesis-testing research process of natural science. The researcher starts with a hypothesis about an area of concern. The researcher designs a test so to obtain repeatable results to confirm its hypothesis and, from there, produces new knowledge or new hypothesis.

As Figure 2.4 shows (and paraphrasing Checkland and Holwell [24]), particular linked ideas F[2] are used in a methodology M to investigate an area of interest A, we define:

- F: Framework of ideas: The specific ideas used in every cycle related to the methodology to follow.
- M: Methodology to follow: The practices used to work on the framework of ideas.
- A: Area of concern: The topic that surrounds our framework of ideas and the methodology we follow.

It is important to note that using the methodology will teach us about our area of concern A and also about the suitableness of our ideas F and the methodology itself M. Also it is relevant that F, M and A are susceptible to change during the research.

[2]Not marked in the original image in Figure 2.4 as F.

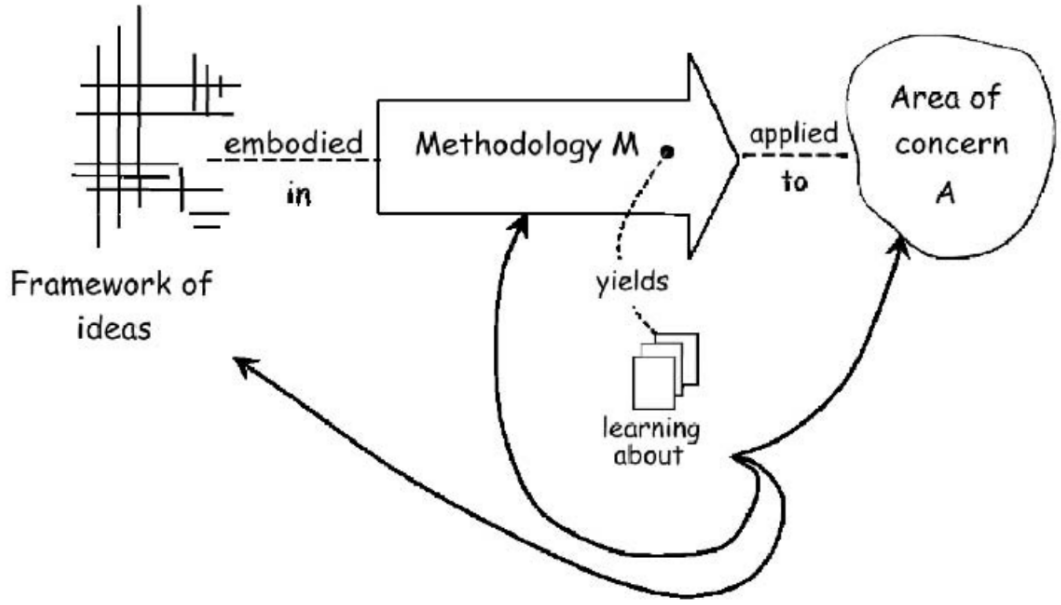


Figure 2.4: Elements relevant to any piece of research, from Checkland and Holwell [24].

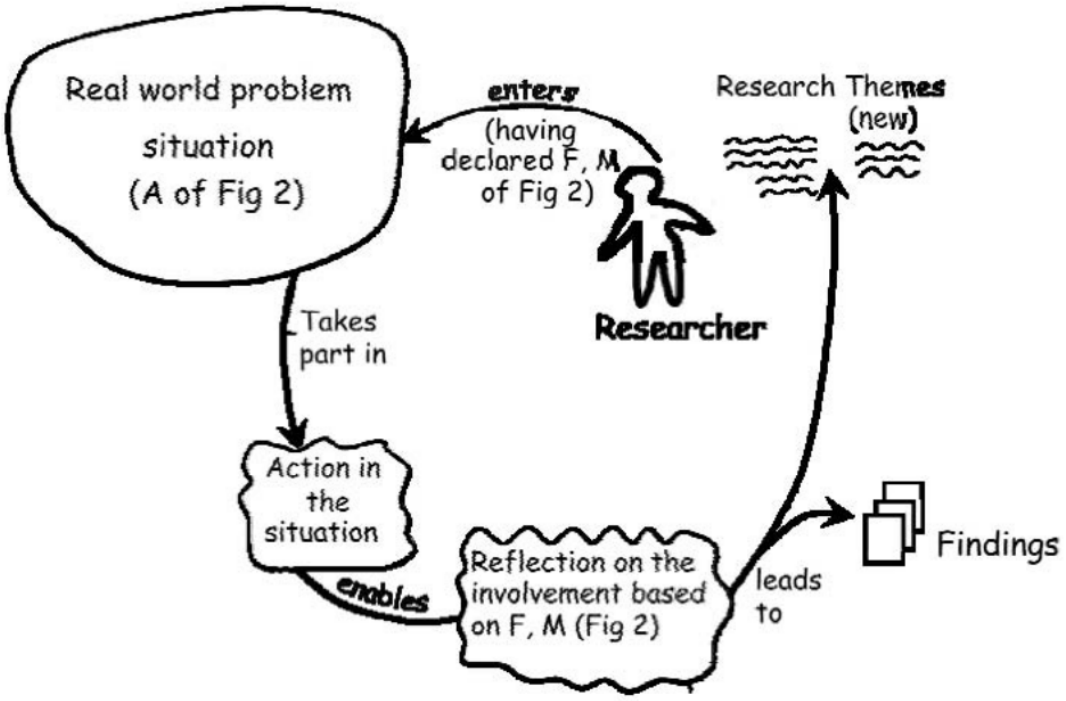


Figure 2.5: The cycle of action research in human situations, from [24].

Having defined F, M, and A we can explore what a cycle of AR looks like. As shown

in Figure 2.5, the researcher interested in particular themes (declaring F and M) enters the "social practice" of a relevant real-world situation and becomes involved both as a researcher and a participant. During and at the end of the cycle the researcher reflects on F and M leading to findings that will evolve the research themes and improve F and M.

2.2.1 Action Research in Software Engineering

This thesis was not the first case of using Action Research in software engineering, Staron [26] presents in detail the usage of Action Research in this scenario with multiple examples. Furthermore, Dos Santos and Travassos [27] also use Action Research and comment on its applicability for software engineering.

2.3 Grounded Theory

Charmaz [28] states that Grounded Theory (GT) methods are a logically consistent set of data collection and analytic procedures aimed to develop theory. Grounded Theory methods provide systematic procedures for shaping and handling rich qualitative materials, although they may also be applied to quantitative data. Charmaz states that the distinguishing characteristics of Grounded Theory methods include the following:

1. Simultaneous involvement in data collection and analysis phases of research.
2. Creation of analytic codes and categories developed from data, not from preconceived hypotheses.
3. The development of middle-range theories to explain behavior and processes.
4. Memo-making, as writing analytic notes to explicate and fill out categories, the crucial intermediate step between coding data and writing first drafts of papers.
5. Theoretical sampling, as sampling for theory construction, not for representativeness of a given population, to check and refine the analyst's emerging conceptual categories.
6. Delay of the literature review.

According to Kathy Charmaz [29], Grounded Theory is defined as a general methodology with systematic guidelines for gathering and analyzing data to generate intermediate

theory. The name grounded theory refers to the premise that researchers can and should develop theory from a rigorous analysis of empirical data. The analytic process consists of coding data, developing, checking, and integrating theoretical categories; and writing analytic narratives throughout inquiry. Glaser and Strauss [30], the originators of grounded theory, first proposed that researchers should engage in simultaneous data collection and analysis, which has become a routine practice in qualitative research. From the beginning of the research process, the researcher codes the data, compares data and codes, and identifies analytic leads and tentative categories to develop through further data collection. A grounded theory of a studied topic starts with concrete data and ends with rendering them in an explanatory theory.

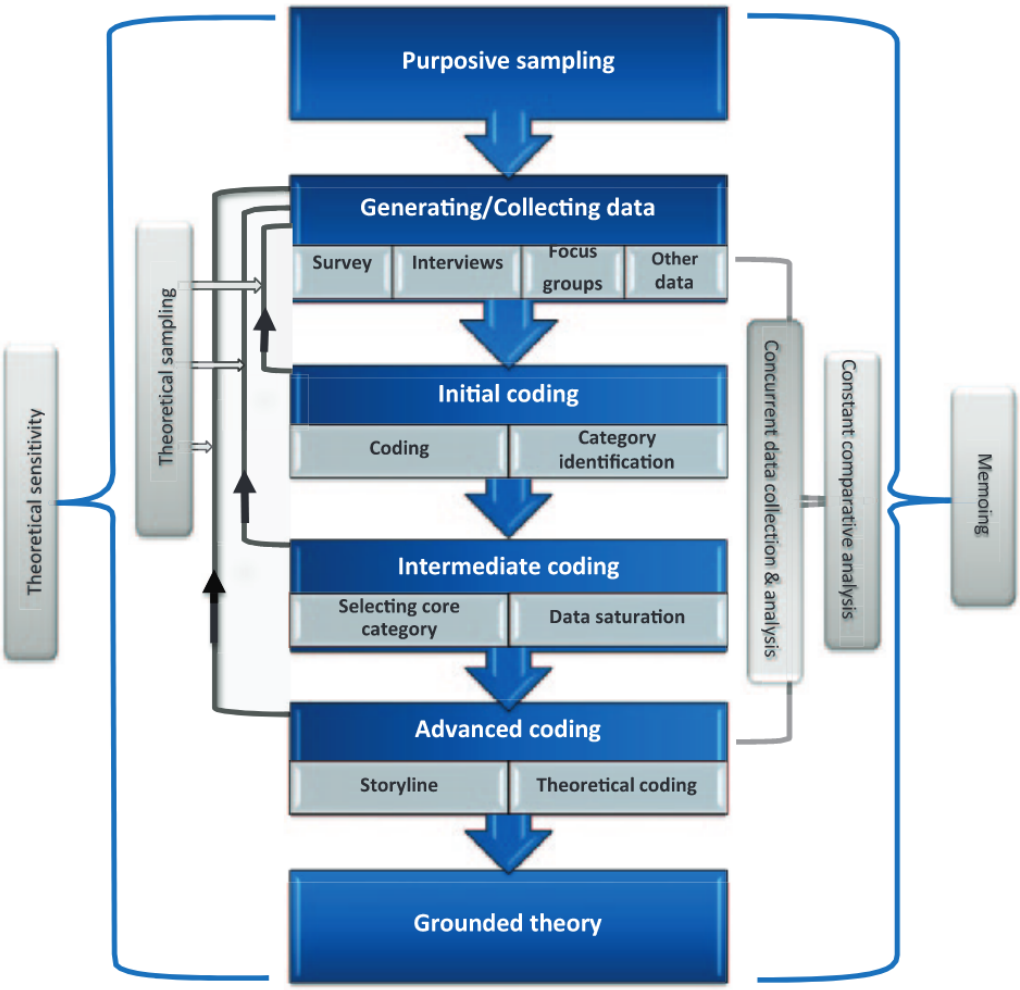


Figure 2.6: Research design framework summary from "Grounded theory research: A design framework for novice researchers" [31]. Shows the interplay between the essential grounded theory methods and processes.

Chun Tie, Birks, and Francis [31] state that a Grounded Theory (GT) research study is not linear; it is iterative and recursive, and that it follows the summarized steps below. The reader may use Figure 2.6 to follow up the explanations.

Purposive sampling: initial purposive sampling directs the collection and/or generation of data. Researchers purposively select participants and/or data sources that can answer the research question [30] [32] [33] [34]. Concurrent data generation and/or data collection and analysis is fundamental to GT research design. The researcher collects, codes, and analyzes this initial data before further data collection and generation is undertaken. Purposeful sampling provides the initial data that the researcher analyzes [31].

Constant comparative analysis: Constant comparative analysis is an analytical process used in GT for coding and category development. This process commences with the first data generated or collected and pervades the research process. Incidents are identified in the data and coded [35].

Memoing: Memo writing is an analytic process considered essential in ensuring quality in grounded theory [35]. Memos are the storehouse of ideas generated and documented through interacting with data [36]. Thus, memos are reflective interpretive pieces that build a historical audit trail to document ideas, events, and the thought processes inherent in the research process and developing thinking of the analyst [35].

Generating/Collecting data: A hallmark of GT is concurrent data generation/collection and analysis [31]. While interviews are a common method of generating data, data sources can include focus groups, questionnaires, surveys, transcripts, letters, government reports, documents, grey literature, music, artifacts, videos, blogs, and memos [37].

Coding: Coding is an analytical process used to identify concepts, similarities, and conceptual reoccurrences in data. Coding is the pivotal link between collecting or generating data and developing a theory that explains the data. In GT, coding can be categorized into iterative phases. For example, constructivist grounded theorists refer to initial, focused, and theoretical coding [37].

Initial coding: Initial coding of data is the preliminary step in GT data analysis [35] [37]. The purpose of initial coding is to start the process of fracturing the data to compare incident to incident and to look for similarities and differences in beginning patterns in the data. Important words or groups of words are identified and labeled. In GT, codes identify social and psychological processes and actions as opposed to themes [31]. Charmaz [33] emphasizes keeping codes as similar to the data as possible and

advocates embedding actions in the codes in an iterative coding process. During initial coding, it is important to ask 'what is this data a study of' [38]. 'What does the data assume, suggest or pronounce' and 'from whose point of view' does this data come, 'whom does it represent or whose thoughts are they?'. 'What collectively might it represent?' [33]. The process of documenting reactions, emotions, and related actions enables researchers to explore, challenge, and intensify their sensitivity to the data [39].

Theoretical sampling: The purpose of theoretical sampling is to allow the researcher to follow leads in the data by sampling new participants or material that provides relevant information [31]. As depicted in Figure 2.6, theoretical sampling is central to GT design, aids the evolving theory [30] [32] [33] and ensures the final developed theory is grounded in the data [37]. Birks and Mills [35] define theoretical sampling as the process of identifying and pursuing clues that arise during analysis in a grounded theory study.

Intermediate coding: Intermediate coding, identifying a core category, theoretical data saturation, constant comparative analysis, theoretical sensitivity and memoing occur in the next phase of the GT process [35]. Intermediate coding builds on the initial coding phase. Where initial coding fractures the data, intermediate coding begins to transform basic data into more abstract concepts allowing the theory to emerge from the data [31].

Advanced coding: Birks and Mills [35] described advanced coding as the techniques used to facilitate the integration of the final grounded theory. These authors promote storyline technique and theoretical coding as strategies for advancing analysis and theoretical integration [31]. Storyline is a tool that can be used for theoretical integration. Birks and Mills [35] define storyline as a strategy for facilitating integration, construction, formulation, and presentation of research findings through the production of a coherent grounded theory. This procedure builds a story that connects the categories and produces a discursive set of theoretical propositions [36].

Theoretical sensitivity: is the ability to know when you identify a data segment that is important to your theory [30]. Conducting GT research requires a balance between keeping an open mind and the ability to identify elements of theoretical significance during data generation and/or collection and data analysis [35].

2.3.1 Action Research and Grounded Theory Together

Relevant for our methodology explained in chapter 4, there are similarities and differences between the Action Research (AR) and GT methodologies. However, they can be

used together. Dick [40] comments on some of the differences and similarities. For example, GT tends not to be participative; the action tends to be someone else's responsibility, while AR tends to be participative. Furthermore, Dick points to some similarities, as both AR and GT are emergent, and the understanding and the research process are shaped incrementally through an iterative process. Moreover, data analysis, interpretation, and theory-building occur simultaneously as data collection.

Moreover, Dick [40] proposes using them together, showing that elements of action research can be embedded in a grounded theory study, with Lingard, Albert, and Levinson [41] and Abdel-Fattah [42] sharing this concept.

Finally, Staron [26] agrees explaining that other research methodologies can be combined with Action Research.

2.4 Software Development Methodologies

A Software Development Methodology (SDM) in software engineering is a framework that is used to structure, plan, and control the process of developing an information system. We can find similar definitions on the web with the term being used in about 17,000 publications, as found in academic search engines like Google Scholar.

2.4.1 History of Software Development Methodologies

Based on Larman and Basili's work [43], a brief history of SDMs is presented here.

By the late 1950s, structured programming appeared with ALGOL 58 and ALGOL 60 programming languages, improving development time, clarity, and quality by making extensive use of block structures, subroutines, and loops [44]. Subsequently, around the decades of 1960 to 1970, depending on the source, the Waterfall model [45] arose characterized by a sequential (non-iterative) process which flows downwards through its phases, moving from phase to phase only when the previous phase is reviewed and verified. In the 1970s, "Top-Down Programming in Large Systems" [46] appeared promoting iterative and incremental development. Following, in the early 1980s, the concept of evolutionary prototyping was discussed and applied [47], consisting of creating prototypes of software applications for users to evaluate the design and revise to enhance the prototype. In the late 1980s, the Spiral model [48], a risk-driven process model generator for software projects, was described. Then, the V model appeared as an extension of the waterfall model where the process phases are repeated upwards to form a V shape [49].

At this point, an end of an era of predictive approaches is reached, and a new era of adaptive approaches starts. By the 1990s, Rapid Application Development (RAD) was defined, showcasing less emphasis on planning and more on process, adaptability, and the necessity to adjust requirements [50]. During the 1990s and 2000s, Agile methods rise implementing adaptive planning, evolutionary development, early delivery, and continuous improvement while encouraging rapid and flexible response to change [51]. In 1995 the Dynamic Systems Development Method appeared, stemming from RAD, with core techniques like time boxing, MoSCoW prioritization[3], prototyping, testing, workshops, modeling, configuration management, and characteristics like being iterative and incremental, being architecture-centric, and risk-focused [52]. Scrum appears in 1995, enabling teams to self-organize by encouraging physical face-to-face communication and collaboration of all team members [53]. Further Agile methodologies appear in 1996 like Crystal [54], a lightweight and adaptable approach with a specific custom set of policies, practices, and processes based on unique characteristics of the team and the project; and Extreme Programming (XP), which advocates for frequent releases in short development cycles, introducing checkpoints for requirements adoption [55]. In 1999 Feature-Driven Development appears to deliver tangible and working software repeatedly in a time [56]. Moreover, in 2001 the Agile manifesto [57] was published as seventeen software developers met to discuss lightweight development methods and wrote it. From there, in the 2000s Agile Unified Process appeared applying agile techniques, including test-driven development (TDD), agile modeling, agile change management, and database refactoring to improve productivity [58]. Moreover, Disciplined Agile Delivery was born in the late 2000s and first described in 2012, a process decision framework that enables simplified process decisions around incremental and iterative solution delivery [59]. Furthermore, in the 2010s, Large-Scale Scrum (LeSS) is described as to deal with the usage of Scrum in multi-team large-scale development [60]. Following the same trend, Scaled Agile Framework (SAFe) appears consisting of a framework with a knowledge-base of integrated patterns intended for enterprise-scale Lean-Agile development [61].

Additionally, the Agile umbrella of methodologies covers lightweight approaches like: Lean software development [62], Kanban [63], Continuous Integration (CI) [64], Continuous Delivery (CD) [65], Test Driven Development (TDD) [66], and Crystal Clear [67]. Moreover, on multi-team approaches it covers approaches like: Scrum-of-Scrums [68], and Scrum at Scale (Scrum@Scale) [69].

In summary, SDMs have been evolving into a more *agile* style to deal with constantly

[3]MoSCow stands for prioritization by Must have, Should have, Could have, and Won't have (this time).

changing requirements. Decades of experience in software development brought to the so-called Agile manifesto [57], which we reproduce here as to understand it further:

> "We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
>
> - Individuals and interactions over processes and tools
> - Working software over comprehensive documentation
> - collaboration over contract negotiation
> - Responding to change over following a plan
>
> That is, while there is value in the items on the right, we value the items on the left more."

Furthermore, from the manifesto, its 12 principles:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity–the art of maximizing the amount of work not done–is essential.

11. The best architectures, requirements, and designs emerge from self-organizing teams.

12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

The spirit of these principles is important for this dissertation. They will be discussed further in the context of the project, however, applying them requires some adaptation or reinterpretation.

Understanding what Scrum and Extreme Programming are about helps in understanding this dissertation; hence, they are described below.

2.4.2 Scrum

Scrum is a framework used in software development for project management, emphasizing teamwork, accountability, and iterative progress towards defined goals. Variations of Scrum exist, and every team adapts the framework for their needs. Schwaber and Beedle [70] describe Scrum as follows.

A Scrum team is usually composed of five to nine people. There are two key figures: the product owner, which represents users, customers or other stakeholders, and the Scrum master, which takes care of managing the Scrum process to make the team as productive as possible. These can be members of the team or external. There is a product backlog, which is a prioritized list of features to develop and a sprint backlog that contains the list of tasks to be completed in a sprint. During sprint planning meetings, which happen at the start of every sprint cycle, the product owner proposes the next items to work from the product backlog, and the team selects the items they can complete in that sprint. Then, these items are moved to the sprint backlog to be worked on. Every day during the sprint cycle, a daily Scrum meeting is held; this is a brief meeting where every member sets the context for that day. All teams are expected to attend, and it is usual to hold these as stand-up meetings as it helps to keep the meetings short. At the end of each sprint, a review meeting is held where the team demonstrates the completed functionality and what was accomplished during the sprint. A sprint retrospective is also held at the end of the cycle, and it is used to reflect on how well Scrum is working for the team and what changes may be done to improve further the process.

2.4.2.1 Sprints

The concept of sprints from the Scrum SDM is used in this manuscript; familiarity with its structure is useful to follow discussion where it is mentioned. We describe further sprints, based on Schwaber and Beedle [70], as follows.

The iteration cycle in Scrum is called a sprint. At the start of the cycle, a planning meeting is held where the team creates tasks, time, and resource estimates for these tasks and volunteers for these. Then, a cycle of a pre-defined length, usually one to four weeks, starts, where analysis, design, code, and testing is done. Every day of the sprint, a daily stand-up meeting is held where the team members keep up to date and help each other resolve problems. The sprint backlog is updated accordingly as goals are met. At the end of the cycle, a sprint review is held where a demonstration of complete, thoroughly tested, and potentially shippable features is done. Further discussion about the completed work is encouraged. Finally, a sprint retrospective meeting is held to inspect and adapt the processes and tools used during the sprint cycle.

2.4.3 Extreme Programming

Extreme programming (XP) takes the best practices of software development to an extreme level. Created in 1996 by Beck [71], the principles of Extreme Programming were initially described in his 1999 book, Extreme Programming Explained [72]. Based on the current description of the official Agile Alliance website of Extreme Programming [73] which itself is written based on Kent Beck's and Don Wells's[4] descriptions, Extreme Programming focuses on customer satisfaction by developing features when the customer needs them. The team is self-organized, and it deals with new requests as part of their daily routine. The methodology is considered applicable in projects with dynamically changing software requirements, risks caused by fixed time projects using new technologies, small and co-located extended development team, and the technology to be used, allowing for automated unit and functional testing.

Extreme Programming has five core values: communication, simplicity, feedback, courage, and respect. Furthermore, it has a core set of interconnected practices: the planning game[5], small releases, metaphors[6], simple design, testing, refactoring, pair

[4] Don Wells published the first version of Extreme Programming on his website about XP while working with Kent Beck.

[5] The game is a meeting that occurs typically once per week intending to guide the product into delivery, it is divided into release planning and iteration planning.

[6] Metaphors are human-friendly stories that anyone can use to understand how a system works. They consist in naming schemes for classes and methods that make it easy for team members to follow the

programming, collective ownership[7], continuous integration[8], 40-hour week, on-site customer, and coding standards.

This technique has two leading roles: the customer and the developers. However, it describes a couple of secondary roles: the tracker, keeping track of relevant metrics, and the coach, an outside consultant, to help getting started in XP.

XP's life cycle consists of first describing the desired results of the project by having customers define a set of stories. As these stories are being created, the team estimates the size of each story. This size estimate, along with relative benefit, as estimated by the customer, can indicate the relative value that the customer can use to determine the priorities of the stories. If the team identifies stories that they are unable to estimate because they do not understand all of the technical considerations involved, they can introduce a spike to do some focused research on that particular story or a common aspect of multiple stories. Spikes are short, time-boxed time frames set aside for the purposes of researching a particular aspect of the project. Spikes can occur before regular iterations start or alongside ongoing iterations. Next, the entire team gets together to create a release plan that everyone feels is reasonable. This release plan is a first pass at what stories will be delivered in a particular quarter or release. The stories delivered should be based on what value they provide and considerations about how various stories support each other. Then the team launches into a series of weekly cycles. At the beginning of each weekly cycle, the team (including the customer) gets together to decide which stories will be realized during that week. The team then breaks those stories into tasks to be completed within that week. At the end of the week, the team and customer review progress to date, and the customer can decide whether the project should continue, or if sufficient value has been delivered.

2.5 Software Development Methodologies in Robotics Competitions

The literature on similar works as this dissertation, embracing the software development process for a robotics competition, is limited. We can find works on implementing well known SDMs for companies in different situations like Scrum [74] [75], Extreme

functionality of a certain element of the codebase by its name only.

[7]Everyone is responsible for all the code; everyone is allowed to change any part of the code.

[8]A practice to avoid integration problems where the team is always working with the latest version of the software.

Programming [76], or generalizing about Agile [77], and even systematic reviews of empirical studies on Agile software development [78]. These studies follow AR models but work with a group of participants with a different profile and a different goal than the one presented in this dissertation.

On the other hand, Van der Zant, co-founder of RoboCup@Home, and Plöger [79] present the application of Extreme Programming (XP) for RoboCup development. They indicate promising results applying practices from the methodology in RoboCup Mid-Size and Aibo league[9]. The paper focuses on the problem of continuously changing rules and team members and how to tackle it via the usage of XP. They mention the usage of test case driven design, simple design, collective code ownership, refactoring, pair programming, short releases, continuous integration, round-trip engineering, and active project management. Additionally, they advocate for standardized hardware and software with tools to ease complex or tiresome tasks like installing software in the robots and their development software. Furthermore, adding to the simple design concept, they use extreme modularity with rigidly defined interfaces and strong typing (via C++). They also report experts doing project steering, not project management, to keep the project in track. Moreover, they estimate that it took them three times less time to achieve the same results compared with previous projects.

Van der Zant and Plöger's work is closely related to our work as we also aim to experiment with practices from Agile SDMs, and it is one of the few publications directly related with RoboCup, even though for different leagues.

Furthermore, Gerndt, Schiering, and Lüssem [80] present the application of Scrum for RoboCup development. They apply practices from this methodology for a RoboCup kid-size humanoid competition team and a RoboCup@Work team. They showcase a teaching philosophy where they allow students to solve problems by themselves, including letting them self-organize and explain how the students analyze and propose the usage of SDMs as Waterfall, V-model, Spiral, and Scrum, finally deciding to try Scrum. The authors conclude that a project management methodology for self-organized teams of students in robotics projects should address easy estimation of work packages, innovation-oriented flexible project planning, quality checks and testing, team building, and transparency. They note that the values of the agile manifesto are well suited for this goal. Moreover, they indicate the usage of two weeks long Sprint cycles with the students creating their own user stories. Fruitful discussions are reported in the Sprint planning meetings, although they add that user stories were often not finished in one cycle. However,

[9]RoboCup Standard Platform League for playing soccer was done with Aibo dog robots at the time.

they describe moving to weekly meetings when the competition was close, not being able to fulfill the product owner role, as per the Scrum methodology, and that Scrum's rigorousness is problematic. Furthermore, they describe using continuous integration, coding standards, using robot simulations, and using the Redmine[10] ticketing tool as a work-tracking board, with a dedicated monitor in the lab showing the state of the project. Finally, the students' feedback about this practice was positive, including an appreciation for transparency and communication and reporting that regular meetings created team spirit.

Finally, other authors that mention the usage of SDMs for RoboCup related projects are Tomatis, Philippsen, Jensen, Arras, Terrien, Piguet, & Siegwart, [81] mention shortly the usage of Extreme Programming, Gerndt, Paetzel, Baltes, & Ly [82] reference the agile manifesto as an inspiration, Paraschos, Spanoudakis, & Lagoudakis [83] explain their approach using Agent-Oriented Software Engineering, and Eaton [84] mentions the need of using some Agile SDM. However, these works do not explore these concepts in great detail.

2.5.1 Team Description Papers: What is Missing

While Agile software methodologies are widely used in business settings, there does not appear to be a significant amount of reports of Agile methodologies in RoboCup.

To examine the use of software development methodologies in RoboCup, we conducted a systematic survey of published Team Description Papers (TDPs) from the RoboCup@Home leagues. Participating teams are required to submit and publish TDPs when they apply to participate in the competition. The survey began by downloading the 47 available TDPs covering from the year 2011 to 2019. A manual review of the TDPs revealed no discussion of methodologies. This was followed by an automated search for keywords related to this topic, which found ten results, but an examination of each result found no matching documents except UTS Unleashed! (i.e., the team represented in this manuscript). The keywords used for the automated search were, in between others: software development methodology, scrum, extreme programming, agile, waterfall, spiral, project manager, meeting, proactive, organization, test driven, testing, simulation, training, workshop.

[10]Redmine is a free and open-source, web-based project management and issue tracking tool.

2.6 Proactivity

In this manuscript, the concept of proactivity appears concerning team members' attitude in regards to software development. It is meaningful to convey a definition of proactivity before using this concept. The Oxford English Dictionary defines proactive as:

> "(of a person or policy) controlling a situation by making things happen rather than waiting for things to happen and then reacting to them."

Bindl and Parker " [85] further add:

> "Proactive behavior at work is about making things happen. It involves self-initiated, anticipatory action aimed at changing either the situation or oneself. Examples include taking charge to improve work methods, proactive problem solving, using personal initiative, making ideals, and proactive feedback seeking."

Subsequently, proactivity is, arguably, a behavior that is advantageous to have in a team, in particular, in teams working with projects with changing requirements, as is the case in this work.

CHAPTER 3

CONTEXT

In the following chapters, decisions will be shown around the common tools and common experiences to be found in the project. In retrospect, after the project was done, it is easy to discern these commonalities and explain them beforehand so the reader finds them for themselves in the proper context.

These facts shaped the planning and development decisions made during the three years of the project and are useful to keep in mind. Some of these elements have the reasoning behind their choice specified here.

3.1 Common Technical Tools

The following tools were used during the three years of RoboCup@Home Social Standard Platform League (SSPL). Their extended use in the communities related to the competition makes one believe that there is value in these tools. Also, its extensive usage in the relevant industries gives the team members familiar with them a useful skillset.

Version control systems are an essential part of software development. The 2018 Stack Overflow Developer Survey reports that only 3.7% of professional developers do not use some version control system. Git is the most popular system: used by 88.4% of professional developers[86].

From the official git website[1] git is described as:

[1] https://git-scm.com/

> "Git is a free and open-source distributed version control system[2] designed to handle everything from small to very large projects with speed and efficiency.
>
> Git is easy to learn and has a tiny footprint with lightning-fast performance. It outclasses SCM[3] tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows."

GitLab is a web-based DevOps[4] lifecycle tool. It provides git repository management, including wiki, issue-tracking, and Continuous Integration/Continuous Deployment pipeline features. One of the most widely used platforms of its kind (alternatives being GitHub and Bitbucket) GitLab[5] it provides an open-source DevOps platform as a website application offering a solution for teams to collaborate in software development in a user-friendly manner.

3.1.1 Asana

Asana is a web application that features Task management (with List and Board views), a Timeline, Calendar, a Progress tracker as well as other tools. A screenshot can be seen in Figure 3.1.

[2]Version control is also known as source code control, source code management system, or revision control system. It is responsible for tracking and managing changes to files, documents, computer programs, and other stored information.

[3]Software Configuration Management.

[4]DevOps is a set of practices that combines software development and information-technology operations which aims to shorten the systems development life cycle and provide continuous delivery with high software quality. [87]

[5]https://gitlab.com

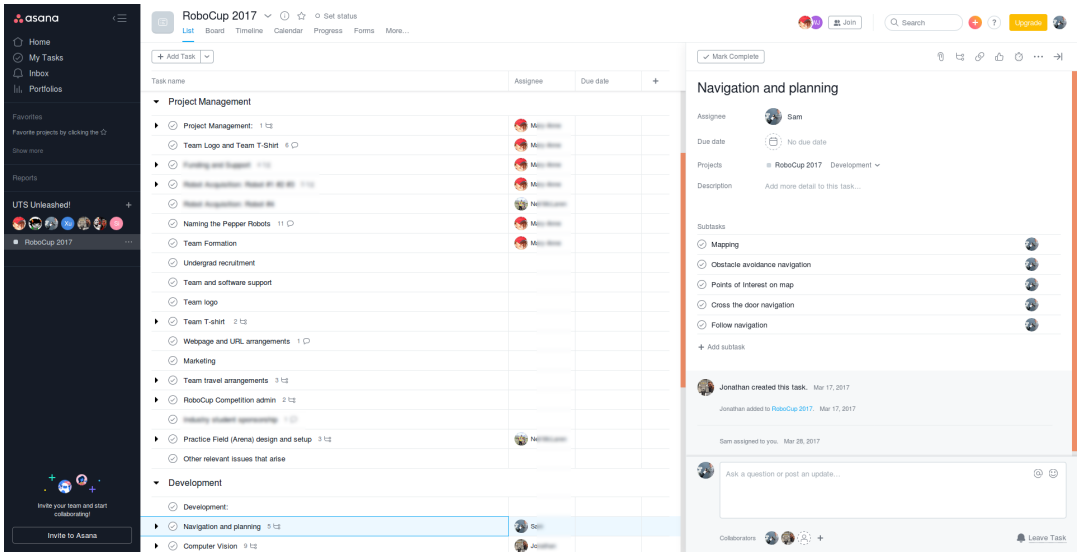


Figure 3.1: Asana screenshot of the 2017 RoboCup project.

3.1.2 Trello

Trello is a collaboration tool to organize projects into boards, representing a whiteboard filled with lists of sticky notes, each one representing a task to be done. It is a web-based application. It advertises itself as a “Kanban-style” list-making application. A screenshot can be seen in Figure 3.2. It provides a calendar view for deadlines, and many extensions are available to add functionalities to the platform.

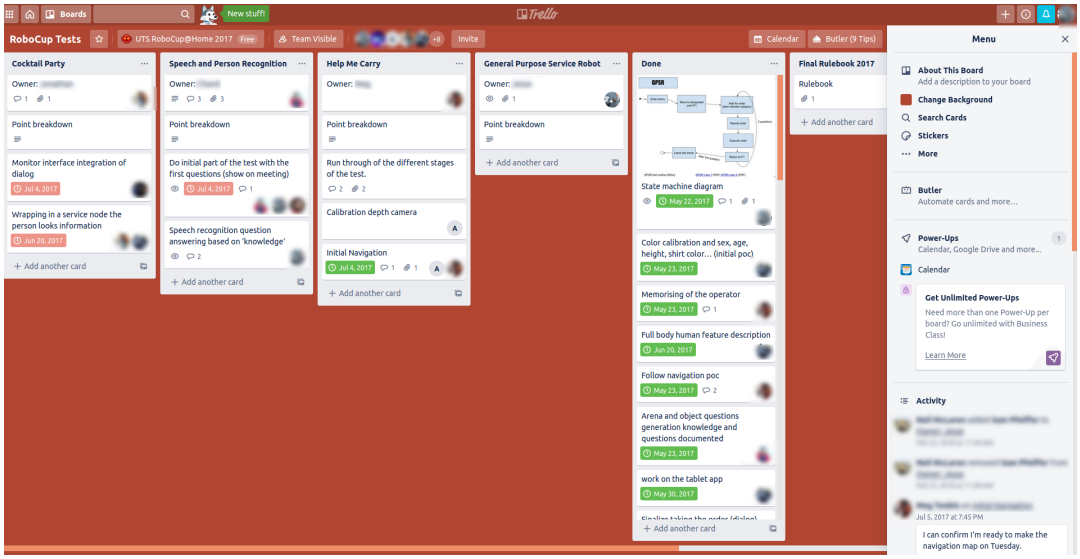


Figure 3.2: Trello screenshot of the 2017 RoboCup project.

3.1.3 Slack

Slack is a chat platform designed to replace email as a primary method of communication. It features workspaces, different rooms for group discussions, and private messages to be sent in between members. It allows sharing different kind of files easily. It is widely used in industry.

3.1.4 Python: Programming language of choice

The team considered the choice of the main programming language important. Python was chosen.

At the start of the project, the lead developer did an informal research that summarizes Python's advantages for this project:

- Accessible, easy to read, and write, which makes it easy to learn.
- Great libraries, standard library, and third-party.
- Healthy, active, and supportive community.
- Used by lots of people in science, industry & academia.
- Fast (optimized and using C/C++ based libraries).

All these characteristics of the language fit well with a team with many members that want to collaborate but do not have a strong coding background. Furthermore, the experts in the team had experience in the language. Also, the Pepper platform had Python as its most supported language and the Robotics Operating System (ROS) framework too. Plenty of state of the art deep learning frameworks also use Python, and it was foreseen that the team might end up using some of them.

Additionally, answers to the question *On what do people use Python?* in our context were found:

- On prototyping.
- When the speed of development matters more than the speed of execution.
- On problems where you can find libraries readily available for them.

While using a language that eases prototyping and speed of development fits well with RoboCup development, the option to use other languages remains open.

3.1.5 C/C++: Backup programming languages

People working in robotics tend to often use C++ and Python [88]. One of the reasons for this is because both are supported in the ROS framework. C++ is used for low-level control and image processing, for example. Python is used for more high-level behaviors or testing and prototyping.

During the years of development, some parts were C++, and some parts were sped up by using C or C++ exposing interfaces for Python.

C++ was not considered the main language because the learning curve is steeper, and there were complications in deploying the code on the robots (as the robots used a different CPU architecture which involved cross-compiling).

3.1.6 ROS: Robotics framework/middleware

Robotics Operating System (ROS) [89]: Developed since 2007, it is the most famous and possibly the most used robotics framework (some people consider it rather a middleware; however, both definitions can fit ROS' goals). Their authors define ROS and its reasons to exist as follows:

> "The Robot Operating System ROS is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms.
>
> Why? Because creating truly robust, general-purpose robot software is hard. From the robot's perspective, problems that seem trivial to humans often vary wildly between instances of tasks and environments. Dealing with these variations is so hard that no single individual, laboratory, or institution can hope to do it on their own.
>
> As a result, ROS was built from the ground up to encourage collaborative robotics software development. For example, one laboratory might have experts in mapping indoor environments, and could contribute a world-class system for producing maps. Another group might have experts at using maps to navigate, and yet another group might have discovered a computer vision approach that works well for recognizing small objects in clutter. ROS was designed specifically for groups like these to collaborate and build upon each other's work, as is described throughout this site."

Figure 3.3: The ROS equation as advertised in the official website.

ROS presents the following features that appeal to the goals of this project:

- A distributed, modular design.
- A vibrant community.
- Permissive licensing.
- A collaborative environment.
- Ready to use tools, libraries, drivers, and algorithms.
- Widely used in academia and industry.

Also, as mentioned in the previous sections, programming in the ROS framework can be done from a variety of languages. Python and C++ are first-class citizens.

Other Robotics frameworks were considered, some of them used by other RoboCup teams like Fawkes [90], or used in industry like Orocos [91]. However, given the familiarity with ROS of the author of this dissertation, and the many users of ROS in the RoboCup competition, ROS was chosen.

3.1.7 NaoQi: Programming framework for the Pepper robot

The manufacturers of the Pepper robot created the NaoQi framework. It is officially described as:

> "The NAOqi Framework is the programming framework used to program Aldebaran robots.
>
> It answers to common robotics needs including: parallelism, resources, synchronization, events.
>
> This framework allows homogeneous communication between different modules (motion, audio, video), homogeneous programming and homogeneous information sharing.

> NAOqi framework is Cross platform and Cross language that allows you to create Distributed applications.” [92]

Given that the Pepper robot uses this framework and APIs, any user of the robot is forced to use it in some way.

3.1.8 Operating System

It was decided to use Ubuntu 16.04 as the development platform (in the year 2017, this was a mature and current platform). Ubuntu is a free and open-source Linux distribution based on Debian. Other RoboCup@Home teams tend to use it[6] as it is a Linux variant friendly for beginners, as some of the team members were. A lot of software and frameworks are usually aiming to deploy in this system so we can hope for a smooth experience using the bug-free state of the art software. One of these frameworks is ROS.

The Pepper robot uses a custom Gentoo Linux. The lead developer thought that Gentoo Linux was too complicated to introduce people to, and to support different laptop hardware. Using Ubuntu, being another Linux system, would allow team members to more easily familiarise themselves with the available tools.

3.2 Common Experiences

The following facts were mostly discovered in the first year of participation, while some of them were known beforehand due to the previous participation in RoboCup@Home by the author of this manuscript. This section starts by presenting the RoboCup@Home SSPL rulebook, then its standard platform: Pepper. It continues with important facts related to the development of the competition: rental of robots for the competition, deadlines for the event, WIFI quality at the venue, and audio quality at the venue.

3.2.1 Commonalities of the RoboCup@Home SSPL rulebooks

The rulebook follows the same structure year by year. The rulebook in itself explains in detail the philosophy of the competition, the qualification process, a description of the venue, what is expected from the teams that participate (and their robots), and the rules and regulations. Also, detailed descriptions of the *tests* that the robots must perform

[6] Some references to Ubuntu were found in RoboCup@Home Team Description Papers.

autonomously are found. The scoring system of these tests is appended to the description of each test.

In the year 2017, the SSPL was introduced. It was unknown at the time by the RoboCup@Home organizers how this league and its standard platform would perform. To approach this situation, the rulebook contained tests to be performed exclusively in SSPL. This was due to the perceived lack of ability of the standard platform to perform manipulation actions, which most other tests in the competition contain at least as a partial task to accomplish.

A summary of the structure and contents of the common parts of the rulebook follows.

The rulebook starts with an *Introduction* section explaining RoboCup, RoboCup@Home, its organization, infrastructure, the leagues of the competition (new this year), a brief explanation of the competition itself and the available awards.

A section titled *Concepts behind the competition*, which, among other things, highlights the importance of concepts like *Autonomy and mobility* for the robots and *Scientific value* of the competition.

General Rules & Regulations covers the team registration and qualification process. The scenario or arena where the competition will happen follows. It describes a household-like setup, as seen in Figure 3.4, with low height walls as separators of rooms but allowing visibility of what is happening in the arena. See Figure 3.5 for an example. The scenario hosts furniture found in an average household of the country that hosts the competition.

Figure 3.4: The scenario map for Nagoya, Japan, in 2017 as provided by the organizers. The numbered items are designated places that may be referenced in the competition tests. For example, the bed, the kitchen table, or the TV. The organizers provide this map, but it is not precise. It roughly marks where the different items should be. During the competition, things move around; for example, people may sit in chairs and leave the chair in a different position. Furniture may be gently pushed inadvertently. The robots must be able to deal with such changes in the environment.

Figure 3.5: The arena in Nagoya, Japan, in 2017 with the Pepper robot. Taken from the bedroom and seeing through thanks to the low height walls into the kitchen and living room. There is a functional door on the right. Standard furniture as cabinets, a sink, a fridge, a table, and chairs can be seen. Other small objects can be appreciated like a jar of tea, some pasta in a container, an orange, and children's books. As a curiosity, it can be appreciated how the Pepper robot height and arm's length can make manipulation tasks challenging as most table-like surfaces stand quite high for it.

The next section is *Robots* and describes what is expected of the robots participating.

From here, the rulebook may change every year. For example, topics that may change every year are: *External Computing* (whether a team can use a computing device that's not part of the robot), the organization of the competition, its schedule, tests for the robots or the scoring system.

These changes were minor from 2017 to 2018 but major from 2018 to 2019. These changes are discussed in chapter 5 on the analysis of every year. A big part of the rulebook is still common in nature; these commonalities have been extracted from an analysis of the rulebook and are explained here:

- The necessary skills that a robot must be able to perform autonomously. A non-

exhaustive list:

- Text To Speech (TTS), e.g., that the robot must be able to speak.
- Automatic Speech Recognition (ASR), sometimes just referred to as Speech To Text (STT), e.g., that the robot must understand spoken commands.
- Sound Source Localization (SSL), e.g., the robot must know where a sound or speech came from.
- Natural Language Processing (NLP), e.g., understanding open speech of a user.
- Autonomous Navigation, e.g., that the robot must be able to navigate autonomously from point A to point B. To do so, it, most probably, needs to know where it is (Localization).
  - This navigation must be performed safely, collision with the environment may not happen.
  - Navigation may happen in a previously known environment (having a 'map' constructed previously).
  - Or in an unknown environment, where usually Simultaneous Localization And Mapping (SLAM) is used.
  - Some tests may ask to follow a person.
  - Some tests may ask to guide a person to a place.
  - There may be obstacles on the way.
  - The robot may need to pass through narrow paths like doorways.
- Perception, e.g., that the robot must be able to detect or recognize things as:
  - People detection, e.g., finding faces that may indicate there is a person there.
  - Face recognition, e.g., learn and recognize the same person by their face.
  - Gesture detection, e.g., realizing a person is waving to the robot.
  - People characteristics detection, e.g., guess the age, gender, color of clothing, height, or other features of a person.
  - Object detection and recognition, e.g., finding where a specific object is and what it is. For example, a can of coke on top of a table.
  - Where the robot and its surroundings are, e.g., in which room is the robot or where is the fridge in relation to the robot.

  - Human Robot Interaction, e.g., the robot must understand what a user wants and get more information from a user if needed. This can be done via:
    * Via TTS and ASR.
    * Via gestures.
    * Via a Tablet interface (on the chest of the Pepper robot).
    * A combination of the previous, or others.
  - Manipulation, e.g., the robot must be able to pick, grasp, drop, place, or transport some items.

- Goals of the RoboCup@Home SSPL tests. The common topics that the robot must solve in a test are:
  - Finding objects in the house and move them.
  - Taking and fulfilling spoken orders from people.
  - Be a waiter in a restaurant/cafe where it has never been before.

- Specific tests that repeat every year. They are not tests per se but are part of the competition.
  - Robot Inspection. The robot is tested for its most basic capabilities: If it can move from the entrance of the simulated home to a room while avoiding a person standing in its way and present itself. Also, testing the robot's emergency stop button should shut the robot down successfully. This test is not scored, but it must be passed to be able to participate.
  - Poster Session. Where a team member will present in a poster in 2 minutes the research of the team. Open to questions from the team leaders. Teams are encouraged to ask questions and bond. This test is scored. Every team leader scores every other team presentation.

- Competition Schedule[7]. The competition is divided into six days. The first day and a half are *setup days*, where the teams arrive with their robots, and they set up their working space and their robots. The arenas and networks are built during these days. They are often done by the end of the first day, but it is not guaranteed. Robot Inspection and Poster Session happen at the end of the second setup day. The following three days are competition days. They are divided as follows:

[7]The schedule was briefly introduced at the start of the chapter, here we expand on it.

- Stage I: a set of RoboCup@Home SSPL tests that test basic skills, expected to be easier than the ones in Stage II, are run in the first 1.5 days. There is usually a block of tests in the morning and a block of tests in the afternoon.
- Stage II: The 50% highest scoring teams from Stage I qualify to participate in Stage II. These run during the following 1.5 days. A set of RoboCup@Home SSPL Stage II tests are run. There are blocks of tests in the morning and in the afternoon.
- Finals: The last day of the competition, the two highest-scoring teams go into the Finals. The topic and specific rules every year are subject to change, but in general, it is about an open presentation of the best skills of the team's robot.

3.2.2 Pepper robot as platform

The Pepper platform uses a set of APIs to access its capabilities within a framework called NAOQi. These APIs can be accessed from Python, C++, JavaScript. NAOQi also provides some support for Java and ROS[8].

A graphical programming suite called Choregraphe is also offered to program the robot. Choregraphe internally builds Python apps that are programmed by placing blocks in a canvas and connecting them, in a similar way of popular visual programming languages like Blockly [93] or Scratch [94].

The robot's main processor is an Intel Atom E3845 with four cores and four threads at 1.91GHz with 4GB RAM. The drive capacity, being a micro SDHC card, is 16GB. The integrated graphics card is limited. The characteristics are similar to smartphones of the time[9]. Standard laptops of the time feature greater computing capabilities[10], which, other teams of other RoboCup@Home leagues embrace as they can integrate into their robot such computing platforms. In comparison, Pepper presents an extra challenge as state-of-the-art robotics software is expected to be run on relatively powerful platforms, the latest available computers.

Other hardware aspects of the platform to consider as they affect the development for the competition are the laser sensors of the robot.

[8]ROS will be further discussed shortly.

[9]Google Pixel with the Snapdragon 821 features four cores at 2.4GHz, and 4GB RAM or the iPhone SE with the Apple A9 features two cores at 1.85 GHz and 2GB RAM. Both were phones released in 2016.

[10]The Alienware R3 series with the Intel Core i7-6700HQ with four cores and eight threads at 2.6GHz to 3.5GHz and 16GB RAM. The MacBook Pro of 2016 with the Intel I7-6920HQ with four cores and eight threads at 2.9GHz to 3.8GHz and 16GB RAM. Both were laptops released in 2016. Both would also include GPUs that would allow additional computing capabilities.

The laser is composed of six laser line generators with a refresh rate of 6.25Hz. But from those, only three report actual data. As seen in Figure 3.6, each one of them has a field of view of 60°. Each one provides fifteen distance readings in the range of 0.3 to 3.0 meters. The further the reading, the noisier it is. That makes a total of forty-five laser data points around the robot with two gaps of 90° in the front diagonals and a gap of 120° on the back with no laser data.

This fact is meaningful as standard approaches for SLAM rely on dense and/or precise laser readings to perform well.

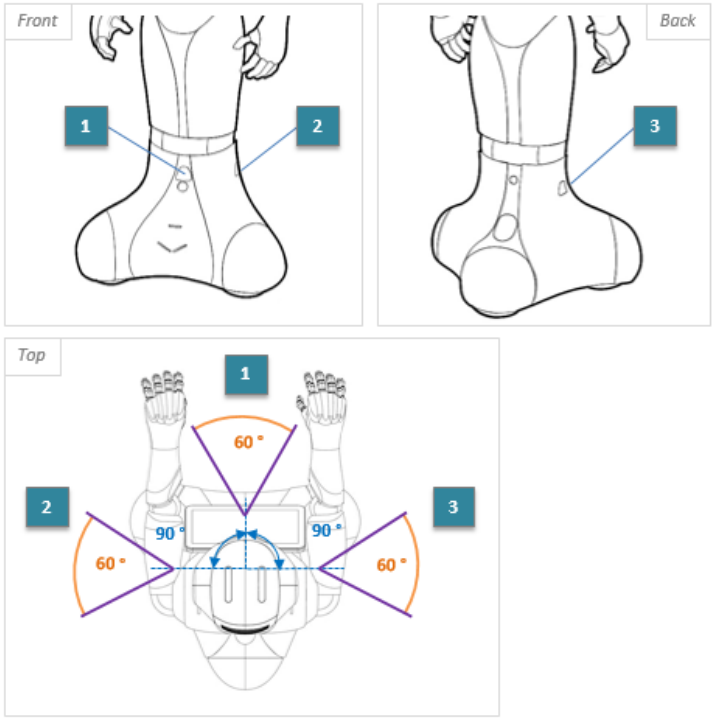


Figure 3.6: The Pepper laser sensors configuration and field of view from the manufacturer official documentation.

The Pepper robot documentation states that the robot can perform Autonomous Navigation with its default NaoQi Application Programming Interface (API)s, but its performance was not satisfactory. This is a crucial fact to keep in mind as Autonomous Navigation is a highly necessary skill for the RoboCup@Home competitions.

Furthermore the cameras presented by the robot have the characteristics shown in tables Table 3.1 and Table 3.2.

2D Cameras specifications		
Camera	Model	OV5640
	Type	CMOS image sensor SoC
Imaging Array	Resolution	5Mp
	Optical format	1/4 inch
	Active Pixels (HxV)	2592x1944
Sensitivity	Pixel size	1.4 μm * 1.4 μm
	Dynamic range	68db@8x gain
	Signal/Noise ratio (max)	36dB
	Responsivity	600 mV/Lux-sec
Output	Camera output	640*480@30fps or 2560*1920@1fps
	Data Format	YUV and RGB
	Shutter type	Rolling shutter
View	Field of view	67.4°DFOV (56.3°HFOV,43.7 °VFOV)
	Focus type	Auto focus

Table 3.1: Pepper 2D Cameras specification.

3D Camera specifications		
Camera	Model	ASUS XTION
	Type	SOC Image Sensor
Imaging Array	Optical format	1/2 inch (5:4)
	Active Pixels (HxV)	1280x1024
Sensitivity	Pixel size	5.2 μm * 5.2 μm
	Dynamic range	68.2db
	Signal/Noise ratio (max)	45dB
	Responsivity	2.1 V/Lux-sec
Output	Camera output	320*240@20fps
	Data Format	Depth color space (mm)
	Shutter type	Electronic Rolling shutter (ERS)
View	Field of view	70.0°DFOV (58.0°HFOV,45.0 °VFOV)
	Focus range	40cm - 8m
	Focus type	Fixed focus

Table 3.2: Pepper 3D Camera specification.

3.2.3 Rental of robots at RoboCup event

Teams of the RoboCup@Home Social Standard Platform League had the option to rent a Pepper robot at the venue instead of shipping their robots. This option presented teams with several advantages:

- It avoids the chance of issues with shipping robots.
  - Delays in receiving the robots.
  - Damage of the robots on the shipping to or from the competition.
  - Logistic issues in a foreign country.
- Maximum availability of the robots as they never leave the lab.
- It is cheaper to rent than to ship in most cases.
- If the manufacturer has issues with their shipping, all teams renting will face the same problem.
- Possibility to quickly swap a faulty robot in the competition with the manufacturer.

Given these advantages, during the two first years of competition, two Pepper robots were rented for the competition. For the last year, the competition was hosted in the team's city, Sydney, in a venue very close to the lab, so the team was able to use its own robots without renting or shipping.

The only disadvantage found on renting robots was the necessity of deploying and testing the rented robots when received.

3.2.4 Deadline for the RoboCup event

Every year the competition took place between June and July. The rulebook is developed during the year, and it was ready about two months before the competition. This makes it so that the team has a similar schedule every year to participate.

3.2.5 Poor WIFI connection at RoboCup event

An official networking team from the RoboCup organization controls the available WIFI networks for the participating teams. The arenas for RoboCup@Home offer WIFI networks to be used by the competitors. However, even with these efforts, WIFI connections

are unreliable. The fact that visitors to the competition, and even some participants, broadcast WIFI networks, or use similar frequencies with their devices makes the WIFI networks unreliable, both in bandwidth as in connectivity.

In chapters 5, 6, and 7 further detail about these issues will be explained.

3.2.6 Poor audio quality at RoboCup event

The RoboCup events are hosted in large convention centers. Multiple leagues share the same conference hall, and there are many sources of noise. Members of the public are welcome to walk around designated paths to watch the competition and are, alongside other participants, free to talk and cheer. Robots and maintenance equipment may be heard. Testing and competition happen during the day, so there can be loud unexpected noises; for example, RoboCup@Soccer tends to result in loud cheers when a goal is scored.

These facts make it so that microphones as the ones found in the robots capture a lot of background noise. On standard platforms like Pepper, where one cannot use specialized microphones, it is hard to differentiate noise from an actual user trying to speak to a robot. This provides an extremely challenging environment for Speech To Text or Automated Speech Recognition systems.

CHAPTER 4

RESEARCH METHODS

The methods of this research project included both Action Research and Grounded Theory. While both methods were introduced in chapter 2 of this thesis, chapter 4 will focus on their rationale and applicability. Moreover, Action Research will further be used in chapters 5, 6 and 7 to interpret the three year RoboCup project. Then, Grounded Theory will be used in chapter 8 to interpret the expert survey's data.

4.1 Action Research over the RoboCup@Home SSPL project

Action Research (AR) was chosen as the backbone research methodology for the three years of RoboCup@Home Social Standard Platform League (SSPL) development, as this project would lack replicability. This project involves a high degree of uniqueness, e.g. the same people in the same context will not take part in the same version of the competitions ever again. Action research provided a widely used[1], structured and comprehensive way to validate this type of qualitative research. Importantly, the Action Research cycles fitted smoothly in this project, as depicted in Figure 4.1.

[1] About 2,890,000 publications found with keyword "Action Research" on Google Scholar. Many of those with thousands of citations.

4.1.1 Action Research Cycles Structure

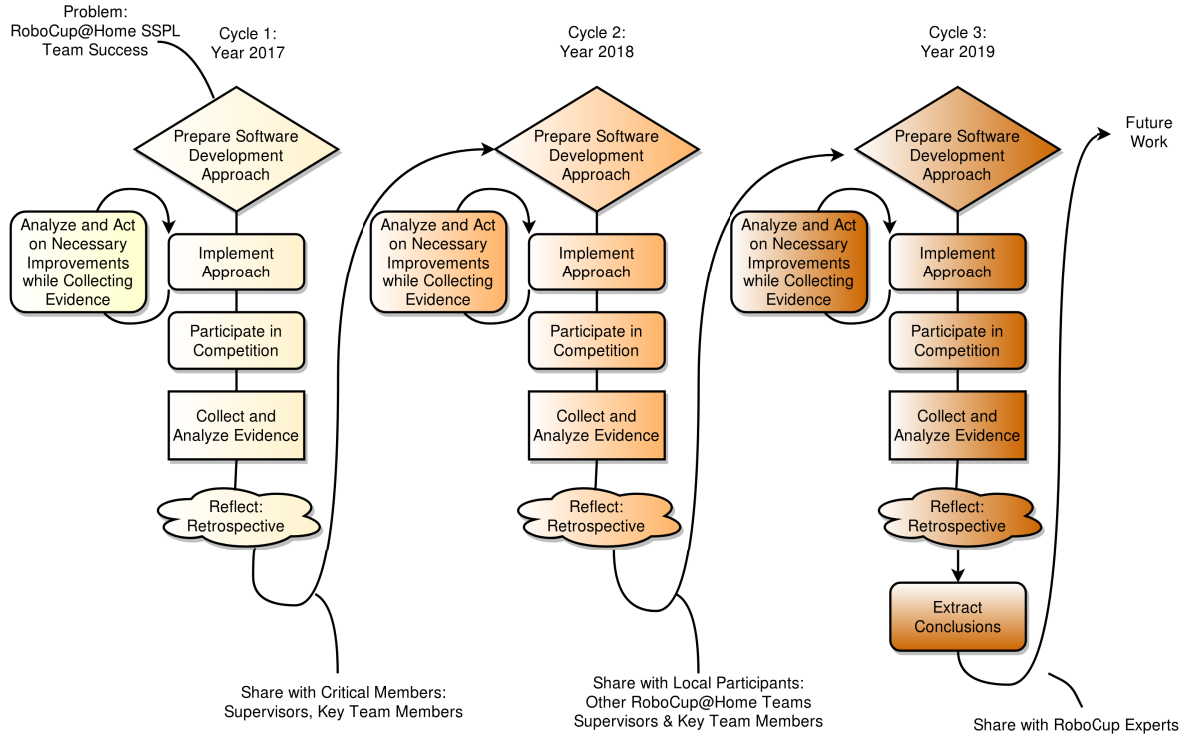


Figure 4.1: Action research cycles diagram for the RoboCup@Home SSPL three years project.

RoboCup@Home SSPL happens once a year. The team participated during three consecutive years. In Figure 4.1 the Action Research cycles for the project are shown. There was an initial research phase each year. Then the researcher and his team iterated over taking action and analyzing the project's progress during the year. After the competition took place, a reflection phase followed. The insights from the year's work were then shared and discussed between experts. Finally, the team members filled a retrospective evaluation document within one month of returning from the competition. This document enabled the collection of their insights, which were further discussed in a meeting.

Another advantage of the nature of Action Research was that given an initial framework of ideas (F), methodology (M) and area of concern (A), these are developed further in the direction that leads to findings. This feature was especially important for us, since it was not clear at the start of the project which factors would play a bigger role in satisfying the process of software development and team management. Additionally, as the team would develop over time, it was therefore predictable that the focus of the

research may be shifted during the Action Research cycles.

4.1.2 Common Topics of the RoboCup AR cycles

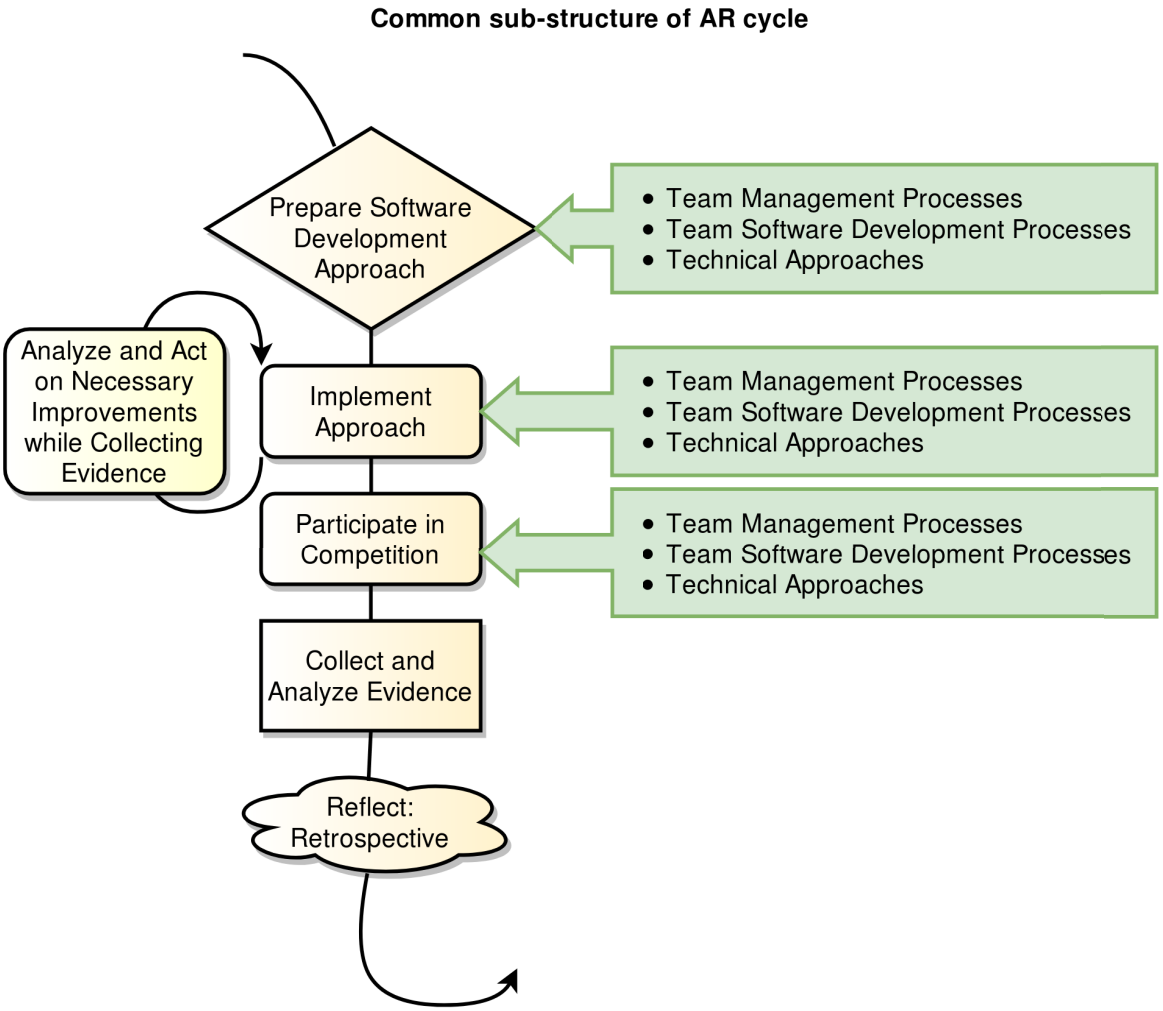


Figure 4.2: One single year of RoboCup Action Research cycle. Repeating topics that include team management processes, team software development processes and technical approaches are detailed.

The same Action Research structure was followed every year, as depicted in Figure 4.1 which made the design of chapter 5, chapter 6 and chapter 7 identical. Moreover, Figure 4.2 illustrates the phases that shared the same substructure: software development plan, software development implementation and competition participation. The substructure contained the following parts or topics:

- **Team Management Processes**: included topics regarding the team's organization. For example, how often the team would meet.
- **Team Software Development Processes**: included topics such as the team's code management and development experience. For example, how the team shared code.
- **Technical Approaches**: comprised topics dealing with implementation details. For example, how the team decided what was more important to be implemented.

This substructure was developed after analyzing the data from the three years of participation. Importantly, these fields were found to be the ones that built up an easy to follow structure to reason about the project. The reader can acquire a full image of the competition by following the three parts. Thus, the substructure represents a valuable tool for any RoboCup@Home participant. Moreover, not only may this work be of use for participants, but also for non-participants, given that the team processes may be insightful for them.

Finally, based on the cycle reflections, a set of guidelines was created at every cycle end. Interestingly, these guidelines provide new and existing teams with recommendations on how to improve the development process for RoboCup competitions.

4.1.3 Scope of Action Research

The context of each Action Research cycle is discussed with each of the three cycles presented in chapters 5, 6, and 7. Together these contexts establish the overall scope of this Action Research project. In particular, the Action Research and its conclusions are limited by the scope of the project: a team of 10-15 participants, competing over a period of three years in the context of a university-driven robotics competition. The generalizability of the Action Research findings to other situations is dependent on the similarity of other situations to this particular project.

As per the roles involved in this research, the members of the team *UTS Unleashed!* were Action Research participants. The author of this thesis and his supervisors acted on both roles of participant and researchers. At different points in time they needed to perform either role. The author was the development lead of the project with the supervisors assisting when necessary. Team members were encouraged to take part in the discussion on how to improve the management of the development and the team

itself. While this created a grey area in between researcher and participant at that particular event, generally team members were participants.

The project had a set timeline of three years. Each year was centered around competing in the RoboCup event and it concluded with the competition itself and a team debrief within a few weeks of the final day of the competition. The nature of software development and team management made it unlikely to find a holy grail methodology that would provide an universal process to manage such a situation and it was unlikely we would find reasons to stop the research process earlier than after the three years of competition.

Conclusions were discussed with experts so as to make the insights found transferable in similar situations. For example, one could find similarities between a startup environment with a small team trying to solve a specific problem via the development of a niche product with the case study presented here. The author believes that specific implementation details of the methodologies found to work for the *UTS Unleashed!* team may not work on other teams, but the process and philosophy of how the team iterated on finding the combination of implementation details is valuable as an inspiration to create a fitting set of practices for other teams.

4.2 Grounded Theory: Experts' Feedback Analysis

Grounded Theory (GT) is used in chapter 8 to develop theories from the data collected. It is also used in chapter 5, chapter 6 and chapter 7 to interpret data from Trello and Git. An example of its usage can be found in Appendix C.

GT offers excellent instrumentation for data interpretation and theory development. Hence their distinct characteristics were broadly applied in this work. Further remarks on these characteristics and how they were applied are listed below:

- **Simultaneous involvement in data collection and analysis phases of research**.

  Collected data from experts are compiled in chapter 8 of this thesis. Importantly, their analyses refined the data collection phase itself as a workshop was held which provided questions for the survey distributed afterwards. Furthermore, the chance of gathering even more data from key participants remained.

- **Creation of analytic codes and categories developed from data, not from preconceived hypotheses**.

Data guided which codes, and with them, which categories were insightful to further work on. For example, during the survey analysis, in many questions with answers in open text format, codes were created to identify ideas repeated in multiple answers, and categories were created from grouping these. The researcher started with a clean slate and an open mind.

- **The development of middle-range theories to explain behavior and processes**.

  As codes and categories arose, middle-range theories were developed to explain the relationship between the participants and their answers. For example, a number of questions in the survey asked to sort by importance, concepts in a specific topic. These were encoded and further categorized, which then made possible the creation of middle-range theories to interpret these responses. Specific ways to manage these theories and their data were developed where needed.

- **Memo-making, that is, writing analytic notes to explicate and fill out categories, the crucial intermediate step between coding data and writing first drafts of papers**.

  Memos were used as a means to path the way into conclusions. For example, looking again into the questions of the survey that allowed open-ended text answers, as the ideas were extracted into codes representing ideas or concepts, these were grouped into categories, and memos were written next to them with possible middle-range theories that aimed to interpret the answers and join them with higher level theories.

- **Theoretical sampling, that is, sampling for theory construction, not for representativeness of a given population, to check and refine the analyst's emerging conceptual categories**.

  Having the memos, the codes and their categories, with the open possibility of interviewing key participants, theoretical sampling was put into practice. Previous data was revisited and key team members were interviewed further about theories that arose.

- **Delay of the literature review.**

  Literature review was done when it became necessary, especially when new concepts to the researcher came up. In addition, the final conclusions were revised based on literature review.

Noteworthy, important artifacts are included in the appendices of this dissertation. Some intermediate artifacts (e.g., codes, categories, memos, intermediate ideas) have not been included, as they do not add value to this dissertation.

4.2.1 Action Research and Grounded Theory Together

Initially, Action Research (AR) was used for the study and development of the three years of the RoboCup@Home SSPL project. Meanwhile, informal chats and interviews with experts took place that not only fed the AR process but also helped on the creation of materials for the Grounded Theory work.

Furthermore the facts, concepts and theories that emerged from our AR study became founding elements of the GT study. As discussed previously in the literature review, the nature of AR enables its embedding into GT. Moreover, both AR and GT imply iterative processes which exploit the continuous development of concepts and theories.

4.3 Statistical Data on Trello and Git

The analysis/study of quantitative data is an unobtrusive alternative to surveys and interviews with team members, that provides an objective insight into activity, productivity and behavior. Therefore, the available data on the Trello platform and the Git repositories in GitLab was analyzed in every AR cycle. Based on these data, hypotheses on the team's working performance during the year were developed/formulated. These hypotheses were then used to discover, confirm or deny theories about the team's conduct. For example, observing which day of the week had more activity in these platforms could be compared to the day of the week where team meetings happened, to create a theory about if these facts were related, i.e. if the team did more work to prepare for a meeting, or after being in a meeting, or a different theory altogether.

Trello and Git are tools of a different nature, with different goals. Thus data in each platform is treated differently, as explained below.

4.3.1 Trello Data

The Trello platform offers boards in which one can create cards and write tasks in them. These cards can be commented and updated, they can have attachments and checklists. These actions can be meaningful to study the team's engagement in organization and progress-sharing behaviors.

Trello offers the data for these boards in JavaScript Object Notation (JSON) format, which in order to be analyzed is converted to Comma Separated Values (CSV) format and loaded into a spreadsheet. The data is treated as anonymous and can be used to create meaningful plots and to extract insights from it. Moreover, the specific contents of the cards or their authors are not reported. The data was analyzed by creating charts for the following scenarios:

- **Trello activity during the cycle**: Trello action count per day during all the development year, providing a sense of how the organizational and tracking of work via Trello was done during the year.
- **Trello actions per month**: Trello action count per month during all the development year, providing a sense of when the most planning and progress tracking was happening in the platform in a more granular shape.
- **Trello actions per weekday**: Trello action count per weekdays. This could show if team members tended to engage in planning and progress-tracking work on any specific days. With this count, we can theorize about the reason for the engagement time.
- **Trello actions per hour**: Trello action count per hour of the day. This allowed observation of at what times of the day this planning and tracking activity happened.
- **Trello board names and their action count**: Trello action count per board name. This could show what needed more focus on that year.
- **Trello board authors and their action count**: Trello action count per team member (anonymized). It may provide a sense of distribution of the planning and progress-tracking work.

4.3.2 Git Data

The team kept its code projects in Git repositories in the GitLab platform. Every Git repository has a history of commits. Every commit is a set of changes with a message identifying what this commit is doing. Therefore, the authors and file changes can be tracked. These commits can be meaningful to represent the team engagement in software development practices.

Obtaining statistics from these repositories either for a single repository or for the full amount of repositories of the team needed custom work. The applied strategy consisted

in downloading all repositories and, for every repository, the commit history was followed, commit by commit, to populate an Structured Query Language (SQL) database on which to run queries to obtain manageable data to be analyzed and plotted.

The concept of "days of commit activity" was considered to provide the most meaningful data, while the raw commit count presented problems with different team members working in different ways. For example, a team member used to commit on every few changes or tests he was making. This added up to a lot of commits, as compared to someone doing a commit only after finishing something, in spite of both doing a similar amount of work in one day. We can presume that if the task was divided in more than one day, even the team member that used to commit only when things are finished, may have done a commit to save the current state of their work and keep working the next day, effectively showing two "days of commit activity" in both cases.

Noteworthy, other approaches were explored, for example, analyzing the amount of lines of code that were changed in the commits. However, this did not provide any meaningful results, as it was complex to find robust data to reason about. Indeed, the case of moving files between folders exemplifies this problem: in the commit data we would find all the lines of the file shown as deleted, and all the lines written again in a different file in the new location. Furthermore, some files such as images were not code and they had to be ignored as well. Another example of these issues is presented by some files being machine generated code. Too many edge cases are found to be able to use changes in lines of code to extract meaningful insight.

Finally this data is analyzed by creating charts for the following setups:

- **Git activity during the cycle**: Git commit count per day during all the development year, providing a sense of how much the GitLab platform was used during the year.

- **Git commits per month**: Git commit count per month during all the development year, providing a sense of when the most coding happened in the platform in a more granular shape.

- **Git commits per weekday**: Git commit count per weekdays. This could show if team members tended to code or submit code on specific days of the week, which could be related to other aspects of the development.

- **Git commits per hour**: Git commit count per hour of the day. This allowed observation of at what times of the day did the coding happen more often.

- **Git commit author count and days with commit activity per repository with more than one author and 1 activity day**: In this chart we showed the repositories where teamwork happened. Repositories with 1 activity day were excluded as it was thought they do not contain enough meaningful information.
- **Git commit author count and days with commit activity per repository with exactly one author and more than one activity day**: In this chart we showed repositories where individual work happened. Repositories with one activity day were also excluded.

4.4 The Author's Role In This Project

During the three years duration of the RoboCup@Home SSPL project, the author of this manuscript, Sammy Pfeiffer, had the role of software development lead.

Sammy lead the software development process for the competition and additionally acted as team leader in some contexts.

Sammy was entrusted to take the architectural decisions over UTS Unleashed!'s systems, usually discussing with the team in order to reach consensus.

He provided the backend software for the robots to run the latest available software. For example, making the Robotics Operating System (ROS) framework available for Pepper. Sammy also took care of the safety and correct functionality of the robots.

In addition, he led the autonomous navigation system, including the mapping, localization and obstacle avoidance enabled navigation.

Sammy mentored and supervised some team members that worked on parts of the system for which he was responsible. Other team members with a lesser level of autonomy were also supervised by him to some extent.

In the years 2018 and 2019, the author was a member of the Technical Committee of the RoboCup@Home organization, participating in the evolution of the competition's rulebook.

Finally, his decisions and actions were influenced by his past participation in robotics competitions, especially in the RoboCup@Home league, and his background working in the company PAL Robotics developing humanoid research robots. In both scenarios the ROS framework was widely used.

CHAPTER 5

ROBOCUP@HOME SSPL: YEAR 2017, 2ND PLACE

2017 was the first year of the existence of RoboCup@Home Social Standard Platform League (SSPL). It was also the first year to participate for the team *UTS Unleashed!*. The rulebook for this year[1] will be briefly explained.

This chapter follows the structure of the diagram in Figure 4.1 which itself follows the structure of action research cycles in Figure 2.2. Every action research cycle is composed of a year of preparation for the competition, participation in it and reflection on the process itself.

In each section the action research cycle setup is explained: the framework of ideas, the methodology and the area of concern for that cycle are first discussed as per the structure explained in chapter 2 and chapter 4. The context for the year follows with its main themes being the rule and competition changes and the team composition of that year.

Following, the software development plan is presented. It is analyzed in three subtopics that will be repeated over every next subsection: team management processes, team software development processes and technical approaches.

Afterwards, the software development implementation presents the same subsections but in the context of the plans put in practice and their iterations.

Then the competition participation is showcased, again with the same subsections, in conjunction with the results, as competition outcomes, for this edition.

[1]The rulebook is 115 pages long. Even though the reader could benefit from reading it once, in this work we will explain briefly specific rules when necessary.

Near the end of the chapter we find the post-competition data collection and retrospectives. Here the available data about the year is collected.

At the end of the chapter the reflection section summarizes and discusses the findings (following again, the same three subsections structure), the new questions that came up from this and the next steps to follow for the next action research cycle. This leads to the end with a proposal of guidelines.

5.1 Action Research Cycle Setup

This is the first cycle of action research, documenting the first year UTS Unleashed! participated in RoboCup@Home SSPL, and the first time its members worked together in such an activity. The research questions here were relatively fuzzy. The area of concern was the development of a team and their codebase for RoboCup@Home SSPL.

The questions behind this action research cycle, which shape the framework of ideas, were:

- What are the processes to set up a new team to compete in RoboCup@Home SSPL?
  - What to extract from the rulebook?
  - What is important about the robot platform?
  - What to look for in the team composition?
  - How to choose the Software Stack?
    - What programming language(s) to use?
    - What frameworks to use?
    - What coding standards to enforce?
  - How to manage the team?
    - How to divide the work?
    - How to track the work?
    - How to gather feedback?
  - How do these questions relate to each other?

The methodology to answer these questions consisted of doing a short research and discussion on them and then form a short-lived plan to re-evaluate as it was being implemented. Further questions arose during the cycle and could have made previous ones irrelevant or requiring a new point of view.

5.2 Context

As presented previously in the scope section of chapter 4, the context of this cycle/year is explained in this section. The context includes the specific rules of the competition for this year and the characteristics of the team.

5.2.1 Rule and Competition Changes

The competition was scheduled for the 25-30 July, in Nagoya, Japan. Providing, from the reception of the robots at the start of May to the competition, approximately 3 months to prepare.

Previously, in chapter 3 the set of common elements of the rulebooks are described. Here the features of the 2017 edition are described. Every action research cycle explains its context in reference to the previous one.

All teams want to perform well. To do so they must follow the rules and score as high as possible while avoiding any kind of penalty. Hence, decisions about development and implementation arise from these rules. Subsequently, it is important to keep these rules in mind. These are explained below.

5.2.1.1 Competition Organization

Extracted from the 2017 rulebook[95] itself, the competition is organized in the following manner:

> "It is organized in two stages each consisting of a number of specific tests. It ends with the Finals.
>
> 1. Stage I: The first days of the competition will be called Stage I . All qualified teams can participate in Stage I . Stage I comprehends a set of Ability Tests, an Integration Test, and an audience demonstration called Following & Guiding. Those Proficiency Tests (Ability Tests, and Integration Test) are performed multiple times (See Section 3.7.4).
>
> 2. Stage II: The best 50% of teams with full integrated capabilities (after Stage I) advance to Stage II . Here, more complex abilities or combinations of abilities are tested. In order to advance to Stage II a team must successfully solve 3 out of Proficiency Tests in Stage I. The Open Challenge is the open demonstration in Stage II.

3. Final demonstration: The best two teams of each league, the ones with the highest score after Stage II, advance to the final round. The final round features only a single open demonstration."

5.2.1.2 Scoring and Penalties

For Stage I *"Each proficiency test is attempted three times. The maximum total score is calculated as the average of the best two attempts for that test"*. All tests, except for the open demonstrations, the tests called Open Challenge and Finals, are rewarded on a partial scoring basis following the guidelines:

1. Tests are split into designated parts.
2. Each part is assigned a certain number of points.
3. A team that successfully passes a designated part of the test receives points for that part.
4. In case of partial success, referees (and TC members) may decide to only award a percentage instead of the full partial score.
5. The total score for a test is the sum of partial scores.
6. Partial scores can be negative (e.g. to penalize failures).

A set of penalties that can be applied to teams exists. Rephrasing from the rulebook, disregard of the following rules can lead to penalties in the form of negative scores, disqualification from a test or even from the entire competition:

1. No touching: During a test, the participants are not allowed to make contact with the robot(s), unless it is in a "natural" way and/or required by the test specification. Robots are allowed to gently touch objects, items and humans. They are not allowed to crash into something.
2. Natural interaction: The only allowed means to interact with the robot(s) are gestures and speech.
3. Natural commands: Only general instructions are allowed. Anything that resembles direct control is prohibited. For example, "Go to the livingroom by turning 180 degrees and moving 2 meters forward" would be forbidden.

4. Remote Control: Remotely controlling the robot(s) is strictly prohibited. This also includes pressing buttons, or influencing sensors on purpose.

5. Penalty for inoperative robots: If a team starts a test, but it does not solve any of the partial tasks (and is obviously not trying to do so), a penalty of 50 points is handed out. The decision is made by the referees and the monitoring TC member.

6. Extra penalty for collision: In case of major, (grossly) negligent collisions the Technical Committee (TC) may disqualify the team for a test (the team receives 0 points), or for the entire competition.

7. Not showing up as referee or jury member: If a team does not provide a referee or jury member (being at the arena on time), the team receives a penalty of 150 points, and will be remembered for qualification decisions in future competitions. Jury members missing a performance to evaluate are excluded from the jury, and the team is disqualified from the test (receives 0 points).

8. Modifying or altering standard platform robots: If any unauthorized modification is found on a Standard Platform League robot, the responsible team will be immediately disqualified for the entire competition while also receiving a penalty of 150 points in the overall score. This behavior will be remembered for qualification decisions in future competitions.

When a test needs an operator[2], the team may request a custom operator (probably a team member of the robot's team) but *"A penalty may be involved when using a custom operator"*. Elaborating further:

> "Automatic Speech Recognition is preferred and any command given to the robot will given by voice first.
>
> 1. Default Operator: The command for the robot is spoken out loud and clear by the human operator. This grants 100% of the available points for understanding the command. The default operator may repeat the command up to three times.
>
> 2. Custom Operator: When the robot renders unable to understand the default operator, the team leader can choose a custom operator can give the command

[2]An operator is a person that will interact with the robot. This operator could be related to the competition, e.g. a referee or a team member of another team, or what is sometimes referred as a naïve user, e.g. a member of the audience.

> exactly as instructed by the referee. Unless stated otherwise, only 75% of the points are granted. A custom operator may repeat the command up to three times.
>
> 3. Alternative Input Method: When the robot renders unable to understand the command given by a custom operator , it is allowed to use any alternative method or interface previously approved by the TC during the Robot Inspection. No points are scored this way."

Furthermore, the rulebook adds about manipulation tasks:

> "When a human assists a robot in a manipulation task, no points are scored for manipulating them. However, the referee may grant proportional points in those cases when the task involves manipulation partially."

The tests have a time limit, usually 5 minutes in Stage I and 10 minutes in Stage II but each test can define its own. If a robot performs anything out of time, it will not be scored, although it may be allowed to continue so the audience, including the other competing teams, enjoy a full performance of a robot in a test.

5.2.2 Team Composition

The team was composed of 14 people. Team members are kept anonymous throughout this document. The roles, time dedication, and background of the team members are to be taken into account. These are presented in Table 5.1.

Two people were dedicated to project management tasks. They were the bridge between the team and the university and provided their expertise in a variety of topics but were not engaged in coding.

One person was assisting with media: photography and videos. This person had no coding experience, however, came to help when the team needed a 'naïve' (no previous experience with the robot) user.

Then, eleven people were in the project with coding roles. Some interviews and coding exercises were set up to roughly assess their coding skills. Of those, four had over five years of professional coding experience (including the author, the Lead Developer), one had around two years of coding experience, two developers had over a year of coding experience, and two developers had completed introductory programming subjects. Finally two team members had basically no coding skills.

#	Team Role	SW Expertise	Background	Time Dedication
1	Project Leader	–	Lab Director, CS & AI	On demand
2	Senior Project Manager	–	IT Project Manager	14h/w
3	Assistant	–	Media Specialist	On demand
4	Lead Developer	Expert	Robotics & CS	32h/w
5	Developer	Expert	Computer Science	Variable (<20h/w)
6	Developer	Expert	Lab Co-Director, CS & AI	Variable (<20h/w)
7	Developer	Expert	Web Developer & IT	20h/w
8	Developer	Advanced	Computer Science	24h/w
9	Developer	Moderate	UX & Multimedia	14h/w
10	Developer	Moderate	CS & AI	15h/w
11	Developer	Beginner	Mathematician	Variable (<10h/w)
12	Developer	Beginner	IT & CS	10h/w
13	Developer	None	Computer Science	14h/w
14	Developer	None	IT & Electronics	30h/w

Table 5.1: Members of the UTS Unleashed! team in 2017.[3]

Most developers were PhD students. Additionally, there were a couple of staff members. Their profiles, interests and backgrounds differed. However, a shared characteristic of most team members was strong inter-personal skills.

On the robotics front, four people had robotics experience: one being an expert with professional experience (the author), another one was acting uniquely as a project manager, and another one had mainly experience in simulation.

On RoboCup experience, four people also had experience: one having multiple years of experience in RoboCup@Home and RoboCup@Rescue, two had several years of experience in RoboCup Standard Platform League, and one had experience in RoboCup 3D Simulation League.

The plan taken from this data was to gather the team availability in hours/week and in which days of the week they could come work to the lab. The time commitment of the

[3]CS stands for Computer Science, IT stands for Information Technology, UX stands for User Experience, AI stands for Artificial Intelligence.

team members was found to be neither equal nor constant, but it was a useful guide.

With the prior information, the division of work was done and training sessions were organized to help team members get up to speed for the development.

5.3 Software Development Plan

This section explains the software development plan for this edition of the competition. It includes the team management processes, the team software development processes and the technical approaches planned.

Particularly in this first year of competition, there was only a short time to prepare for the competition, uncertainty about the team composition, the competition itself, and the robot. Plans were mostly exploratory and subject to change.

5.3.1 Team Management Processes

Choices about the team management related topics were based on the experience of team members in an ad-hoc fashion. They were expected to evolve as required.

5.3.1.1 Team Management tool

For team management, given some previous expertise by the project managers with the platform for another project, Asana was initially used as a tool to organize tasks to be done. The tasks were divided in the following set of topics: project management, development, RoboCup tests, and training. An initial set of tasks was created by the lead developer and the viability of the platform was evaluated as we used it.

5.3.1.2 Team Meetings

Weekly team meetings were setup where each team member would showcase their progress, problems would be raised and general training sessions would be performed. Everyone was encouraged to stay working in the lab the day of the meetings to increase team work and aid in becoming familiar with each other.

The day and time for the meetings was chosen to be Tuesdays at 11AM as it was found, after a short survey, to be the slot where everyone could attend.

5.3.1.3 Task Assignment

Topics to work on were proposed, and from them, tasks to work on were created. Team members were encouraged to pick the tasks that they were most interested in, either because of previous experience or because they wanted to learn more about them. Most team members did not express interest in doing any specific kind of task. In those cases tasks were proposed to members. The lead developer perceived this as a manifestation of a lack of proactivity.

5.3.2 Team Software Development Processes

This section presents the practices and tools regarding how the team was to develop software.

5.3.2.1 Coding Standards

An optional tutorial on Python, its coding style, and naming standards was held shortly after the development started to introduce the team to the language. The lead developer believed that due to the short time until the competition, it was not advisable to put much pressure on obeying coding style and coding standards. It was better to have some code that accomplished something than non-functional beautiful code.

5.3.2.2 Coding Tools

Some coding practices were adopted as they became necessary, such as a centralized code repository. GitLab was chosen for this job as explained in chapter 3. Using continuous integration and continuous deployment (CI/CD) was also discussed as nice to have but not enforced.

As for code editors/Integrated Development Environment (IDE) Sublime Text 3 with Python support[4] was proposed.

[4]Automated code formatting, syntax highlighting, and code completion were features required by any editor.

5.3.2.3 Social Coding Practices

Expert developers would supervise or practice *pair programming*[5] from time to time with less expert team members to share knowledge, increase effectiveness, and improve code quality.

5.3.3 Technical Approaches

The technical approaches, or specific decisions made during the competition itself, are discussed in this section. These are related to the RoboCup@Home SSPL itself and describe the competition requirements, the robot's capabilities, and the software stack.

5.3.3.1 Competition Requirements

The development lead read the rulebook carefully for this first year (2017) as software development started, and invited the team to do the same as all team members needed to be familiar with it. From this initial reading, a set of tests that seemed the most likely to be able to be implemented in time, and that would provide the highest chance of scoring, was chosen. These were chosen by the lead developer in consultation with the team. The set was not final, but was tightly related to the skills that the robot could perform with minimal development.

Only tests from Stage I were chosen initially, because if insufficient points were scored in Stage I, any effort invested in developing Stage II tests would be wasted. Additionally, it was believed that if the robot skills developed for the tests in Stage I were general and robust enough, implementing Stage II tests based on those Stage I skills should be possible in a shorter timeframe. The chosen tests in Stage I were:

- **Robot Inspection**: Not exactly a test, but a pre-requisite to participate in the competition. The robot has to navigate safely some distance, stop when encountering a referee blocking its path, then present itself to the referee. Finally, the robot's Emergency Stop buttoni, which effectively shuts down the electrical current on the Pepper robot, is pressed to confirm it is active and working. A team member also explains any meaningful detail about the robot's tablet interfaces or External Computing devices used by the team.

[5]Different interpretations of what does pair programming consist of exist. The lead developer understood it as two programmers working together in one computer. One of them writes the code and explains it, meanwhile the other reviews every line and asks questions.

- **Cocktail Party**: The robot has to learn and recognize previously unknown people, and fetch drink orders. This test focuses on human detection and recognition, safe navigation and human-robot interaction with unknown people. There is a party room with guests, the robot must navigate there and identify guests wishing to order a drink. It will take the guests' orders and go to the bar and ask the barman for those drinks, then return to the guests with those drinks.

- **Speech and Person Recognition**: The robot has to identify unknown people and answer questions about them and the environment. This test focuses on human detection, sound localization, speech recognition, and robot interaction with unknown people. After stating that it wants to play a riddle game, the robot turns around and waits for 10 seconds while a crowd is assembled behind it's back. When the time elapses, the robot must turn around and find the crowd. After turning around, the robot must state the size of the crowd (including male and female count) and request for an operator (e.g. *"Who wants to play riddles with me?"*). The crowd will move and surround the robot, with the operator standing directly in front of the robot. The operator will ask 5 questions. The robot must answer the questions without asking for them to be repeated. Afterwards the Blind Man's Bluff / Circling Crowd game will start. The crowd will reposition, making a circle around the robot. A random person from the crowd surrounding the robot will ask a question. The robot may turn towards the person who asked the question and answer the question. Or directly answer the question without turning. Or turn towards the person and ask them to repeat the question. This process is repeated with 5 (possibly) different people. Each option is scored differently.

- **Help me Carry**: The robot's owner went shopping for groceries and needs help carrying the groceries from the car into the home. The robot starts in a room in the arena. An operator steps in front of the robot and tells it to follow them to the car. The robot has to memorize the operator, then follow them. The operator will walk naturally to the car (leaving the house). Upon arrival at the car, the operator will indicate to the robot that they have arrived. At this point the robot is asked to look for help to carry the groceries (e.g. *"Look for Louise in the Kitchen and ask her to help us"*.). On the way back to the house a person will cross the path of the robot and another person will step in front of the robot, stop and ask for the time. The robot then must find the person in the room, memorize this new operator and guide this new operator to the car. This new operator will get distracted on the way

and the robot must re-gain the operator's attention.

- **General Purpose Service Robot (GPSR)**: This test evaluates the abilities of the robot that are required throughout the set of tests in Stage I of this and previous years' rulebooks. In this test the robot has to solve multiple tasks upon request. That is, the test is not incorporated into a (predefined) story and there is neither a predefined order of tasks nor a predefined set of actions. The actions that are to be carried out by the robot are chosen randomly by the referees from a larger set of actions. These actions are organized in three categories with different complexity. For example, valid commands are *"Tell the time to Ana in the bedroom"*, *"Tell me how many beverages on the shelf are red"* or *"Follow John (John's location is not specified)"*.

- **Poster Session**: Each team presents a poster with their research. This is not exactly a test, but it is mandatory and it is scored.

Furthermore, from analyzing this set of tests, a set of skills that the robot needed to be able to perform was developed. These are listed in the next section.

Finally, the team delayed allocating development tasks to team members until sufficient knowledge of the robot's in-built skills and the development tools to be used, had been acquired.

5.3.3.2 Software Stack

The lead developer, taking into account the profile of the team, the goals to be accomplished, the previous experience of the team, advice from other experts, and the fact that there would not be enough time to change, decided the operating system and the programming language to use as well as other basic software. The chosen software stack is summarized in Table 5.2. Further description of the elements was explained in chapter 2 in chapter 3.

There were a large amount of unknowns to explore. Any time a new robot is used, both the hardware and the software necessary to access that hardware, must be analyzed to understand the capabilities of the platform.

A set of tasks tightly tied to the competition requirements were created:

- Evaluate native audio processing capabilities: Automatic Speech Recognition (ASR), Text To Speech (TTS), Sound Source Localization (SSL) and Natural Language Processing (NLP).

Category	Name	Comments
Operating System	Ubuntu 16.04	Latest stable Ubuntu
Programming Language	Python	Easy to learn, thriving ecosystem
Middleware/Framework	ROS	Widely used in robotics
Pepper Framework	NaoQi	Necessary to use the platform

Table 5.2: Software stack chosen to work for the year 2017. It became the base for the next years of development too as discussed in chapter 3.

- Evaluate Computer Vision capabilities: Face Detection, Face Recognition, Gender Detection, People Detection, Tracking, Waving Recognition, Age Detection, Object Recognition.
- Evaluate Human Robot Interaction / Human Computer Interaction: Face Enrolment, Tablet Interface, Gesture to Speech coordination, Dialog Design.
- Evaluate Navigation and Planning: Mapping, Obstacle Avoidance Navigation, Cross the Door Navigation[6], Following a person Navigation[7].
- Evaluate the Choregraphe suite.
- Evaluate the Robotics Operating System (ROS) support for Pepper.

These tasks were distributed to the team members by the development lead with help of the project manager by discussing with them the scope and feasibility of these tasks. These tasks required weekly progress reports in the team meetings.

5.4 Software Development Implementation

This section discusses how the software development plan went in practice.

5.4.1 Team Management Processes

The team had a short preparation time of around three months, hence, many changes happened at the start of the development as a consequence of discovering how the team

[6]Related to the rulebook, some tests require the robot to start the test by crossing a door.
[7]Related to the rulebook, some tests require the robot to follow a person somewhere.

worked best. From then on, the team stuck with those practices, as no further need to change was felt necessary.

5.4.1.1 Team Management Tool

As previously explained in chapter 3 the team switched to Trello after roughly a month of Asana usage. It was used in a similar fashion as Asana but perceived as more user-friendly by some team members and the lead developer.

A set of boards were opened with the topics being: Pepper Robots, Robot Skill Development, RoboCup Tests, Team and Research. This illustrates the intention to work out on the one hand how the robots worked, how to code skills for them, how to integrate these into actual RoboCup tests, and on the other hand topics about the team itself and research that is related to all these elements.

In practice, only the boards RoboCup Tests, Robot Skill Development and Team were used. This will be analyzed further in a later section.

A whiteboard with post-its also was used just before going to the competition when all the team was working together in the lab every day.

5.4.1.2 Team Meetings

Team meetings stayed weekly on Tuesdays. Sub-teams appeared for different tests, they made their own weekly schedule.

The agenda for every meeting was filled by every sub-team leader before the meeting, everyone was expected to read it before attending the meeting. This allowed the meetings to be kept short and focused, as they were meant to be used to raise problems and ask for help, instead of just explaining what everyone had done.

By the last weeks before the competition, the team was mostly working together in the lab everyday. So meetings happened naturally as needed.

Team meetings were held with one person leading the conversation. Initially, every person was asked about the progress of their tasks and there was open discussion to learn about everyone and their work. Later on, the conversation was focused on specific RoboCup test development, and as the team was divided into small sub-teams working on each test, a spokesperson outlined progress, findings and issues. When people were developing skills or features aside of a test, they would speak separately. Quick tutorials with examples were given when a piece of code to be used by everyone was ready (or at least ready to be tested by others). Some meetings were reported to be too long.

These meetings evolved into RoboCup@Home fake tests sessions. Initially people that had a prototype of a test would just run it, then more tests joined. Additionally, more structure, as if it was the competition, was added. For example, a schedule with time for setup and an exact time for each test to start, with little time to prepare, was added. Finally, the concept of Operational Readiness Tests (ORTs) emerged. In these, the team would practice, as close as possible, the scenario of arriving at RoboCup, doing the setup, and running all the tests in a tight schedule. A team member acted as a referee by scoring and following the rules strictly.

In order to make these ORTs as realistic as possible a set of low height walls and doors were manufactured to create a realistic arena in the lab.

One ORT happened out of the lab in a different venue which included transporting the robots there, setting them up, setting up a new arena that was never seen before, dealing with networking issues, dealing with only having 2 robots for all the team to prepare and test. In general it simulated the stressful conditions of the competition. This showed where the team was overconfident and what needed to be addressed clearly and in practical fashion.

This first off-site ORT is an important moment of the project. It became the best tool to identify the real state of the development of the project. It is further used in the following years of development.

5.4.1.3 Task Assignment

During the development the lead developer noticed that some members were struggling to get up to speed on coding their tasks, even after training sessions and one-on-one sessions. Tasks involving less coding were assigned, but as the deadline to the competition approached, it became increasingly difficult to keep an appropriate ratio between management and coding for some team members. In the final weeks leading up to the competition, these people were not assigned tasks.

5.4.2 Team Software Development Processes

While it turned out that team members found and used their own ways to work, there were partial adoptions to the proposed processes and tools.

5.4.2.1 Coding Standards

Most team members did not follow Python coding standards. This became an issue when team members wanted to share code, as they had a hard time understanding each others code. Differentiating between functions and classes was one major issue.

The members with more expertise tried to fix these issues and give examples when possible, but everyone was busy "just making things work".

5.4.2.2 Coding Tools

Team members embraced command-line git and *GitLab* to save their work. Their usage of git was mainly to push their latest changes when they had finished a feature. Commit comments were of low quality. Multiple occurrences of "previous version" files committed, instead of using git features to be able to go back to previous versions.

As code editors most team members adhered to the usage of *Sublime Text* with Python plugins, however, some could not make autocompletion work. Similar issues arose during the year showing that installation issues were common.

Furthermore, as deployment strategy in the robots, some people used *rsync*[8], others just copied the files by hand via *scp*[9] and others used some File Transfer Protocol (FTP) graphical client. All the methods had their positive and negative points. Everyone deployed in their own folder with their experiments. Only expert team members deployed code that was common to all team members, like system dependencies.

The robot itself behaved in unreliable ways. Errors appeared sporadically without seemingly reproducible reasons. Internal modules of the robot would stop working without reporting an error. Some modules even would start by what team members jokingly referred to as "magic", and an "aliveness" mode of the robot, which would fight for control of the robot body with any code a team member was running.

State machines within the library State MACHine library (SMACH)[10] were used to orchestrate RoboCup tests. Some boiler-plate code was provided and was mostly well embraced by the teams. Other team members used plain Python scripts, which also

[8]Rsync (remote synchronization) is a fast and versatile command line utility that synchronizes files and folders between two locations over a remote shell, or from/to a remote Rsync daemon. It provides fast incremental file transfer by transferring only the differences between the source and the destination. [96]

[9]SCP (secure copy) is a command-line utility that allows you to securely copy files and directories between two locations. [97]

[10]SMACH is a Python library to create hierarchical state machine commonly used for rapidly implementing complex robot behaviours.

worked, but were harder to integrate into other people's work. Mixed approaches were also used.

5.4.2.3 Social Coding Practices

Team members shared code by using git submodules[11]. In their RoboCup tests, team members would include a submodule folder pointing to the version of a library that was developed in a different repository with the exact version that they tested that had worked for them. This practice was used as a workaround for other team members breaking APIs or functionality when they needed to update these common libraries. These issues and workarounds showcase the issues that the team faced when they needed to integrate code between them.

Working in branches[12] and creating Pull Requests[13] was proposed, but some team members said it was too complicated, so it was not pursued further.

README files were promoted as a way to document and give examples of usage in the repositories.

Team members worked in pairs and practiced pair programming occasionally. Most of the time two people were needed to test the robot. This was because of safety concerns (both for the robot and the users to some extent: the robot could knock into things or fall over) and because of the capabilities needed to be tested, for example, following a person.

Testing was mostly done on the robots. No working simulation was available. Most team members were not trained to use tools that saved sensory data from the robot and to code using this test data. If they had been trained, it would have avoided wasting too much time dealing with issues with the robot unrelated to the problem they were trying to solve.

Soon the importance of testing full RoboCup tests arose. When only testing individual abilities of the robot, it used to perform well enough. But when mixing all the skills together in a RoboCup test, unexpected problems arose. These problems were different in nature. Not only coding mistakes or bugs, but also expectations and assumptions that were wrong once tested. Also, lots of mistakes were made because of team members doing a lot of manual steps in order to be able to run a RoboCup test. For example, launching

[11] Submodules allow you to keep a Git repository as a subdirectory of another Git repository. This lets you clone another repository into your project and keep your commits separate. [98]

[12] A branch is a pointer to a snapshot of your changes, multiple branches can coexist and be merged together when desired.

[13] A Pull Request is a mechanism to notify a team or collaborator that some changes to a codebase are complete and they are ready to be merged definitely into the codebase.

the services that provide navigation or speech recognition capabilities were sometimes forgotten to be run at the start of the test.

The focus shifted into trying full RoboCup tests from Stage I, and fix problems as they came up. Subteams working on a test tried to have all the parts of a RoboCup test ready so that it was possible to perform all of it. But, as the different parts that were developed were not robust enough, it was decided to put all the team's efforts into having the elements that brought it to the first scoring part of the test completed first. Only when that worked most of the time, would they move forward.

Some testing sessions in the lab were done with a person acting as the referee and being strict with the rules. While this provided good feedback, the team left many aspects, that in the competition were variable, as static. As mentioned previously, these testing sessions evolved into something similar to the RoboCup competition, getting the name of Operational Readiness Tests (ORTs). The team would do a complete new setup (as if the team was in the setup day of the competition), with a new home environment to map, new names for people, new objects, a new WIFI network to work with, a person acting as a referee, having slots of time for setup, and strict slots of time for testing. These sessions were found to provide the most realistic feedback as to how the team was going to perform in the competition, and they helped prioritize critical elements to be developed and to make robust.

5.4.3 Technical Approaches

The capabilities of the available tools were discovered, providing mixed results.

5.4.3.1 Software Stack

The tasks distributed to the team members to evaluate the capabilities of the robot had made good progress. The findings were discussed in team meetings, and actions were taken to take advantage of the useful components and further research was identified for action in others.

It was discovered that the Pepper robot lacked some default capabilities that were showstopping. The main missing capability was autonomous navigation, composed of mapping and navigation. In order to fill this gap, and benefit from all the extra libraries and tools that this would provide, Robotics Operating System (ROS) was made available for the platform. Because the default ROS support for the platform was so poor, a full

compilation of ROS and its dependencies was needed[14]. A project called *pepper_ros_setup* was created with a set of bash scripts which manually downloaded and compiled all dependencies and packages. The process was complex to manage, with long compilation times (due to large packages such as GCC[15] itself, and the need to compile it in a Virtual Machine) and a long list of dependencies making it regrettable not having a package manager that would deal with most of this work.

The Python Application Programming Interface (API) of the robot was found to lack in-place documentation, so a package called *QiMate* was created to fill this role and also provide a layer of customization between our code and actual API calls to the robot. Additionally, it aimed to make the usage of the API easier when possible. The framework itself provided some tools that were found useful, such as a global shared memory where to read and write. Text to speech and speech recognition were found to perform adequately.

The different sensors of the robot performed worse than expected in the short tests that were performed. This made it necessary to keep trying new techniques to achieve acceptable performance of the robot in different test elements, like people tracking or robot navigation.

The graphical programming suite Chroregraphe was found to not fulfill the team's needs. It was found to perform what were regarded as undocumented actions that left the robot in a state where tests could not be repeated until the robot was rebooted. Also, one single instance of Chroregraphe connected to the robot blocked the WIFI network bandwidth. Thus, and as a solution, a Python-centric approach was taken.

Further tests with the platform confirmed that the Central Processing Unit (CPU) of the robot had low computing power, implying that extra work was needed to be done to make different parts of the system work together without starving the CPU.

Taking advantage of having team members with an interest in Human Robot Interaction (HRI) and user experience design, a sub-team was established to develop a tablet interface to enable interaction with users, test robot features, and debug and launch code without the need of an external computer.

Moving the robot, by pushing it, was problematic as the robot had safety checks in place for sudden movements and it would stop, often. Also, pushing the robot was found to make the base slide on the floor and it would lose track of its odometry, which made

[14]Usually installing pre-packaged and pre-compiled binaries is done in operating systems like Debian or Ubuntu.

[15]GCC is the GNU Compiler Collection, a well known, and large, compiler system that supports various languages like C, C++, Fortran, ADA or Go.

anything related to mapping and navigation perform worse. Given that, an innovative and original way to drive the robot called 'MotorBike' mode was developed. This mode allowed it to be driven and steered by positioning its arms to resemble motorbike handlebars and twisting its hands like a motorcycle throttle.

To collaborate on the development of the RoboCup tests themselves, the team decided together to use a single repository containing the tests implementation, with git submodules storing their dependencies. This was because team members manifested that dealing with common dependencies that broke often as they were in constant development was problematic. They initially dealt with it by copy-pasting the code that they needed which brought many more issues. Git submodules allowed developers to add a subfolder with a specific version of a library to be used just by one specific RoboCup test. When needed, one could just try to update the submodule version and check if everything was working. If it did, the submodule would be updated to point to that new version. This was an improvement, but it came with its own problems as it was frequently forgotten to update the submodules. Other issues arose from this approach, as some tools needed to be able to perform many different behaviors depending on the requirements of each developer. When that became an issue, tools were duplicated and the team ended up with many versions of the same tool. There was a clear lack of structure, but this was to be expected as there were just too many unknowns at the start of the project. Working with branches was found hard, it involved learning more about git than the team was comfortable with.

5.5 Competition Participation

The development lead felt confident about being able to perform adequately in the tests the team had prepared for, but expected other teams to also be well prepared.

5.5.1 Team Management Processes

The focus of the processes during the competition was interpersonal interaction and feedback.

5.5.1.1 Team Management Tool

A whiteboard with post-its was used over Trello, avoiding any network requirement for task management. The whiteboard had three sections: To Do, Doing, and Done. Once someone took a task, this person would add their name to the post-it. The whiteboard provided easy and quick task management.

Every test was recorded in video by at least one team member, if not more, to be able to review how it went. It was also a useful tool to show the referees any detail that they may have missed which may affect the scoring.

5.5.1.2 Team Meetings

A morning meeting to plan what to do during the day was held. A checkup before lunch to see how things were going followed. Another meeting before leaving the venue to plan the next day finished the day.

Short meetings to be on the same page when meaningful events happened were also held. For example, when some capability of the robot was seen as working well or when it was to be assumed a capability would not work, or when the team competed in one test to share the results and experience.

A project manager was dedicated to support only. This person took care of duties like having drinks and snacks for the team members. Organizing lunch and dinner. Making sure people had enough rest. In general, the project manager dealt with situations that were not about the competition in itself.

All team members made an effort to make best use of the available work time at the venue. In practice this was from 9AM to 11PM every day, which made the team members become increasingly tired as the days advanced.

5.5.1.3 Task Assignment

Task assignment happened by taking post-it notes from the whiteboard from the To Do column to Doing while adding the name of who was working on it. Once done it was moved to Done and the person would pick another task to do.

Tasks that would block the work of others would be prioritized. For example, the robot needed to have a map of the arena for the tests that made use of it, so this was the first thing to do once the arena was ready to be used.

There were time slots available for each team to have exclusive access to the arena. These were planned to be used to their full extent. These time slots presented the best opportunity to build maps with a clean arena so no obstacles would be seen by the sensors of the robot, for example, the legs of other teams members.

The project manager took care of checking that the robots were charging when they weren't being moved. Having a robot run out of battery when the robot was to compete was to be avoided.

Team members did their best to use all their available time and energy communicating to each other when they were ready for a new task.

5.5.2 Team Software Development Processes

The major philosophy here was to avoid developing new code, but instead to configure and adapt existing code. Furthermore, "hack" whatever was needed.

5.5.2.1 Coding Standards

Coding standards were pretty much ignored. Committing often was favoured over any other concern.

As the team aimed to avoid developing new code, fixing bugs and workarounds was the most common kind of coding performed during the competition.

The team only had enough code for Stage I, so in reality, a lot of fast prototyping or "hacking" happened to achieve components that performed adequately.

5.5.2.2 Coding Tools

The team prepared scripts to setup the rented robots automatically.

A local GitLab server was also ready to be used in the case of the internet not working or being too slow.

To deploy each teammate's work, each team member would create a folder in the robot and work in it to avoid breaking other team member's setups.

Everyone had their own laptop that was preconfigured before coming to the competition.

5.5.2.3 Social Coding Practices

The team aimed to maximize the usage of the robots and the usage of the arena while not letting people be idle without reason. However, at the same time, allowing people to have a break by organizing the schedule was important. This was achieved by communicating between the team members as events and milestones happened, such as a system being ready for testing or a RoboCup test being finished. Whenever possible, testing with the robot was done in pairs so one person could be actively working on their laptop and the other one would help to operate the robot. When a team member needed to focus deeply in a task they would warn the team about it and they would then not be interrupted.

5.5.3 Technical Approaches

The reality of the competition pushed all the prepared software to its limits, and encouraged the development of smart tricks to get components up and running quickly.

5.5.3.1 Software Stack

The team arrived at the competition with the tests for Stage I prepared. Only one test was partially prepared for Stage II, Enhanced Extended General Purpose Service Robot (GPSR) (EEGPSR), as it was an extension of GPSR. Finally, nothing was prepared for the Finals.

The competition network went down a few times, but most of the time it performed correctly. The lead developer expected it to not operate satisfactorily from the experience in other years participating in the competition, so efforts to support external computing were not done during the development time. The short time to prepare for the competition also implied that investing in this front would take time off from other necessary tasks.

The software systems were believed to be ready for the competition thanks to the ORTs held. But they didn't perform accordingly. Summarized comments from the team members are found in the retrospective section.

Localization didn't work at all in the venue, which made the rest of the navigation system unstable. The navigation system performed extremely poorly. During the ORTs the

robot could successfully navigate around the map. But in the competition, workarounds were coded for it to at least reach the critical parts of the tests, in order to score.

Moreover, speech recognition performed poorly. The noise levels were high, even higher than expected. There was a discussion between some team members about how it was felt that participating in the first test of the day, being the first team to participate, helped to have a lower level of noise. Certainly, in comparison to tests in the evening where more public would visit the venue and other competitions were happening.

Furthermore, perception performed worse than expected. The illumination varied during the day and night given a set of big windows were present. Also the lighting was different to the conditions found in the lab. This was a known problem, but only limited time could be invested in it.

Other aspects of the system were insufficient. There was a lack of monitoring capabilities for the onboard computer resources. The team didn't have a system capable of storing sensor data from real tests in order to debug what happened after a run, the WIFI and internet connectivity were not taken advantage of, and there was a lack of test scripts to test separate parts of the system with ease.

However, some aspects did perform satisfactorily. The HRI experience and the robot tablet interface was useful and regarded positively, not only by our team members by also by other teams. The team's ability to code, test and integrate features quickly provided good results too, as the in-situ development for the Open Challenge and the Finals tests showcased. This capability was based on the libraries, tools and the teamwork built during development phase.

5.5.4 Results: Competition Outcomes

The results for UTS Unleashed! in Stage I can be seen in the scoring sheet in Figure 5.1. The team was second at that point. The team only scored 0 in the *Help Me Carry* test, but its difficulty can be perceived by only two other teams scoring on it, and they scored low (5 and 10 points out of a maximum of 200).

SSPL Stage 1

	Robot Inspection	Poster	Speech & Person	Cocktail Party	Help Me Carry	GPSR	Stage 1	Rank	
AUPAIR		45.00	117.5	30	10	42.5	245.00	1	Stage 2
UTS Unleashed!		33.33	85.5	27.5	0	17.5	163.83	2	Stage 2
SPQReL		41.67	32.5	10	5	7.5	96.67	3	Stage 2
KameRider		31.67	60	0	0	0	91.67	4	Stage 2
UChile Peppers		31.67	50	0	0	0	81.67	5	
UvA@Home		20.00	47.5	0	0	0	67.50	6	
ToBI@Pepper		41.25	17.5	7.5	0	0	66.25	7	
Maximum		50	200		200	250	700		

Figure 5.1: 2017 scoring sheet of RoboCup@Home SSPL Stage I. The team was second at that point.

UTS Unleashed! went into Stage II unexpectedly. The team decided to prepare the Open Challenge and the EEGPSR test, as it was an extension of the GPSR test that was prepared for Stage I. The team didn't have the appropriate base skills to be able to implement the other tests in time. For the Open Challenge there were projects that were not used during the competition for other tests but were developed during the year as tools. These were unified in a presentation and made into a demo for the Open Challenge.

SSPL Stage 2

	Stage 1	Open Challenge	Tour guide	Restaurant	EE-GPSR	Stage 2	Rank
AUPAIR	245.00	178.47	95	40	70	628.47	1
UTS Unleashed!	163.83	121.53	0	0	0	285.36	2
SPQReL	96.67	130.56	0	10	20	257.22	3
KameRider	91.67	136.81	0	15	0	243.47	4
UChile Peppers	81.67	0.00				81.67	5
UvA@Home	67.50	0.00				67.50	6
ToBI@Pepper	66.25	0.00				66.25	7
Maximum	700	250		285	250	1485	

Figure 5.2: 2017 scoring sheet of RoboCup@Home SSPL Stage II. The team was second at that point.

From this point the team qualified for the Finals. This was also unexpected. The team quickly hacked up a demo with some other projects that were also unused during the competition.

There is no scoring available for the Finals (it was not released), but the final classification, and awards can be found in Figure 5.3.

Final Results	
Social Standard Platform 2nd Place	UTS Unleashed!
Social Standard Platform 1st Place	AUPAIR
Open Platform 3rd Place	ToBI
Open Platform 2nd Place	TechUnited Eindhoven
Open Platform 1st Place	homer@UniKoblenz
Domestic Standard Platform 3rd Place	UT Austin Villa
Domestic Standard Platform 2nd Place	eR@sers
Domestic Standard Platform 1st Place	Hibikino-Musashi@Home SPL

Awards	
Open Source Software Award	eR@sers
Best Scientific Poster	Alle@Home
Best in Person Recognition	N.A.
Best in Speech Understanding	Pumas
Best in Manipulation	WrightEagle (in Open Challenge)
Best in Navigation	O.I.T. Trial (Help me carry)
Best Human-Robot Interface Award	UTS Unleashed!
Social Standard Platform 3rd Place	SPQReL (sparkle)

Figure 5.3: 2017 final classification scoring sheet of RoboCup@Home SSPL Stage II. UTS Unleashed! was second and won the Human-Robot Interface award.

UTS Unleashed! earned 2nd place after team AuPair. The team also won the best Human-Robot Interface award as a result of the tablet interface and well-curated dialogues.

The team gained considerable experience about the competition, Pepper's capabilities and themselves as a team. From the winners, team AuPair, the possibilities of taking advantage of the latest machine learning advances in combination with external computing via WIFI was learnt.

5.6 Post-Competition Data Collection and Retrospectives

Statistical data was collected from the Trello boards and from the Git repositories. A retrospective document was also collected from the feedback of the team members, these are discussed in this section.

5.6.1 Statistical Data

Every year the available data from Trello and Git was analyzed as explained in chapter 4.

5.6.1.1 Trello Cards Data

Overall, the Trello data for this year showcases continuous work during the development months, May to July. Activity in Trello relates to the team's meeting day of the week. As the competition approaches, Trello is used less with most activity shown to happen in developing RoboCup Tests.

Steady work can be observed week by week and month by month in Figure 5.4.

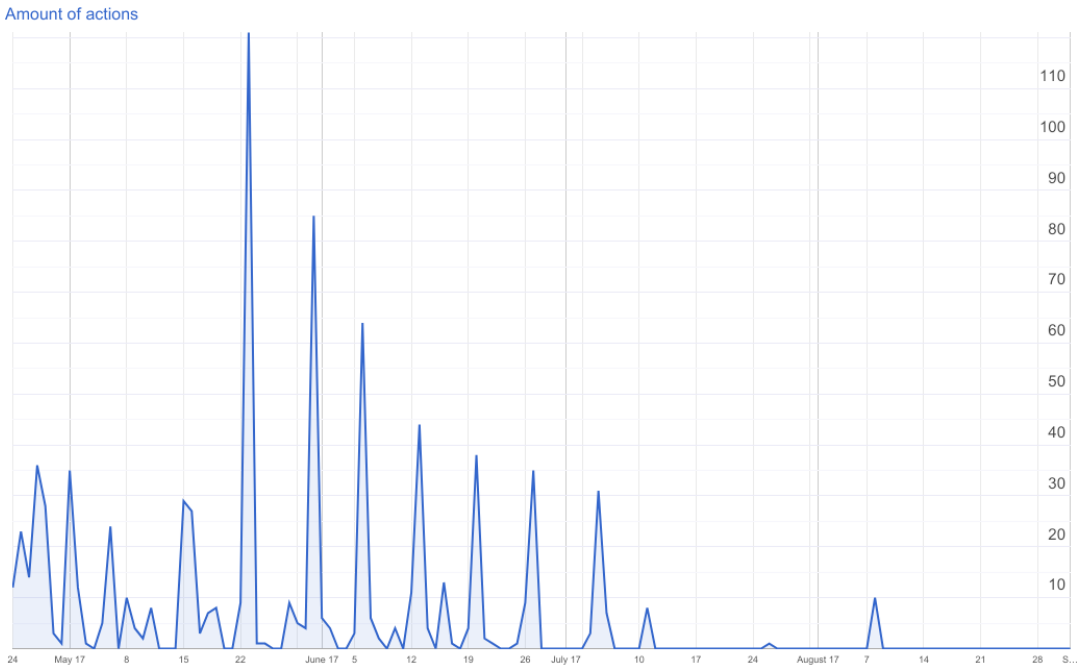


Figure 5.4: Activity (Trello actions) on the year 2017.

Analyzing what kind of actions were triggered the most in Trello in Figure 5.5, the most common usage was to update cards with content and commenting on them. Creating them, adding team members to cards, and adding attachments were the following most used actions. Additionally, cards have a long life presenting updates and comments.

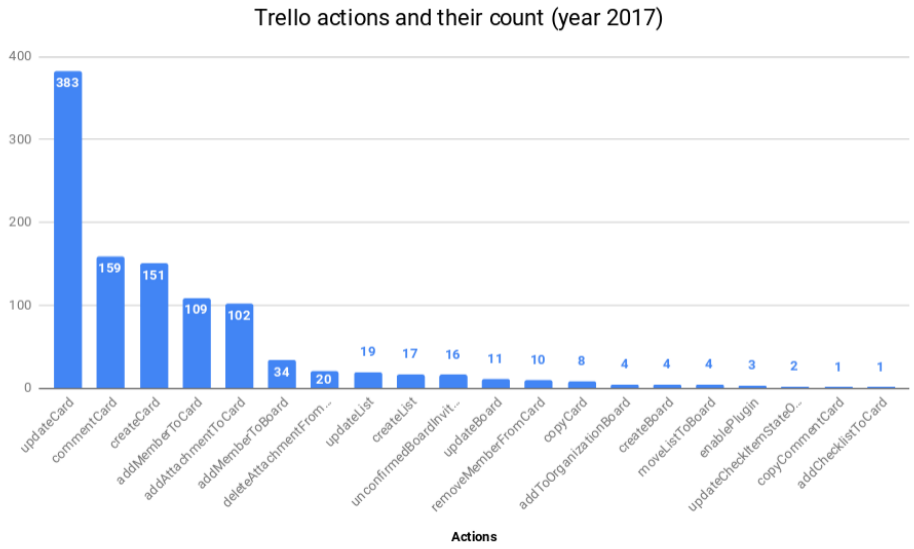


Figure 5.5: Distribution of Trello actions types on the year 2017.

In Figure 5.6 we can observe that the closer the competition was, the less Trello activity happens. This matches with what could be observed during development, as by the end, team members would just work together and a physical post-its board would be used by the last month to quickly work on tasks. Also in May, the first month, where more planning and discoveries were made, the largest amount of activity was observed.

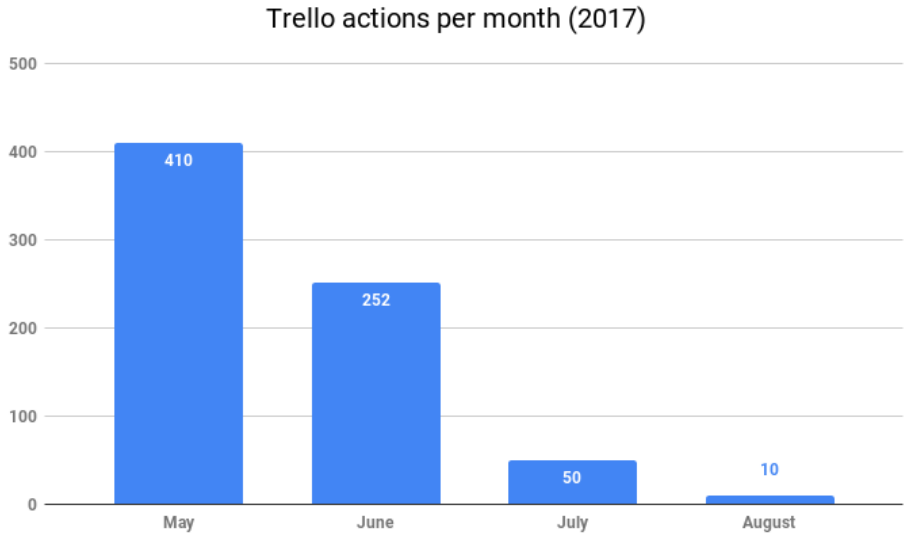


Figure 5.6: Distribution of Trello actions by months on the year 2017.

In Figure 5.7 we can observe how Tuesday's were the days with most Trello activity. This matches with Tuesday's being the day where the team would have meetings, thereby Trello was updated accordingly.

Figure 5.7: Distribution of Trello actions by weekdays on the year 2017.

Moreover, by checking the hours with most activity in Figure 5.8, activity tended to start at 10AM and ending by 6PM. Peak activity was shown in between 4PM to 6PM

which could have to do with team meetings finishing around that time. Additionally, the Trello activity did not stop until past midnight.

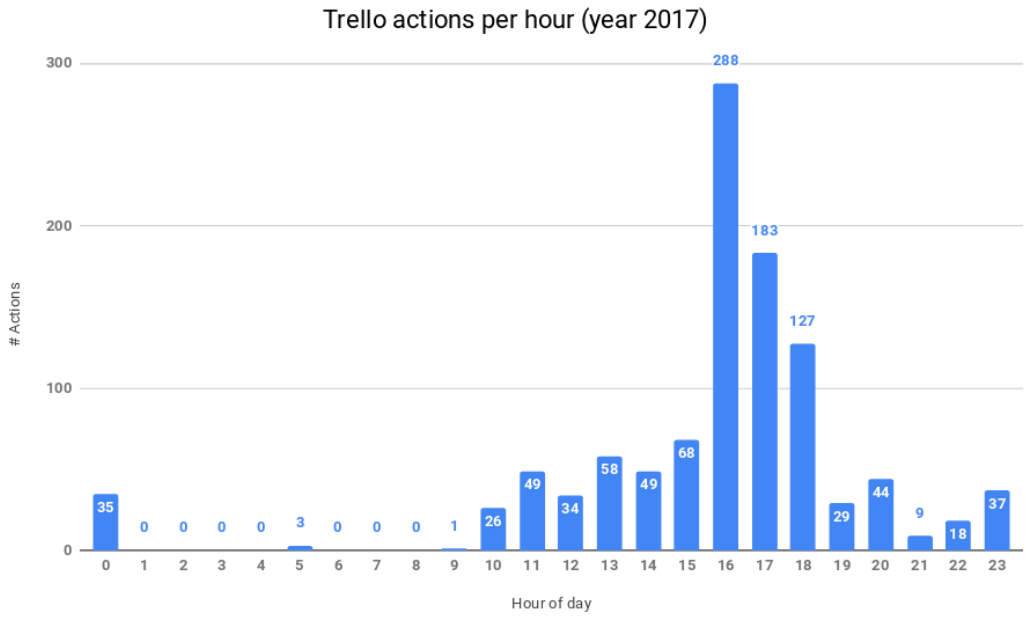


Figure 5.8: Distribution of Trello actions by hours on the year 2017.

Furthermore, by checking which Trello boards had more activity, the most worked topics could be investigated (with the supposition that there is a direct correlation between those aspects, which we can presume but not be sure about without deeper analysis, e.g. reading each Trello card). We observed that the *RoboCup Tests* board had the most activity which made sense as the priority was to have tests working for the competition. The boards *Team* and *Robot Skill Development* followed in activity. The *Pepper Robots* board had the least activity, probably caused by not having time to analyze the platform deeply enough, or, by the tasks in that board being mixed up with development for RoboCup tests or specific skills of the robot.

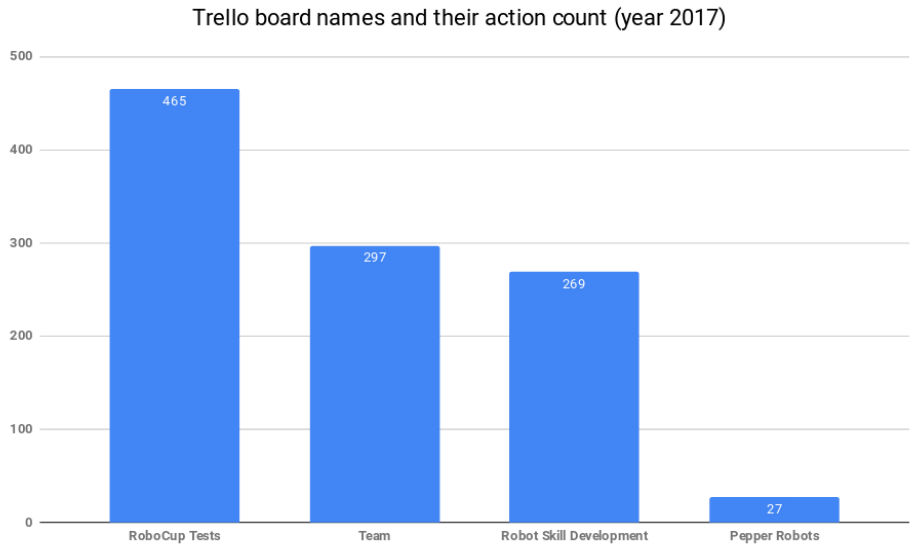


Figure 5.9: Distribution of Trello actions by Trello board on the year 2017.

Finally, checking the distribution of authorship of the actions in Figure 5.10 (anonymized) there was one author having most activity (most probably creating and updating most cards in the team meetings) but the rest of the team followed a smooth decline in contributions.

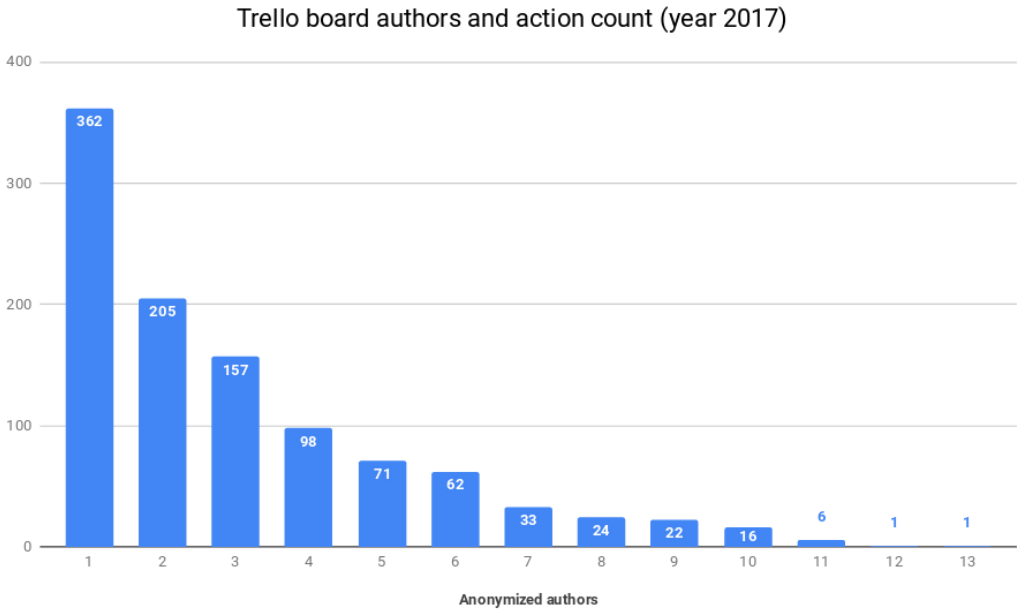


Figure 5.10: Distribution of Trello actions by anonymized authors on the year 2017.

5.6.1.2 Git Commit data

The Git data showed steady activity, increasing as the competition got closer, with even more activity in the competition. Significant teamwork was shown. There were some tasks that needed a lot of work with only one expert able to take care of those jobs.

From the Git repositories point of view we observed steady activity (measured by number of commits per day) since the start, as can be seen in Figure 5.11. As a curiosity, it was easy to observe a decline in commits on the days previous to travel to the competition (the competition started on 25th of July up to the 31st of July) and a large increase in the amount of commits during the competition.

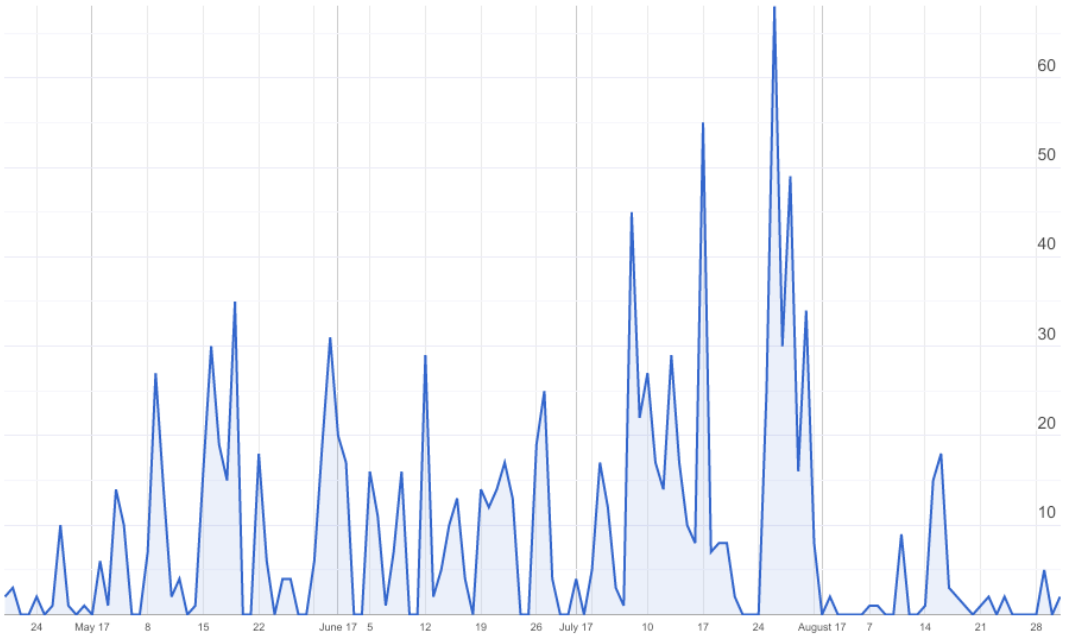


Figure 5.11: Activity (commits per day) on the year 2017.

Moreover, looking at the commits divided by months on Figure 5.12 it became apparent that there was a similar amount of activity in May and June, and a peak in July. The team did a considerable push in the last weeks and during the competition.

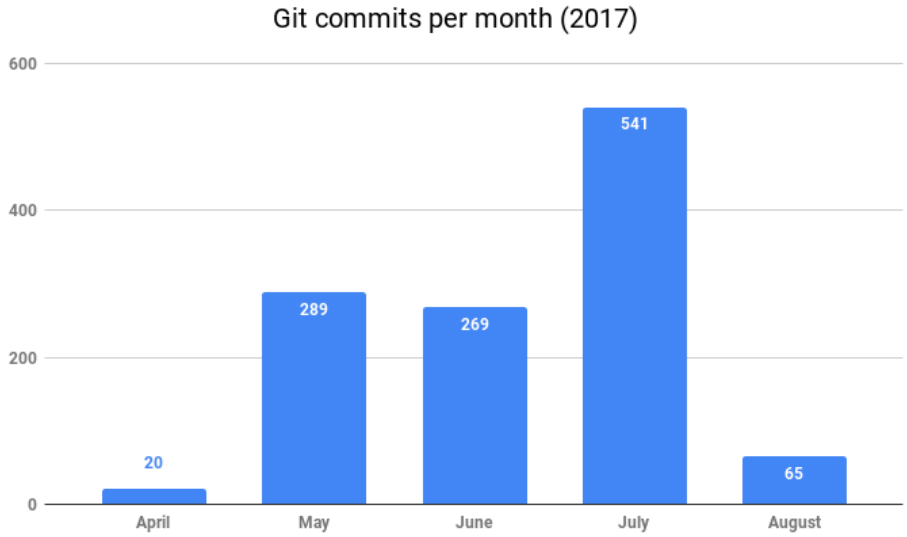


Figure 5.12: Commits per month on the year 2017.

In contrast with the Trello data where most activity happened on Tuesdays, the team meeting day, checking the commits by weekday in Figure 5.13, shows a steady amount of commits every day of the working week, with some activity also on the weekends.

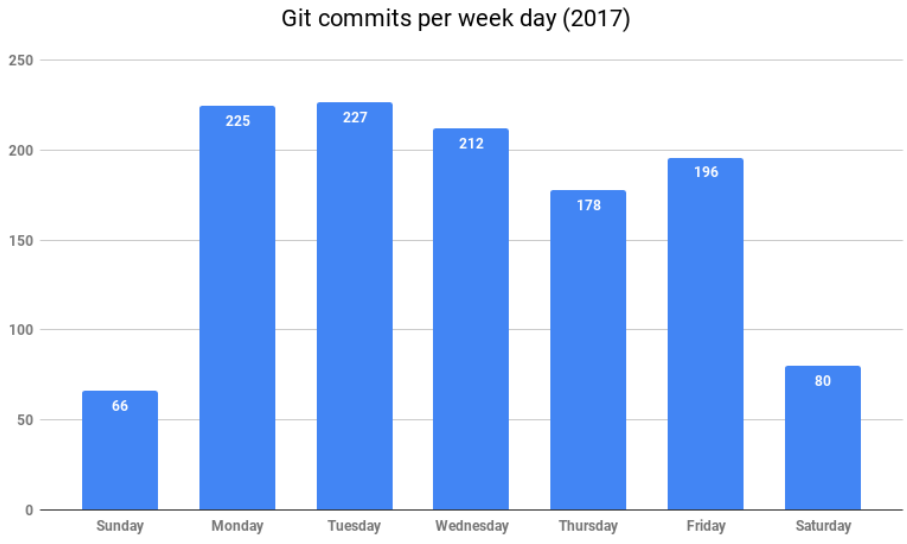


Figure 5.13: Commits per week day on the year 2017.

The commit activity per hour of the day was similar to the Trello data as can be seen in Figure 5.14. Activity started around 9AM until peaking at 6PM. Then it continued

until midnight. We observed two peaks: at 1PM and 6PM. These match with the team members going for lunch or leaving for the day. Some activity happened even during late hours in the night, showing a rather unhealthy but probably necessary push to finish work.

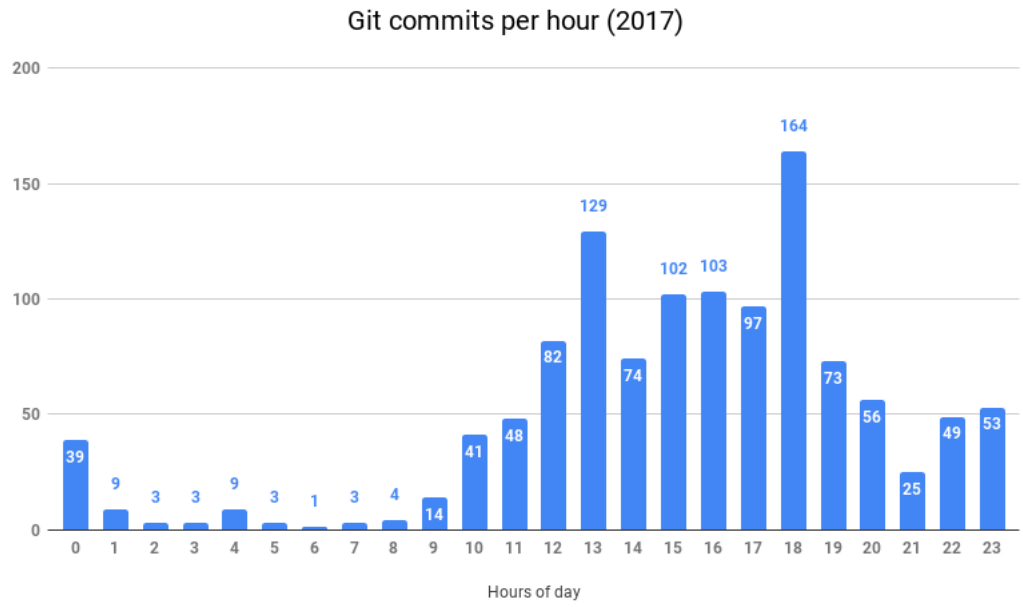


Figure 5.14: Commits per hour on the year 2017.

Moreover, to analyze teamwork the plot in Figure 5.15 showcases the name of the repositories with, in blue, the number of authors for a given repository and, in red, the number of days with commit activity for that repository. The chart is sorted by days of commit activity and it is filtered by only showing repositories that have more than one author and more than one day of activity. This way, we avoid taking into account repositories that were not meaningful. We can observe in this chart that there was a smooth decline in activity in the repositories which followed closely the number of authors. Additionally, by checking the name of the repository and analyzing its context, insights were extracted:

- Most of the team participated in updating the wiki that hosted documentation for the team. Moreover, it had the most activity days, showing that documentation was important.
- The next item with a similar amount of participation and activity was the project

called *robocup_tests* which contained the implementation of most Stage I tests. Hence, implementing the tests was important, and it was done as a team effort.

- *UTSUnleashedTools* and *gpsr* contained code that was used in multiple places in the codebase. They show a lot of activity (to appeal to people's needs and bug resolution) or many authors (everyone added the parts they had expertise on).

- Both *pepper_move_base* and *people_tracking_stuff* were related to the robot navigation, a important skill needed for the competition. Less authors were present here (probably related to the expertise needed to work on this field) but a considerable amount of activity was shown.

- It is meaningful to also mention the *pepper-monitor* repository. It showed 9 days of activity by two authors. The team considered it important to have a satisfying user experience and the tablet interface was a key element of this, as shown in these statistics.

- The rest of the repositories did not present meaningful data. *qimate* provided a common interface building up on the robots *qi* API and *stuffed_pepper* built on top of that to provide easier-to-use versions of the robot API. *naoqi_driver* was the ROS driver from the manufacturer of the robot which needed some patching for our use-case.

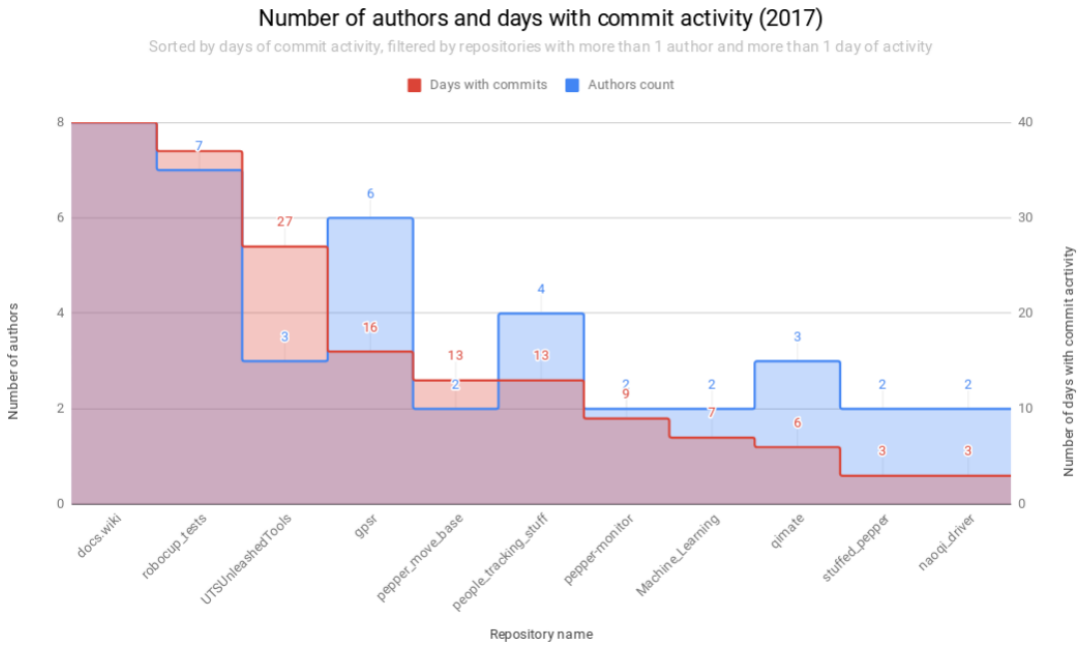


Figure 5.15: Number of authors and days with commit activity per repository, sorted by days with commit activity and filtered by more than 1 author and more than 1 activity day.

On the other hand, we analyzed the work that was done individually instead of by teamwork. This can be seen in Figure 5.16. The following insights were extracted:

- The first three repositories: *pepper_docker, on_boot_goodies and pepper_ros_setup* (and also *pepper_bringup_magiclab*) were about setting up the robot for RoboCup development. *pepper_docker* dealt with building all the dependencies necessary for ROS and also ROS itself with *pepper_ros_setup* being the initial implementation of this system. *on_boot_goodies* dealt with the software to be launched on the boot of the robot with *pepper_bringup_magiclab* containing specific configurations related to ROS bringup. These projects were done by the lead developer as it needed profound expertise in Linux operating systems and ROS. There were no other team members with such expertise available or time to teach anyone. Furthermore, it was a time-consuming task as compilation of some modules would take hours.

- The rest of the repositories were projects used by a single author for some RoboCup test or experimental approaches.

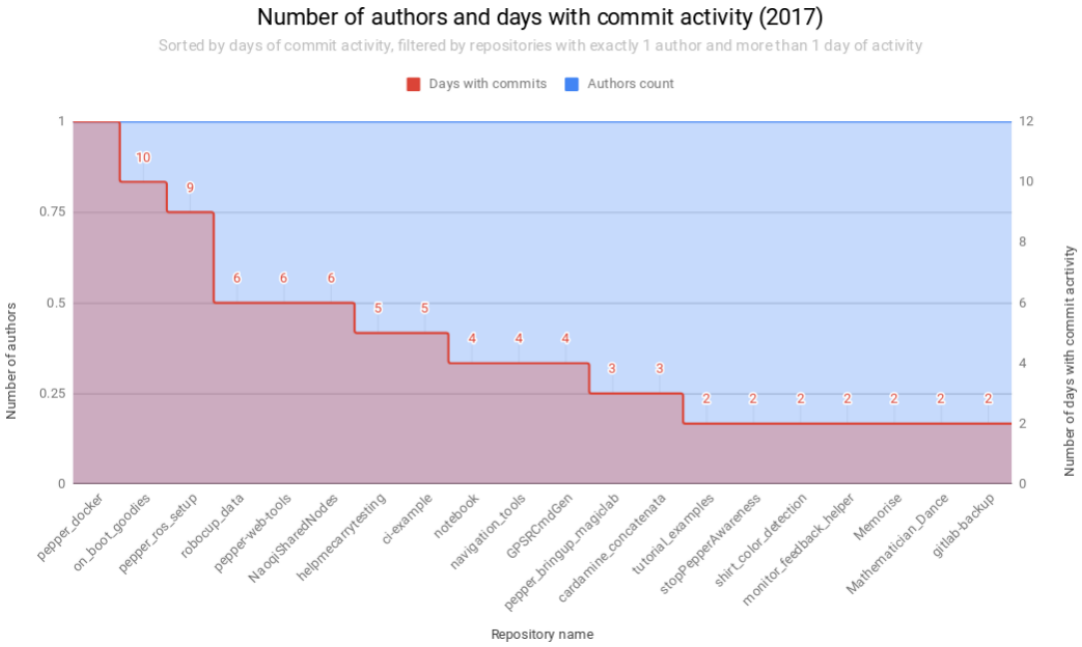


Figure 5.16: Number of authors and days with commit activity per repository, sorted by days with commit activity and filtered by exactly 1 author and more than 1 activity day.

The average number of authors in repositories with more than one author and more than one day of commit activity came up to 2.3 authors per repository. Taking also into account that there weren't many solo projects, it can be interpreted as an indication of teamwork. On average, more than two people worked on the common repositories.

5.6.2 Team Retrospective

In this section the team's retrospective was summarized. The original retrospective document for the year 2017 can be found in the section B.1 from the Appendix B.

The first question of the retrospective was "What worked well?" about the positive facts from the preparation process and the competition itself. Roughly half of the answers were about the competition event, highlighting the successful team collaboration, well prepared on-site technology resources, and positive feedback on giving presentations in the poster session, open challenge and finals. The other half of the answers were about the preparation for the event. The team regarded as positive the ORTs and general practice runs with focus on scoring for Stage I, using GitLab as a collaborative coding platform, the planner system created for the GPSR test, the tablet interface providing

feedback and tooling for the team, a shim layer to ease the usage of the robot APIs, and the human clothing analysis skill.

The next question was “What did not work well?” about the negative facts in the same context. The team documented issues related to the preparation for the competition: a lack of preparation for Stage II, and the presentations in the open challenge and the finals. Furthermore, the team reported dysfunctional coding practices, with git submodules not providing satisfactory dependency management, a lack of templates and best practices for code, general low code quality, and abuse of complex software frameworks. Finally, it was noted that the team lacked focus on the social aspect of the robot and the competition. In regard to the competition event, concerns were raised about the team wellness due to the long hours in the venue and stress from rushed work. Issues were documented about difficulties in the management of the robot’s running software, and specific problems were written down with regard to the speech recognition, navigation, object recognition, and "*aliveness systems*" of the robot.

The last question was “What should we do next?” about proposals on how to improve the next year of development and competition performance. A software development plan was proposed where the team will prioritize robot skill development, increase the amount of ORTs, and aim to have something prepared for every competition test including presentation material. A list of coding practices to be improved was reported in relation to the topics of coordination of packages and dependencies, standardization of the team’s development environment, code repeatability and code analytics. Furthermore, approaches on how to manage existing code were listed, keeping working components and extracting value from the rest. Finally, items addressing missed opportunities were added.

5.7 Reflection

The reflection is composed of the findings, which include the same substructure of three sections as explained in chapter 4, answers to the action research questions stated in this cycle, new questions to start the next cycle, and next steps to be implemented.

5.7.1 Findings

The findings stem from a review of the action research cycle and the team’s retrospective, they aim to describe facts learned in this cycle.

5.7.1.1 Team Management Processes

The backlog boards in Trello were widely used as the place to share approaches and progress during the first month. Their usage fell as the competition approached. In the competition a physical whiteboard with post-its replaced it effectively. It allowed to quickly identify what was going on, what needed focus afterwards and who was doing what. It also had the advantage of being offline.

ORTs were a key element to the successful development of the RoboCup tests. These events forced the team to prepare for a realistic scenario so they tested their code focusing only on situations that were likely to happen. The team also had additional pressure to deliver. The Trello and Git data show that in the days surrounding the team meetings, Monday to Wednesday, there was an increased amount of activity. The ORTs had the same effect but, as perceived by the lead developer, in a more focused fashion. These events often showcased failures in the systems, which highlighted what needed further work.

Lack of preparation for Stage II could be perceived as a mistake. This is so because scoring on Stage II could potentially have given us the victory. But the state of the tests of Stage I were not robust enough. Dedicating time to Stage II had a high chance of making the team unable to even get to Stage II. The development time was short and trade-offs were necessary.

5.7.1.2 Team Software Development Processes

Collaboration between team members and within the team as a whole started this year. Teamwork acquired an important role. New knowledge was discovered at the same time by multiple teammates working together. Also, knowledge transfer from the experts in software development towards the junior developers began. The competition itself showcased how the team learned to work together efficiently and in a enjoyable environment.

Pair programming was used by some team members. The team members that practiced this technique with the lead developer said that they found this to be an enjoyable practice. Their knowledge of how the parts of the system that were worked on together using pair programming became additional evidence of knowledge transfer within the team.

It was discovered that a balance between the team members preferences on how to work and what will be better for the team in the long run, needed to be found. Sometimes

the team was more comfortable with an approach different from one believed to be more beneficial by the lead developer, and it needed to be accepted. In this year of development there was not enough time to prepare, teach, and encourage better approaches.

5.7.1.3 Technical Approaches

The robot's tablet interface proved beneficial, not only enabling debugging, but development and testing also benefitted from it

Teaching inexperienced programmers in a short time: a new programming language, how to use libraries, how to use a complex framework like ROS; all at the same time proved too difficult and was ultimately unsuccessful. More preparation time is needed.

Other teams used WIFI and cloud services which provided an advantage. It must be explored for the following year when it isn't a trade-off with efforts in more critical parts of the system.

Using the robot's laser as a means of localization didn't perform at all, and needs further research or an alternative.

Developing without the physical robot was complicated. A better setup, maybe with a simulator, is to be worked on.

During the last weeks of development, within the ORTs, focusing in scoring over other topics like reusability of code did provide better scores. It also allowed a shift in the mentality from trying to build complex systems to only focusing on the performance in a single RoboCup test.

5.7.2 Answers to AR Cycle Questions

After a review of the cycle and the team's retrospective, the proposed action research questions were revisited. This cycle made the researcher learn about the processes necessary to set up a new team to compete in RoboCup@Home SSPL, and these are reproduced next.

The rulebook was read to extract the most achievable tests and prioritize their development. Team meetings were setup to discuss all the necessary matters, the team members were asked about their expertise and their interests to find roles that align with both them and with the team's needs. At the same time, the team explored the capabilities available in the robot and what was possible to be developed from those capabilities.

The software stack was chosen based on discussion with the team members about their roles with regard to the predicted RoboCup tests to be implemented and the exploration of the robot capabilities. The software stack included the programming language that fitted best with the profile of the team, was aligned to the frameworks and libraries found to be widely used to solve problems in similar domains, and with special regard for those domains in which team members had expertise. It is noteworthy to mention that this software stack was a starting point and was subject to change. Afterwards, coding standards were chosen by how likely they were to improve the team's progress while not detracting from their development experience.

With regard to team management, the team kept communication open while appealing to the leader's management style, aiming to make the process comfortable for all members. As previously mentioned, pleasing the preferences of team members on what tasks to work on was a priority, and feedback about their satisfaction in working on those tasks was gathered by asking occasionally during work hours and with open discussions in team meetings when necessary. The tool to track the backlog of tasks was changed as a response to the dissatisfaction expressed by some team members, including the lead developer. Having the tools align to the management style was important, but the lead developer realized that having a mental model of the team's progress was more important. Following this realization, as the competition approached and team members worked in a larger number of tasks in parallel, the usage of Trello fell down. This had consequences in team members having a harder time understanding the progress of other team members.

Finally, it is important to note that it was found that all the questions raised during this action research cycle were tightly tied together, decisions or opinions in one topic affected others, as team members acted on these. Seeking balance was an ongoing effort.

5.7.3 New Questions

This cycle raised the following research questions, based on the following observations:

- Team management:
  - How to find new team members?

    We struggled to get enough team members and we anticipated the need for more team members for the next year.
  - How to match people with the best tasks for them?

We tried to discuss with each team member what they would like to work on but the lead developer believed this process could be improved.

- Which software development methodology fits RoboCup@Home development better? Is there one?

  The development this year was done mostly ad-hoc and the lead developer believed the team could benefit from practicing some kind of software development methodology.

- How to transfer knowledge effectively?

  As team members developed their parts of the system and learned the different topics and tools, it was not clear how to share this knowledge so efforts would not be duplicated or hard to join together.

- How early on should ORTs run?

  Testing was performed during the development but it took several weeks before the testing was similar to the competition situation. For the next year, with the rulebook possibly being modified and more time to develop, it is to be researched when to start doing ORTs.

- How to improve the usage of the backlogs?

  The team used backlogs to store tasks first in Asana, then in Trello, and finally using a whiteboard with post-its. This provides evidence that further work in this regard must be done.

- Coding practices:

  - How to make the development easier?

    Unexperienced developers struggled to perform tasks related to networking and software management, which delayed their contributions. We aim to improve this situation for the next cycle.

  - Which coding standards must be followed?

    Coding standards present advantages for sharing code but they add an overhead to coding. The lead developer believed there was not enough time this year to put pressure on this topic, so for the next year it must be investigated further.

  - How to best unify the coding efforts?

Some parts of the system were built in parallel, doubling efforts instead of unifying them in a better code base. Practices to improve this will be researched.

  - How to create the best APIs for the team to use?

    As different team members had different expertise and preferences in how to code, creating APIs for everyone to use was challenging.

- Technical approaches

  - What should be developed first? How to prioritize?

    This year the lead developer took many decisions for the team based on his experience. For the next year a more open and data-based approach should be taken.

  - What should be kept from the robot APIs? What should be re-developed?

    A significant amount of work on developing and testing APIs was done this year, but the code quality was low.

  - What architecture fits best our problem?

    The architecture for this year was ad-hoc, for the next year it will be prepared further.

  - How to increase ROS adoption?

    As ROS presents a wide variety of tools, libraries, and concepts useful for developing robotics capabilities and applications, it is desirable for the team to embrace it.

5.7.4 Next Steps

This section describes the aims for the next action research cycle, based on the review of this cycle and the team's retrospective, using the same structure from chapter 4.

5.7.4.1 Team Management Processes

For the next cycle, new team members needed to be recruited. To do this, some kind of recruiting event must be held. The criteria to choose these new team members and the number were to be defined with the project managers.

Improved usage of backlogs, both for long term tasks and current tasks must be done. Research on the tools available in Trello for this goal must be performed. A fine grained

description of necessary robot skills must be developed. Tasks must be generated and followed from those. The whiteboard approach during the competition worked well too, using it alongside Trello was to be explored.

ORTs would be held sooner and their structure would be formalized.

Careful planning as to when to work on Stage II tests would be taken into account.

Further training and resource preparation would be prepared, including code templates and coding best practices.

5.7.4.2 Team Software Development Processes

For the next cycle the development tools available for the team were to be improved. Finding common issues and addressing them with documentation and tutorials or workshops would be done.

The practice of pair programming would be kept and further explored. The same would be done with dividing into subgroups parts of the development or the RoboCup tests. When and how to divide the team in subgroups would be researched taking advantage of having a full year of development ahead.

Finally, as the repository containing all RoboCup tests with submodules approach had reported problems, a distributed approach, with every test being in a different repository and avoiding the usage of submodules, would be taken for the next cycle. This would affect the structure of the code and the methods to share and reuse it, hopefully improving the development experience.

5.7.4.3 Technical Approaches

For the next cycle an analysis of the necessary skills for every test would be done. This way a table would be created to guide the prioritization of the development of these skills and their features.

Programming by configuration and also programming via graphical interfaces (block dragging) would be researched. Having robust prepared skills that can be tested separately increases the trust in these skills. Programming by configuration could enable less technical team members to help on the project.

Infrastructure using the ROS middleware would be further built and the team would be trained on it.

5.8 Possible Guidelines

The most important findings and reflections from the year of development will be summarized in a set of guidelines. These can be refined and extended every year. These guidelines follow:

- Perform end-to-end testing. Simulation of the full competition including setup on a previously unknown place, strict schedule and timing, strict scoring, by a strict referee, naïve users and other elements that otherwise may be overlooked. Examples of other elements can be: networking issues, environmental noise, unavailable team members or unusual lighting.
- Find a balance between what the team wants and what is believed to be the best for the team regarding development processes. This is something to be re-evaluated during the development year. Topics such as coding standards, testing standards, documentation, deployment strategies and code sharing approaches fall into this category.
- Provide the best tools available for the job. Development tools, frameworks, simulations and hardware are topics that fall into this category. References for these can be taken from other teams or industry.
- Ensure teamwork is possible and encouraged. Set up a backlog of tasks and distribute them. Promote grouping team members to work more effectively. Examples can be working on a competition test in groups or engaging in pair programming.

CHAPTER 6

ROBOCUP@HOME SSPL: YEAR 2018, 2ND PLACE

As in the previous chapter, this chapter follows the structure of the diagram in Figure 4.1 which itself follows the structure of Action Research (AR) cycles in Figure 2.2. Every AR cycle is composed of a year of preparation for the competition, participation in it, and reflection on the process itself.

In each section, the AR cycle setup is explained: the framework of ideas, the methodology, and the area of concern for that cycle are first discussed as per the structure explained in chapter 2 and chapter 4. The context for the year follows. Its main themes are the rule and competition changes of that year and the team composition of that year.

Following, the software development plan is presented. It is analyzed in three subtopics that will be repeated in each new subsection: team management processes, team software development processes, and technical approaches.

Afterward, the software development implementation presents the same subsections but in the context of plans put in practice and their iterations.

Then the competition participation is showcased, again with the same subsections, presenting the results as competition outcomes for that edition.

In the next sections, we find the post-competition data collection and retrospectives. Here the available data about the year is detailed.

At the end of the chapter the reflection section summarizes and discusses the findings (following again, the same three subsections structure), the new questions that came up from these, and the next steps to follow for the next action research cycle. This leads to the end with a proposal of guidelines.

6.1 Action Research Cycle Setup

The research questions for this action research cycle stem from what was learned in the previous cycle and the new questions found. Here they are repeated along with their accompanying observations.

- Team management:
  - How to find new team members?

    We struggled to get enough team members, and we anticipated the need for more team members for the next year.
  - How to match people with the best tasks for them?

    We tried to discuss with each team member what they would like to work on, but the lead developer believed this process could be improved.
  - Which software development methodology fits best RoboCup@Home development? Is there one?

    The development this year was mostly ad-hoc, and the lead developer believed the team could benefit from practicing some kind of software development methodology.
  - How to transfer knowledge effectively?

    As team members developed their parts of the system and learned the different topics and tools, it was not clear how to share this knowledge so efforts would not be duplicated or hard to join together.
  - How early on should be Operational Readiness Tests (ORTs) run?

    Testing was performed during the development, but it took several weeks before the testing was similar to the competition situation. The next year, with the rulebook possibly being modified and more time to develop, it is to be researched when to start doing ORTs.
  - How to improve the usage of the backlogs?

    The team used backlogs to store tasks first in Asana, then in Trello, and finally using a whiteboard with post-its. This evidences that further work in this regard must be done.
- Coding practices:

  - How to make the development easier?

    Unexperienced developers struggled to perform tasks related to networking and software management, which delayed their contributions. We aim to improve this situation for the next cycle.

  - Which coding standards must be followed?

    Coding standards present advantages for sharing code, but they add overhead to coding. The lead developer believed there was not enough time this year to put pressure on this topic, so it must be investigated further for the next year.

  - How to best unify the coding efforts?

    Some parts of the system were built in parallel, doubling efforts instead of unifying them in a better codebase. Practices to improve this will be researched.

  - How to create the best APIs for the team to use?

    As different team members had different expertise and preferences in how to code creating APIs for everyone to use was challenging.

- Technical approaches:

  - What should be developed first? How to prioritize?

    This year, the lead developer took many decisions for the team based on his experience. For the next year a more open and based on data approach should be taken.

  - What should be kept from the robot APIs? What should be re-developed?

    A significant amount of work on developing and testing APIs was done this year, but the code quality was low.

  - What architecture fits best our problem?

    The architecture for this year was ad-hoc, for the next year it will be prepared further.

  - How to increase Robotics Operating System (ROS) adoption?

    As ROS presents a wide variety of tools, libraries, and concepts useful for developing robotics capabilities and applications it is desirable for the team to embrace it.

6.2 Context

The context of this cycle/year will be explained in this section. This includes the specific rules of the competition and the characteristics of the team.

6.2.1 Rule and Competition Changes

This year the competition is scheduled for the 16-22 of June of 2018 in Montreal, Canada. There are roughly nine months to prepare for the competition. Compared to the previous year that had only three months, it allows the team to prepare better.

The rules did not present significant changes from the previous year, with only minor corrections appearing in the 2018 rulebook [19]. The Enhanced Extended General Purpose Service Robot (GPSR) (EEGPSR)[1] test was simplified, both in categories and in scoring. Additionally, a bronze (3rd) place was added for leagues with enough participants.

6.2.2 Team Composition

The team was composed of 18 people, as seen in Table 6.1. Team members are kept anonymous. Recruiting was performed during the months of October and November by creating a workshop with activities and topics similar to be found in RoboCup tests. This was based on the belief that students would join based on their motivation to work on such a project. Three interns joined the team with scholarships offered. Another intern joined later to perform a project for a subject they were enrolled in.

The team was larger than the previous year. More than half of the team was participating for the second year in a row. This year, 12 members were in the role of developing software. The team also included two team members working on marketing for the team as part of their degrees. A psychologist, being part of the lab, also gave their input to the project.

[1] EEGPSR is a test where the robot can be asked to do anything that has ever appeared in previous rulebooks. An open-ended and hard test.

#	Team Role	SW Expertise	Background	Time Dedication
1*	Project Manager	–	Lab Director	On demand
2*	Project Manager	–	Project Manager	14h/w
3*	Assistant	–	Media Specialist	On demand
4*	Lead Developer	Expert	Robotics & CS	32h/w
5*	Developer	Expert	Computer Science	20-40h/w
6*	Developer	Expert	Lab Co-Director & CS	20-40h/w
7*	Developer	Expert	Web Developer & IT	20-40h/w
8*	Developer	Advanced	Computer Science	20-40h/w
9	Developer	Advanced	IT & CS (Intern)	20-40h/w
10	Developer	Advanced	IT & CS (Intern)	20-40h/w
11	Developer	Advanced	IT & CS (Intern)	20-40h/w
12*	Developer	Moderate	UX & Multimedia	On demand
13*	Developer	Moderate	CS & AI	20-30h/w
14	Developer	Beginner	IT & CS (Intern)	10-20h/w
15*	Developer	Beginner	Mathematician	On demand
16	HRI & Assistant	–	Psychologist	On demand
17	Marketing	–	Marketing & Biz	On demand
18	Marketing	–	Marketing & Biz	On demand

Table 6.1: Team composition table for the UTS Unleashed! team in 2018. Team members marked with an asterisk (*) are on their second year of participation in the team.[2]

The interns were bachelor students in Computer Science and/or Information Technology. The team expected their background to be useful even though their exposure to actual software development was limited. They were also new to the field of robotics and to RoboCup itself.

The rest of the developers fit the description of the previous year being mostly PhD students and staff from the lab.

[2]CS stands for Computer Science, IT stands for Information Technology, UX stands for User Experience, AI stands for Artificial Intelligence, Biz stands for Business.

A year-long plan based on our previous reflection was created. The lead developer divided the development in an early phase where the team would research new and better ways of solving challenges, a middle phase where the team would develop and implement base skills while the rulebook came out, and a late phase where the priority was implementing the tests and scoring points.

6.3 Software Development Plan

As in the previous cycle, this section explains the software development plan for this edition of the competition. It includes the team management processes, the team software development processes and the planned technical approaches.

6.3.1 Team Management Processes

This year the spirit of the team management processes was influenced by standard software development methodologies. The lead developer and project managers aimed to have more structure while monitoring the team's satisfaction.

6.3.1.1 Team Management Tool

Trello was kept as the online tool to track work. This year six boards were created with the topics:

- General: thought to be for general topics not fitting into other board topics.
- RoboCup 2018 Preparations: acting as a kind of roadmap for our Project Managers.
- Software Engineering: It was identified from the last year that our (non-existent) architecture and technical approach could benefit from taking inspiration from classical software engineering practices, so a board for these tasks was created.
- Skills: as the previous year, for Robot Skills.
- Challenges: as the previous year, for RoboCup Tests.
- Finals: for proposal and tracking of what to do for the Finals RoboCup test.

The lead developer discussed with key team members what boards were needed, taking into account the boards that got actual usage the previous year (RoboCup Tests,

Robot Skill Development, and Team) and in which topics we wanted to focus on and keep track of.

It was proposed to use checklists in the Trello cards to keep track of tasks, and to set deadlines where they applied.

The lead developer, with further discussion with one key team member, created for the first time a spreadsheet with the set of skills to be developed in order to prioritize the development. It was created by reading the rulebook and extracting the necessary skills for every test. It was divided into topics such as navigation or speech recognition, and specified abstract definitions of the skills, including concise implementation descriptions of these. Additionally, these contained which RoboCup tests would use these skills. This allowed the team to prioritize the work by quantifying the number of tests that needed a skill. An example of the spreadsheet can be seen in Figure 6.1.

Category	Abstract Skill	Concrete Skill	Specific function call	Total	Inspection	Cocktail Party	Help Me Carry	SPR	GPSR	EEGPSR	Restaurant	Tour Guide	Open Challenge
tablet interface	monitor app			9.5	maybe	yes	yes	yes	yes	yes	yes	yes	yes
voice interface	text to speech		def say(text): """Say text. :param str text: Text to say. """	8	yes	yes	yes	yes	yes	yes	yes	yes	
	collision avoidance			8	yes	yes	yes	yes	yes	yes	yes	yes	
	multimodal speech	tts	def hri_say(text, exageration_gestrues=0.2, captions=yes, emojis=yes)	8	yes	yes	yes	yes	yes	yes	yes	yes	
		captions	def follow_tts(text) ********follows text to speech v	7.5	maybe	yes	yes	yes	yes	yes	yes	yes	
	speech to text			7		yes	yes	yes	yes	yes	yes	yes	
perception	people perception	detect person	def detect_people(): """The method initialize the people detection processes of the robot. The detected people info are stored in the memory/belief system. When a person is detected characterizing features of that person are extracted and stored and a tracker for that person is initialized. """	7		yes	yes	yes	yes	yes	yes	yes	
		stop people detection	def stop_people_detector(): """It stops the detection of people. """	7	no	yes	yes	yes	yes	yes	yes	yes	
		look who is visible	def visible_people(): """It reads the memory/beliefs system and returns a list of peson_ids of the visible people in the robot current FOV. """	7	no	yes	yes	yes	yes	yes	yes	yes	
	localization			6.5	yes	maybe	yes		yes	yes	yes	yes	
	move to absolute location		def navigate_to(x, y, is_absolute=True): """Navigate to a coordinate (x,y) either absolute """	6.5	maybe	yes	yes		yes	yes	yes	yes	

Figure 6.1: Excerpt from the robot skills spreadsheet to prioritize skill development.

Slack was proposed as a new chat solution for collaboration. It aimed to have people connected so that remote collaboration would be easy.

6.3.1.2 Team Meetings

Mondays were chosen to be the day where team members were expected to come to work together in the lab to encourage teamwork and knowledge sharing, there was no proposed day previously.

Design thinking workshops[3] were introduced and organised for mid-January for the Open Challenge and the Poster presentation.

Monthly ORTs were also scheduled for the end of February given the success of these in the previous year.

Team meetings stayed in the same format from the previous year. The lead developer would lead the meeting, and the team members would share their progress.

6.3.1.3 Task Assignment

As outlined earlier, a detailed spreadsheet with the required set of skills needed for the competition was created. It was initially inspired by the GPSR test as it contained all of the possible skills needed. It was detailed to the level of function call definitions and, where possible, the description of the skill's usage.

From the spreadsheet, a high-level plan was created for the year. This plan was initially created by the lead developer and refined by discussing it with key team members. Trello tasks were created for the different elements of the plan. Every topic had a mentor to guide that part. New team members were assigned tasks that would build up their skills and expose them to the different areas of robot programming, so they could find what interested them most.

The initial tasks were related to creating a better low-level understanding of the robot sensors and capabilities. For example, for the navigation topic, the initial tasks were to: get data from the laser of the robot and understand its behavior (minimum range, maximum range, noise in the data, etc); write our own driver; and then compare it to the one used in the previous year. This allowed the new team members to explore the topic by themselves, familiarize themselves with the tools, and ask for help when needed.

These tasks also aimed to encourage proactivity on the part of the new team members, as the goals were purposely, only described vaguely. For example, the goal of testing the laser data was initially just described as "Test the quality of the laser data" with a set of links with documentation about it and some encouragement to ask for help whenever needed. When contacted for help, more descriptive tasks would be created in collaboration with that team member. For example, "Get a flat wall in front of the robot

[3]Design thinking workshops are hands-on sessions focused on collaboration and problem-solving. They aim to teach people how to problem solve and to foster innovation and teamwork. These sessions were commonly used as a tool in the laboratory to create approaches for challenges.

and record the laser values at different distances". From there, new opportunities to propose tests and insights by the team member would arise.

6.3.2 Team Software Development Processes

This section presents the practices and tools regarding how the team was to develop software.

6.3.2.1 Coding Standards

Team members would update the spreadsheet with the different skills to have a centralized documentation point for these robot abilities.

This year the team was to adhere more closely to two standards: Python coding standards and ROS naming standards. However, no naming scheme for package names was pushed. The lead developer believed that by doing so, the team would avoid the problems that arose the previous year, where it was hard to share code between team members.

6.3.2.2 Coding Tools

Powerful Alienware[4] laptops with Nvidia Graphical Processing Unit (GPU)s were acquired to make the team able to take advantage of state of the art deep learning technologies.

Due to team members' previous experience and the quality of available documentation, the Theano [99] library with TensorFlow [100] backend was chosen as the deep learning framework to be used.

6.3.2.3 Social Coding Practices

Mentoring and pair programming were used, especially at the start of the year, to guide new team members. Workshops were held to introduce topics like ROS and team-made packages to improve code sharing and knowledge transfer. These practices were new considering the previous year similar efforts were made but in an ad-hoc manner.

[4] Alienware is a brand used by the Dell computer company for its range of high-end gaming computers.

6.3.3 Technical Approaches

As in the previous cycle, the technical approaches, or specific decisions made for the competition itself, are discussed in this section. These are related specifically to the RoboCup@Home Social Standard Platform League (SSPL) and describe the competition requirements, the robot's capabilities, and the software stack.

6.3.3.1 Competition Requirements

The team wished to involve itself in the evolution of the rules for the competition. To this end, team meetings started with a request to develop a "view of the Social Standard Platform League" for the laboratory. This would answer the question "What is our general view of social robotics?".

The tests chosen to participate in 2018 for Stage I were the same as in 2017. Their rules did not change from 2017, their description is available in chapter 5. Additionally, for Stage II the following tests were chosen:

- **Open Challenge**: In the Open Challenge, teams are encouraged to demonstrate recent research results and the best of their robots' abilities. This is an open demonstration with a presentation. The team decided to create a real application for the Pepper platform where the robot would detect an infant crying, and it would offer help when that was detected.

- **Tour Guide**: The robot guides spectators to the audience area and answers their questions after explaining what the @Home league is about. This test focuses on safe outdoor navigation, people detection, gesture recognition, unconstrained natural language processing, and Human-Robot Interaction

- **Restaurant**: This test focuses on online mapping, safe navigation in previously unknown environments, gesture detection, human-robot interaction, and manipulation in a real environment. The robot will need to create its own map from the environment and then move within it to handle human requests, such as delivering drinks or snacks, while people are walking around. As this test is performed with two robots (two teams, each with their own robot) in parallel, the robots will also have to avoid each other. The test starts with a client waving and calling the robot for its attention. The robot must approach the person and take their order. Then it must go back to the starting location and fulfill the order, and then navigate back to provide the order to the customer.

- **EEGPSR**: This test evaluates the abilities of the robot that are required throughout the set of tests in Stage I and stage II of this and previous years' RuleBooks. In this test, the robot has to solve multiple tasks upon request over an extended period of time (30-45 minutes). That is, the test is not incorporated into a (predefined) story, and there is neither a predefined order of tasks nor a predefined set of actions. The actions that are to be carried out by the robot are chosen randomly by the referees from a larger set of actions. These actions are organized in several categories targeting a special ability. Scoring depends on the abilities shown.

Some ideas were proposed for the Finals, but the development was left to be done in the competition itself.

6.3.3.2 Software Stack

The lead developer and the key team members involved in this decision had a new vision for this year's software stack. They believed GPSR was to be prioritized as developing for this test would provide the skills for most other tests. Furthermore, any test should be possible to be implemented as a GPSR variant. Navigation was the second-highest priority, as almost every test had a navigation component. Generic skills were to be developed into what was called SkillStates. A SkillState was defined as a class with an execution method that achieved one specific task, and that could be executed independently.

A SkillState was agreed to have multiple interfaces. It shall be able to be used in SMACH State Machines as used in the previous year, and also in plain Python code as a function call, and, finally, taking advantage of a planner interface so a set of skills could be orchestrated to create a test. The planner interface would allow the creation of plans in a human-readable specification (JavaScript Object Notation (JSON)/Yet Another Markup Language (YAML)) with a web-based graphical interface to be able to create, edit, store, run and monitor these plans. This kind of approach was seen to be used successfully by other teams in projects like PetriNetPlans [101] and ROSPlan [102]. The lead developer felt there was too much duplication of effort in the previous year, so this was thought to be a viable way to improve that situation. Additionally, having a graphical interface to create these plans would enable non-technical team members (as user experience designers) to be able to contribute more than before. The planner software stack diagram can be seen in Figure 6.2[5].

[5]Note that this diagram reflects the final implementation.

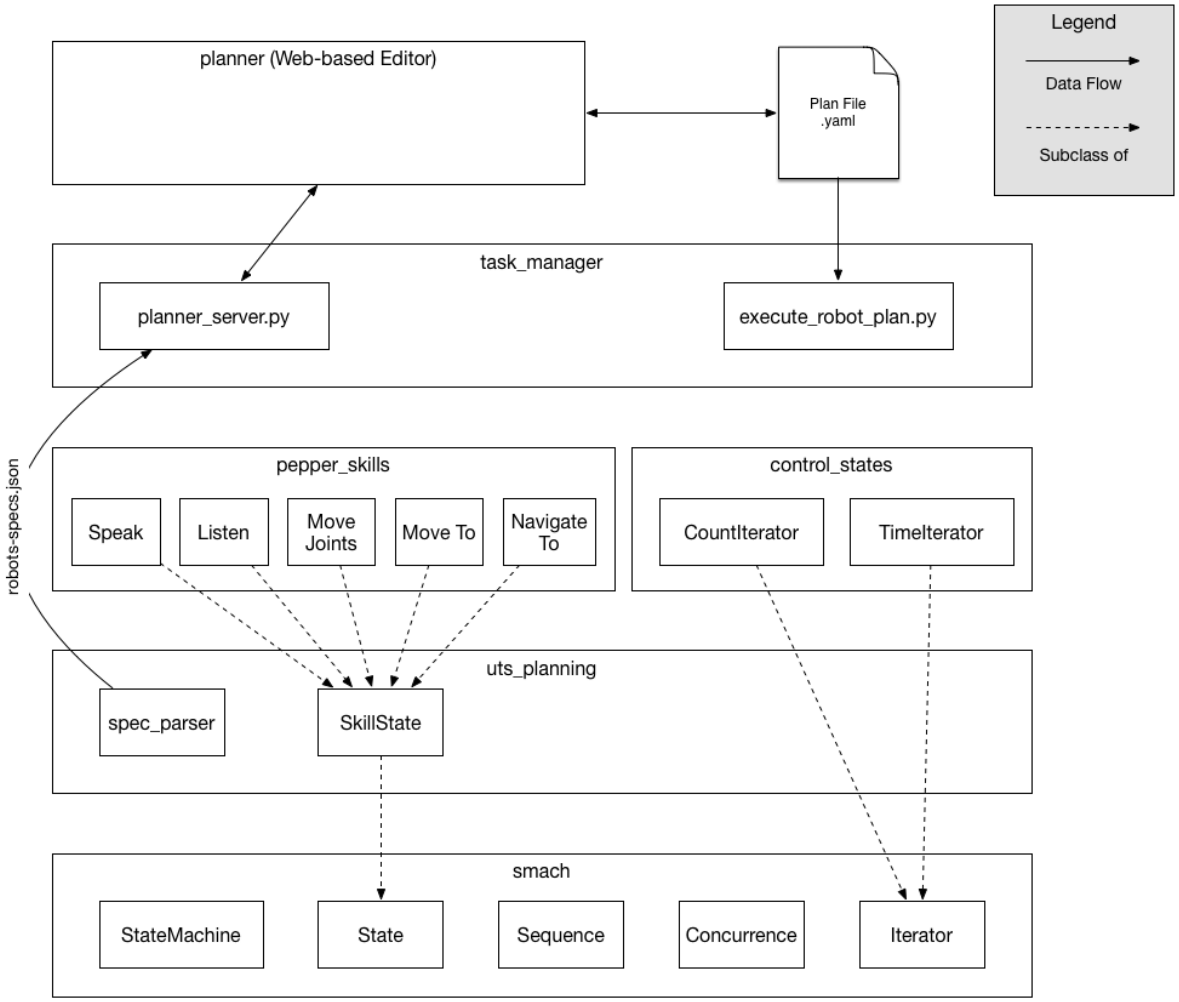


Figure 6.2: Diagram for the planner software stack.

An improved version of the base software building and deployment was planned. The previous setup was not well documented and could only run on a single machine. This was proposed to at least be converted to a virtual machine approach, but was considered even better if automated building and testing could be introduced.

Deep learning was to be used to take advantage of the latest state of the art networking giving capabilities like robust face recognition or people skeleton detection. Custom compilation of dlib [103][6] and TensorFlow [100][7] libraries for the Pepper platform would be needed.

[6] dlib is a C++ library focused on machine learning algorithms.

[7] TensorFlow is a library used for machine learning applications such as neural networks.

6.4 Software Development Implementation

The plans evolved during the year in more or less the direction expected until around March. Problems started to arise as there were discrepancies between the vision and implementation of the task planner software system. Features were requested but were not implemented or were approached in a different way that would not align with the users' expectations, the users being the rest of the team.

6.4.1 Team Management Processes

The team management processes for this year included the development of the team's vision and improvement over the processes discovered in the previous year.

6.4.1.1 Team Management Tool

Some Trello boards were successful in their usage, especially in the first half of the development time. Others were not.

- General: not used at all.
- RoboCup 2018 Preparations: Project Managers captured what was decided at meetings here, so it was used.
- Software Engineering: It was heavily used, especially by a pair of teammates.
- Skills: heavily used, especially at the start, as a lot of tasks created to introduce new members were followed here.
- Challenges: only used at the start, then ignored.
- Finals: same as the challenges board, only used at the start, and only briefly.

Lists with checkboxes for subtasks in Trello cards were added. They were used during the first few months of the development year; then they stopped being used.

This year the practice of adding specific deadlines for Trello cards was added. It was mostly used with people that could not come to work in the lab at the same time, or people that were lacking productivity, i.e., it was believed that their outcomes could be of a higher number or quality.

The spreadsheet described previously containing the complete set of skills required to be developed was not updated later on. Work simply moved to Trello. Most knowledge was shared in days working together in the lab.

Slack did not get any traction at all. It was not used.

6.4.1.2 Team Meetings

The work towards setting a common view of social robotics for the team, and the competition, stagnated. However, the team did participate in the creation of that year's rulebook via the lead developer being part of the technical committee of the competition, and therefore acting as a bridge in between the team and the competition organizers. One test proposed by UTS Unleashed! was incorporated into the rulebook for the 2018 competition.

There were meetings set every two weeks for most of the year to ascertain progress. There were monthly ORTs executed as planned.

ORTs started early on as planned. The first one was held at the end of February, then they were held monthly, also as planned. From April, they were held every two weeks. The last ORTs were focused on the robustness of scoring. ORTs included gathering together for free lunch for the attendees.

Table 6.2 summarizes the results of the ORTs of 2018. Tests in Stage I more or less improve during the year. Scores were lower towards the end after a high score in ORT number 5. Being able to consistently score (avoiding 0 scores) was more important than having a one-time high score. As we got closer to the competition, the team leader became a more strict referee and the environment was changed more often. This influenced the scores to drop. For example, a subteam may have their test tuned for a specific distribution of the arena. In the last ORTs, the arena would be changed (almost) every time.

The Help Me Carry test barely improved its scoring during the year.

Tests in Stage II also improved during the year. There is also a high score on ORT 5. It can be observed that the EEGPSR test did not improve during the year. This test was tightly coupled with GPSR, and on Stage I, it increased its score less than other tests in comparison.

The Open Challenge stayed at a 0 score all year, but this was because it was not actually scored. The scoring notes were taken in this format, so they have been left as they are. To score Open Challenge the team needed a set of judges that would evaluate the presentation, and it was deemed hard to find objective judges for this task. The practice runs were still given feedback within the team.

It is worth noting that the last ORT was held in a new environment out of the lab, which included a complete simulation of the competition with a new arena. This

may negatively affect the scoring of that session, but the team was prepared and still performed close to their expectations, providing them with confidence.

A landscape page containing Table 6.2 follows.

		Max Score	Bonus	TOTAL	ORT 1	ORT 2	ORT 3	ORT 4	ORT 5	ORT 6	ORT 7
	Robot Inspection	Pass			Fail	Pass	Pass	Pass	Pass	Pass	Pass
Stage I											
	Cocktail Party	270	37	307	0	50	60	165	65	75	25
	GPSR	250	35	285	20	1	10	0	10	31	10
	Help Me Carry	200	30	230	0	0	0	0	0	10	0
	SPR	200	30	230	80	80	135	65	185		130
	Stage 1 Total			1052	100	131	205	230	260	116	165
	%				9.5%	12.4%	19.4%	21.8%	24.7%	11.0%	15.6%
Stage II											
	EEGPSR	500	60	560	30	0	10	0	0		0
	Open Challenge	250		250	0	0	0	0	0		0
	Restaurant	285	38	323	0	0	0	40	100	45	55
	Tour Guide	390	49	439	60	0	110	80	300	40	110
	Stage 2 Total			1572	90	0	120	120	400	85	165
	%				3.8%	0.0%	7.0%	5.0%	19.0%	2.5%	7.0%
Total Points		2345	279	2624	190	131	325	350	660	201	330
Total %					7.2%	5.0%	12.4%	13.3%	25.2%	7.7%	12.6%

Table 6.2: ORT results of the year 2018. Note that some entries have either 0 or are empty. The table has been reproduced from the notes taken during the time. Having an empty score means the test was not tried, meanwhile having a 0 means it was run and it scored 0.

ORTs were set up to mimic a real competition as close as possible, but during a single day, instead of a week of competition. Practicality did not allow these to be performed out of the laboratory often. However, when possible, the team was advised to prepare their laptops and the robots to move to a new location. The ORT was announced at least two weeks prior.

A schedule was created by discussing it with key team members to ensure its viability. The schedule was shared with the team days before the event. In Figure 6.3 an example from a ORT session can be seen. It consisted of a tight schedule. Every item had a strict time slot, although it was expected that problems might arise, which could delay some slots. This ORT was expected to end by 4 PM, but team members were advised to stay available for another hour or two in case delays happened.

Operational Readiness Test - Schedule

Note: one attempt plus critique by team members for each test ALL team members to participate / watch each test		Duration	Start Time	End Time
Setup		55 minutes	09:00	09:55
Robot Inspection (3 mins)		10 minutes	10:00	10:10
	Review by Team	5 minutes	10:10	10:15
Stage I				
	Cocktail Party (5 mins)	20 minutes	10:20	10:35
	Review by Team		10:35	10:40
	General Purpose Service Robot (10 mins)	30 minutes	10:45	11:05
	Review by Team		11:05	11:15
	Help Me Carry (5 mins)	20 minutes	11:20	11:35
	Review by Team		11:35	11:40
	Speech and Person Recognition (5 mins)	20 minutes	11:45	12:00
	Review by Team		12:00	12:05
Lunch (30 mins)		30 mins	12:15	12:45
Stage II				
	Tour Guide (10 mins)	30 mins	12:50	13:10
	Review by Team		13:10	13:20
	Open Challenge (10 mins)	30 mins	13:25	13:45
	Review by Team		13:45	13:55
	Restaurant (15 mins)	40 mins	14:00	14:30
	Review by Team		14:30	14:40
	Enhanced Extended General Purpose Service F	30 mins	14:45	15:05
	Review by Team		15:05	15:15
Finals	Not to be tested yet			
Debrief (30 mins)		30 mins	15:20	15:50

Figure 6.3: Example schedule for an ORT from UTS Unleashed! in the year 2018.

The ORT started with one hour to set up. In this setup time, the network was configured; the robots were loaded with the necessary software; and mapping of the environment (arena) was performed. Some team members arrived earlier to unload the robots and to create a new arena configuration, taking care of making it different from previous designs.

From this point, tests were run in their defined time slots. A leader acted as a strict referee. This person was required to have recently reread the rulebook with particular focus on the tests to be run that day and their scoring. While keeping track of the timing and scoring, notes about errors and improvements were taken, both by the referee and the team members responsible for the test. Test runs were also recorded in video for further analysis. Other team members also needed to be attentive to be able to provide feedback and, potentially, gather ideas for their own tests.

After a test was run, a short review by the team was held. Scoring, problems, and improvements were discussed. Compliments were also given, even cheering was encouraged on success. Afterward, the people responsible for the next test prepared to run their test in their defined time slot.

Lunch was provided to the participants on a short break, which was reported as a beneficial activity in the retrospective documents. Then the testing continued until all tests were done. The day finished with a debrief session, acting as a kind of retrospective, with actions to be performed by the team recorded. The team was also required to pack up the robots and their belongings. A social activity was offered after the long day of ORT.

6.4.1.3 Task Assignment

Subteams for topics like navigation, perception, or task planning were created. Later on, subteams for RoboCup tests were created. The leader of every subteam would take care of distributing work while listening to the interests of the rest of the members. The lead developer would be updated weekly on the status of the different parts of the system.

Tasks built to increase the proactivity of interns were not successful. These tasks were mostly acted on, but the lead developer felt they were not applying themselves enough. They lacked diligence. For example, a task would include a set of checkboxes of concise things to try, test, or code. These were expected (as was the case in previous examples working together with the interns) to be used as starting points to gather useful information. Instead, these checkboxes were ticked as done by doing just exactly what the checkbox said. Further conversations with the aim to clearly communicate

what was expected from the interns when performing these tasks with checkboxes were not successful. These conversations confirmed that it was not a misunderstanding by the interns, but that they were not behaving as expected for unexpressed reasons. Possible explanations by the lead developer were: a lack of technical self-confidence, external factors in their lives affecting their work, or personality traits not aligned with this type of workflow.

6.4.2 Team Software Development Processes

The team embraced further teamwork in regards to team software development processes.

6.4.2.1 Coding Standards

This year, code quality was improved by roughly half of the team having successfully adopted Python coding standards. However, the closer to the competition, the less they were followed. Team members that did not adopt these coding standards expressed that they were focused on solving problems instead of in the shape of the code itself.

Team members used their own naming convention for packages, coming up with creative names. While these were found to be not easy to remember, they added fun to the development process.

6.4.2.2 Coding Tools

Team members supervised closely by the lead developer embraced Sublime Text with Python plugins as an IDE, as he was using it himself.

Most team members used GitLab to browse the source code of the libraries (made in house) they were using. They did not use the editors' capabilities to open the implementation of files or automatically generated documentation. The README of the packages was their first documentation resource.

6.4.2.3 Social Coding Practices

Mentoring happened often. The lead developer put attention in introducing team members to technologies and tools that may help their workflow. Some team members did the same based on the experience from the previous year of competition or because that was natural for them.

Pair programming was mostly used for bug fixing. The lead developer encouraged team members to dig as far as they could into a problem before calling him. Then, he would sit with them and explain what he did to find and fix a bug, so they learned as much as possible during the process.

Subteams formed for the RoboCup tests and the development of some parts of the system. While the subteams for parts of the system formed more or less organically, as team members asked questions about how parts of the system worked or should work. For the RoboCup tests themselves, they were set up by the lead developer, while discussing it openly with the team, taking into account the experience that the team members acquired during the year.

6.4.3 Technical Approaches

Technical approaches were marked by a conflict in regards to the architecture vision.

6.4.3.1 Software Stack

Given there was the possibility of running tests of Stage I three times and the score of the two best runs were averaged, some tests were prepared with parallel implementations that allowed for a riskier approach that could earn more points.

The team approach to prioritize GPSR and the work on the planner did not work out. Theoretically, the approach seemed promising, but in practice difference of opinion on how to approach some problems made unforeseen circumstances arise. Much effort was put into mitigating problems and satisfying the team members that would use the prepared tools. However, when changes were requested, they were initially fought against by the main implementer. An agreement would then be reached, but it would not be followed thoroughly, ending in dissatisfaction from both sides. Also, when this system was tested in ORT, it was not ready to be used. Much effort was put into doing automated testing via unit tests, but not much effort was put into executing it in the robot. Additionally, the graphical interface, which can be seen in Figure 6.4, further suffered from different opinions on its desired workflow.

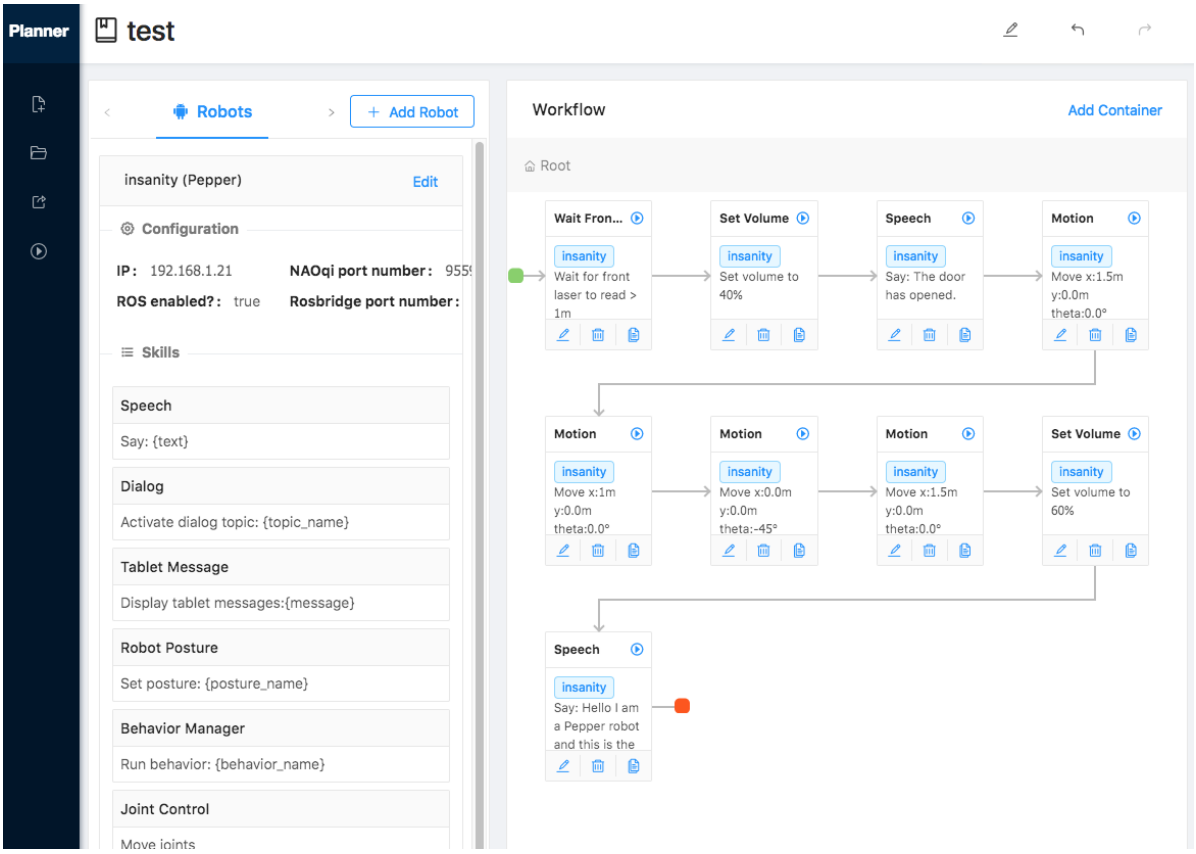


Figure 6.4: Example screenshot of the planner web interface.

This situation caused much tension. It was finally agreed, just two weeks before the competition, to give a fair test to this system to accomplish the GPSR test in an ORT. The system did not perform, and it became a crisis for the team. At this point, the team united and put together a simple version of the skills, a planner, and a task executor that did perform well just before flying to the competition.

The base system for the robot was reimplemented as a Virtual Machine that reproduced with fidelity the operating system image of the robot [8]. This virtual machine would have deployed a version of Gentoo Prefix, and over it, build via the package manager of this distribution, the necessary dependencies. Another project was brought in and collaborated online, called *ros-overlay*, to enable the build and deployment of the ROS framework. Now the system could be reproduced in any machine by deploying the Virtual Machine.

[8]The virtual machine provided by the manufacturer was a generic image for their robots and was found not to match the libraries and binaries found in our robots. This caused issues when building and deploying software.

In the first few months of development, much care was put into analyzing the behavior of the depth sensor, laser sensor, and the odometry of the robot. A special effort was made to try to improve the quality of the data of the laser sensor, but it was not fruitful. However, all this effort allowed a better understanding of the capabilities of the robot. Some work into trying to calibrate the cameras of the robot was also undertaken, but in the end, manual calibrations were used especially for the extrinsic calibration of the Red Green Blue (RGB) camera in reference to the depth camera.

A professional-grade laser rangefinder (from the brand Hokuyo) was used for experiments to initially provide ground-truth data. Later on, the possibility of creating maps with this laser for later use by the robot were also raised. This was discarded though after testing showed that it did not provide better results and decreased the performance of localization.

The ROS navigation stack was configured and tested during the year. Just prior to the competition, a final push was given to optimize its configuration. Mapping and localization used the robot laser, giving more weight to the robot odometry. The local planner was changed from the default one in ROS to the Timed Elastic Bands [104] local planner.

Different approaches were tried during the year to achieve robot localization, namely, using vision for this task. None worked more effectively than the laser version, additionally, they were computationally too expensive.

From the perception part, two significant efforts were made. These were developed in parallel and did not share infrastructure, even though they had in mind the same kind of behavior on how to access the systems.

First, people perception was done via the popular OpenPose package, which provides human skeletons from a picture. This package would run fast in a machine with a powerful GPU, and it would run slowly on the robot, which lacks a GPU. For example, the processing time of a normal detection on the external computing machine with a GPU would take around 0.2 seconds. On the robot, it could take eight seconds (while making the system unstable as it was using all available CPU). To enable usage of this package and take advantage of the WIFI in the competition arena, an architecture was built where the clients would try to use a server that was running externally if it was available. If it was not, or it failed to return something in time, it would be run internally in the robot. A set of fallback strategies were implemented for well-known cases.

Secondly, object recognition was done via a ROS package that used a TensorFlow pre-trained neural network and finetuned for the teams' needs. Pre-processing was done via

Red Green Blue + Depth (RGBD) detecting objects over planes, and only the bounding boxes of thought-to-be objects would be fed to the neural network. Extra processing was done for hard cases like differentiating between cans. This approach had the same issue as the OpenPose approach, but implemented the remote usage in a different way. However, this too raised issues regarding how well it would work.

The team made good use of the provided Alienware laptops with their powerful Central Processing Unit (CPU)s & GPUs. Docker was used to enable the deployment of deep learning setups.

6.5 Competition Participation

The development lead felt confident that they would achieve top-three ranking given the outcomes of the last ORTs but also knew that other teams could be as well prepared.

6.5.1 Team Management Processes

The processes were similar to 2017 but optimized and streamlined. The team practiced in the ORTs in a similar setup, so even the new team members had a better idea of what to expect.

6.5.1.1 Team Management Tool

Like in 2017, the team used a whiteboard with post-its to keep track of tasks, ignoring Trello. A set of columns titled To Do, Doing, and Done were used.

Every RoboCup test was recorded in video by at least one team member, if not more, to be able to review how it went, just as had been done in 2017. It was also an excellent tool to show the referees any detail that they may have missed.

6.5.1.2 Team Meetings

Meetings were held every morning at the start of the day; they were used to distribute work and responsibilities. Lunch and dinner together were encouraged, when possible, to keep track of everyone's progress, requirements, and to keep their spirits up. Every night before going home, a meeting was held to have a clear idea of what should be done the next day.

As in 2017, the team aimed to maximize the usage of the time in the venue and the arenas.

6.5.1.3 Task Assignment

Post-its were taken from the board, adding who was working on that task and updating from To Do, to Doing, and then to Done when appropriate.

The Project Managers took the role of ensuring that the robots were charging whenever they were not being used.

The team kept their setup of subteams for different tests while working in pairs whenever it would benefit the situation - which was most of the time when testing on the robots.

This year tasks that would block future tasks were prioritized and scheduled in that order, notably, mapping the arena and getting the data related to the tests (people names for the tests, object names, names of places in the arena, and others).

6.5.2 Team Software Development Processes

The goals were to minimize development and maximize testing to ensure robust and high scoring runs.

6.5.2.1 Coding Standards

A strong emphasis on not creating new work but only configuring, testing, and patching the hard work done during the year was encouraged. The team avoided last-minute changes and running code that was not completely tested.

Changes that may be considered hacks instead of fixes were committed into a different branch (usually named robocup2018) so as not to wreck the work done during the year. These branches would be re-implemented properly after the competition.

6.5.2.2 Coding Tools

A tool was developed, *magic_launcher*, allowing team members to launch RoboCup tests directly from the robot's touch-screen tablet. It consisted of a simple configuration[9] file that contained the command needed to launch it, with an automatic button generated to stop what was launched.

magic_sync was developed as a tool to ensure that the basic libraries developed by the team were in their latest version. This was to overcome issues found where team

[9]The YAML markup language was used to ensure it was simple for team members.

members tested their code against an outdated version of these basic libraries and thereby reporting already fixed bugs.

Everyone had their own sub-folder in the robot, where they would make a workspace to deploy their RoboCup tests and experiments. This ensured no one could break the workspace of anyone else.

As in 2017, a local GitLab server was ready to be used in case of a network outage.

During the year, the team members were encouraged to have prepared scripts to test elements of the system or the whole tests separately. Some tests of this type existed and were used.

6.5.2.3 Social Coding Practices

The aim was to work as a team as efficiently as possible. If anyone needed to test on a robot while also taking care of the robot placement or the environment, a second person would always be there to help. However, if someone needed to focus intensely on a task, it would be announced, and this person would not be disturbed.

If any task required an expert, the expert would be asked to do it or help whoever needed it, so as to advance work as fast as possible. Moreover, the team leader, having knowledge of all systems, was left free to help everyone and unblock teammates.

6.5.3 Technical Approaches

Technical approaches were characterized by a general satisfactory performance and a successful offline approach.

6.5.3.1 Software Stack

The tests for Stage I and Stage II were considered well developed, but a lot more time was invested in the Stage I tests as they were necessary to qualify for Stage II.

The network in this edition had huge problems. The network was slow and dropped out frequently. At some point, the network went completely down for hours, making two tests, Help Me Carry and GPSR, unable to be performed with WIFI connectivity. The team was prepared for such an event, and somehow this may have helped UTS Unleashed! to qualify for Stage II. It was stated in the rulebook that the quality or availability of the network was not guaranteed, and only two other teams were also prepared to compete in the absence of a network.

Whenever there was network connectivity, the team tried to make use of it to improve the execution time of OpenPose. OpenPose was used for object recognition and speech recognition via Google in the cloud. However, the development of this capability was left until too late and, even though it worked, it did not deliver the best results.

Our systems did not perform as well as we would have liked, but still worked well enough to get us to second position in the competition. Namely, the navigation system improved from the previous year, but using the robot laser as the primary means of localization kept being problematic. New approaches were to be explored after the competition.

Speech Recognition performed differently in different tests and times of the day. It provided us with a high score in the Tour Guide test and a good score on Speech and Person Recognition, but other tests had issues. Possible improvements became apparent after competing.

Perception did not perform satisfactorily, especially object recognition. People perception did not have the opportunity to show its capabilities as much as the developers would have liked. The team's OpenPose network, running locally, performed well and provided the robot with vital information for the robot's decision-making process.

Other systems worked well, for example, deployment of the robot software, starting the RoboCup tests from the tablet, and the robot being totally offline. Also, our capability to quickly hack something together was even better than the previous year.

Some tests were engaged in a trade-off between robustness and the possibility of scoring higher. These tests had multiple implementations ready to be used with the one to be used decided just before the tests (after evaluating how they performed in the event).

Some tests were ready to record rosbags of meaningful data for analysis later on. However, it was not generalized.

6.5.4 Results: Competition Outcomes

UTS Unleashed! performed satisfactorily in Stage I, as can be seen in the scoring sheet in Figure 6.5. The team was in second place at that point. The team did not score 0 in any test, which observing there were twelve 0 scores, and comparing with the previous year where UTS Unleashed! did score 0 in one test (Help Me Carry), the team had obviously improved. Only ToBI@Pepper performed better in all tests (with one matching score). The test where most teams scored was Speech & Person Recognition. This test did not feature any navigation and was fully based on perception and speech recognition. Once

navigation is introduced in the tests, the scores start dropping. Our lowest scores aligned with being the lowest scores for everyone.

Even though our GPSR implementations were only improvised in the last weeks before the competition, it earned us the second-highest score by accomplishing partial scoring in one command in two runs. This test is of high complexity, and only three teams scored. The team was happy with the results.

Some teams were totally dependent on the network and were not able to participate in the Help Me Carry and GPSR tests due to the network interruptions. UTS Unleashed! was prepared to run 100% offline as it was known that network difficulties are not unusual (and the experience in 2017 where the network performed exceptionally well was an outlier). The Restaurant test also had no connectivity, but in this case, it was by design as the test happened in a real restaurant out of the RoboCup venue.

SSPL Stage 1

	Robot Inspection	Poster	Speech & Person	Cocktail Party	Help Me Carry	GPSR	Stage 1	Rank
ToBI@Pepper		39.17	123	33	38	73	306.17	1
UTS		35.00	78	33	10	27	183.00	2
UChile		32.50	90	20	-	0	142.50	3
LyonTech		34.17	48	30	-	0	112.17	4
AUPAIR		30.00	48	0	15	13	106.00	5
SPQReL		40.83	40	8	0	0	88.83	6
Gentlebots		37.50	43	0	0	0	80.50	7
CMPepperBot		36.67	0	0	0	0	36.67	8
LIU		27.50	-	5	-	-	32.50	9
Maximum		10	200	270	200	250	930	

Figure 6.5: 2018 scoring sheet of RoboCup@Home SSPL Stage I. The team was second at that point.

UTS Unleashed! made it into Stage II as expected. In Figure 6.6 we can see that our Open Challenge test-presentation was regarded as the second-best one after ToBI@Pepper. The scoring is based on judges giving a score to the presentation instead of actual tasks with partial scoring. Our Tour Guide test performed well, achieving the highest score in Stage II and being close to the test's maximum score. Both Restaurant and EEGPSR tests did not go that well, but the team did not score 0, which other teams did. This performance got us to second place at this point, far ahead of the team coming third, and not that far behind the team coming first (ToBI@Pepper).

SSPL Stage 2

	Stage 1	Open Challenge	Tour guide	Restaurant	EE-GPSR	Stage 2	Rank
ToBI@Pepper	306.17	250	50	60	150	816.17	1
UTS	183.00	178	310	10	10	690.93	2
AUPAIR	106.00	142	45	65	0	357.57	3
UChile	142.50	150	50	0	-	342.20	4
LyonTech	112.17	98	0	-	0	210.54	5
Maximum	930	250	390	285	250	2105	

Figure 6.6: 2018 scoring sheet of RoboCup@Home SSPL Stage II. The team was second at that point.

UTS Unleashed! qualified for the finals. The finals were not prepared before the competition, so a demo unifying the best skills was created at the competition itself. Scoring of the finals is similar to the Open Challenge but with a different set of judges. The scoring itself was not made public in this edition of the competition.

Objectively, UTS Unleashed! and ToBI@Pepper were not far apart on scoring. The team did very well, and maybe luck played a role in having the judges score both the Open Challenge and the Finals in favor of ToBI@Pepper. It must be noted that the ToBI team has been participating in RoboCup@Home since 2009, making them one of the most experienced teams. Getting a score that was similar to theirs can be considered a great outcome.

SSPL Finals

	Rank
ToBI@Pepper	1
UTS	2
UChile	3
AUPAIR	4
LyonTech	5
SPQReL	6
Gentlebots	7
CMPepperBot	8
LIU	9

Figure 6.7: 2018 final classification sheet of RoboCup@Home SSPL.

As can be seen in Figure 6.7 UTS Unleashed! finished in second place again this year.

6.6 Post-Competition Data Collection and Retrospectives

Statistical data was collected from the Trello boards and the Git repositories as in the previous year. A retrospective document was also collected from the feedback provided by the team members; these are discussed in this section.

6.6.1 Statistical Data

Every year the available data from Trello and Git was analyzed as explained in chapter 4.

6.6.1.1 Trello Cards Data

This year Trello was used heavily at the start for the initial planning and research for approximately a month. Then its usage decreased as the competition got closer just like the previous year. The work during the weekdays was more steady than the previous year, where the team meeting day dictated the most activity in Trello. Most activity was shown to be in the development of robot skills.

The Trello activity over the year of development is shown in Figure 6.8. Mid-November the activity kickstarts with a peak at the start of December, where the planning for the year and the architecture view was created. We observe continuous activity until May, when Trello was not used that much until a spike, which coincides with the RoboCup event dates on 16-22 of June. Task management during the period between May and the competition start was handled via a whiteboard with post-its instead of Trello.

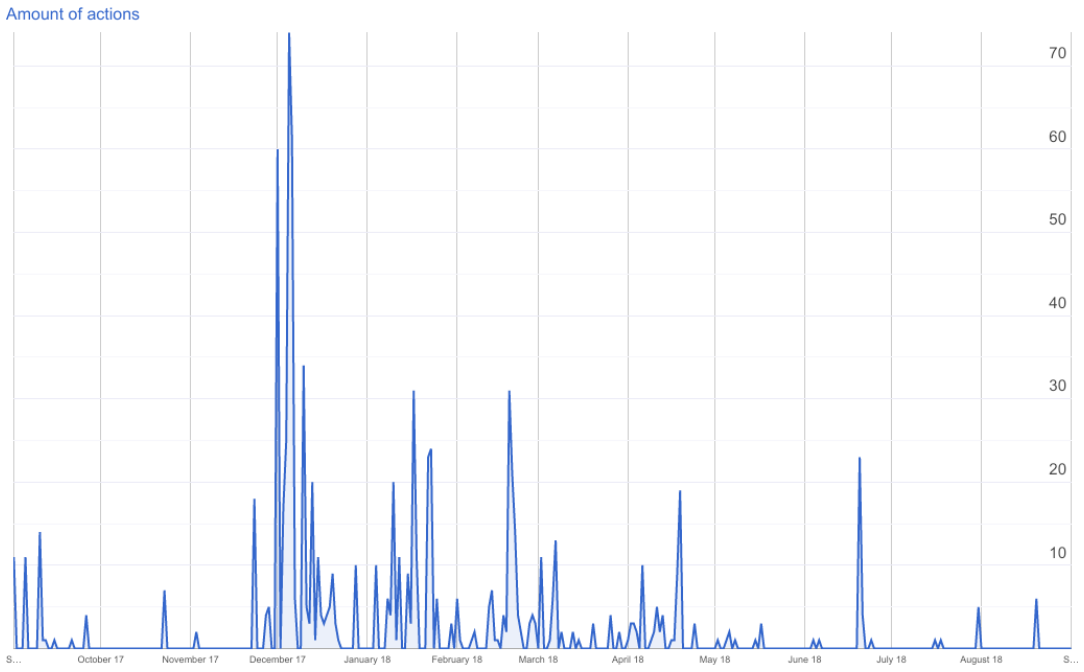


Figure 6.8: Activity (Trello actions) on the year 2018.

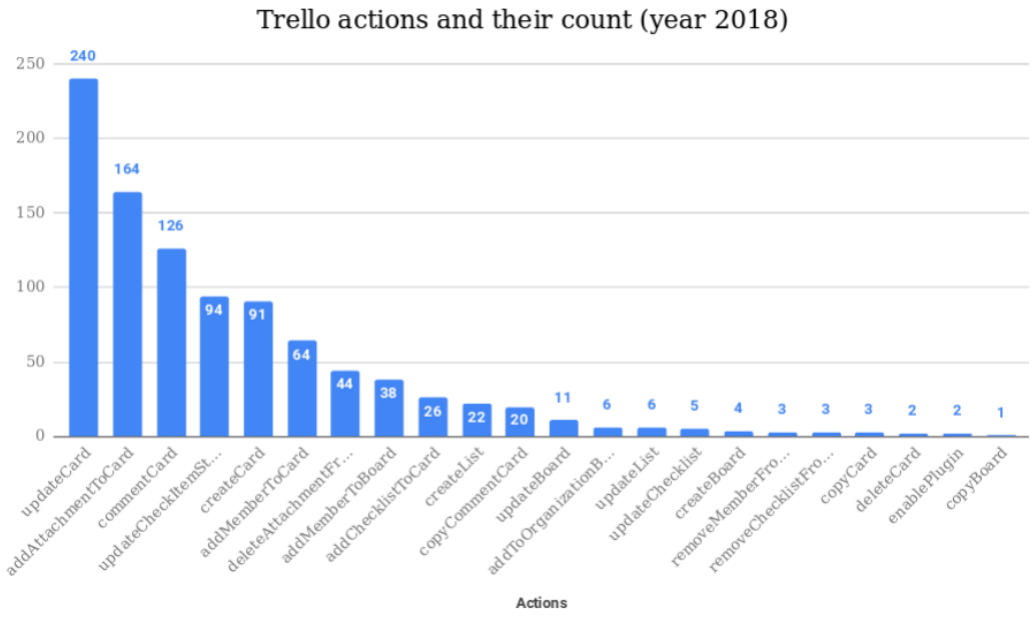


Figure 6.9: Distribution of Trello actions types on the year 2018.

This year the actions we find in Trello contain the usage of new features. In Figure 6.9 we observe how, like in 2017, updating cards was the most used action. However, this

year adding attachments to cards followed, with as much activity as commenting cards. Additionally, the team members attached many screenshots and plots of their work. Furthermore, checklists appear to be used this year to track subtasks in a card.

Looking at the month with most actions in Figure 6.10 December stands out on top of any other month. This coincides with the planning for the year and the initial research done in different fields. From January, Trello usage lowers steadily, similar to what happened in 2017.

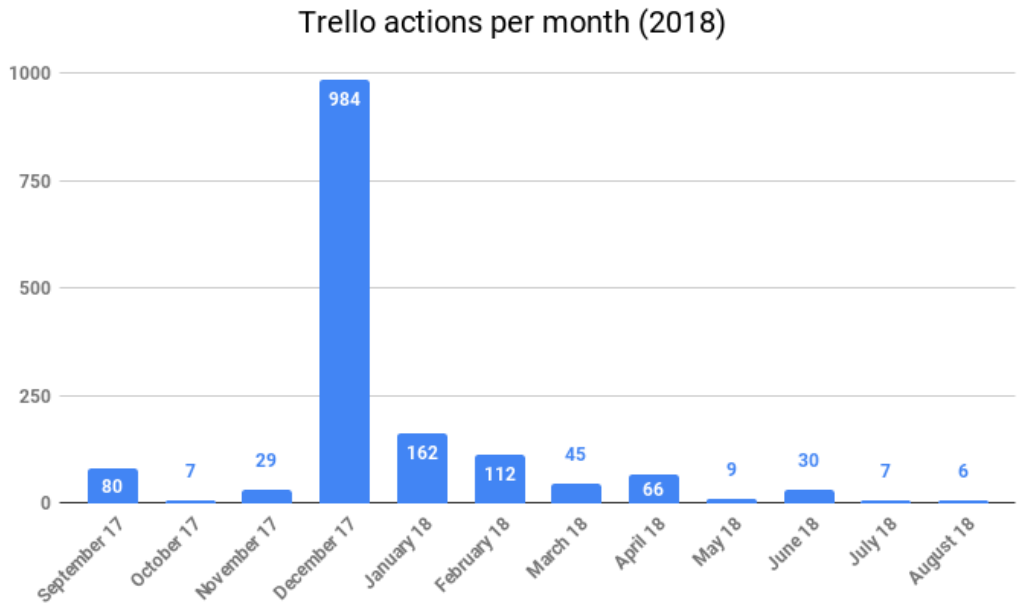


Figure 6.10: Distribution of Trello actions by months on the year 2018.

Checking which day of the week had more Trello activity Figure 6.11, we observe that both Tuesday and Wednesday had the same amount of actions. Team meetings happened on Tuesdays as in 2017, but it cannot be observed in this chart. A similar amount of activity was observed during the rest of the working days of the week. Sundays had some activity too!

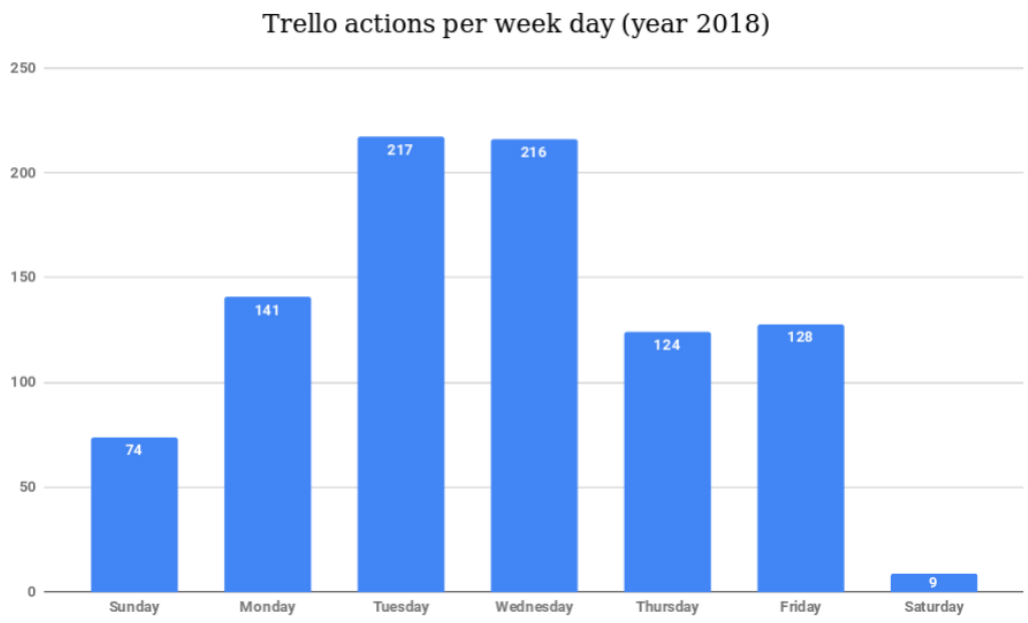


Figure 6.11: Distribution of Trello actions by weekdays on the year 2018.

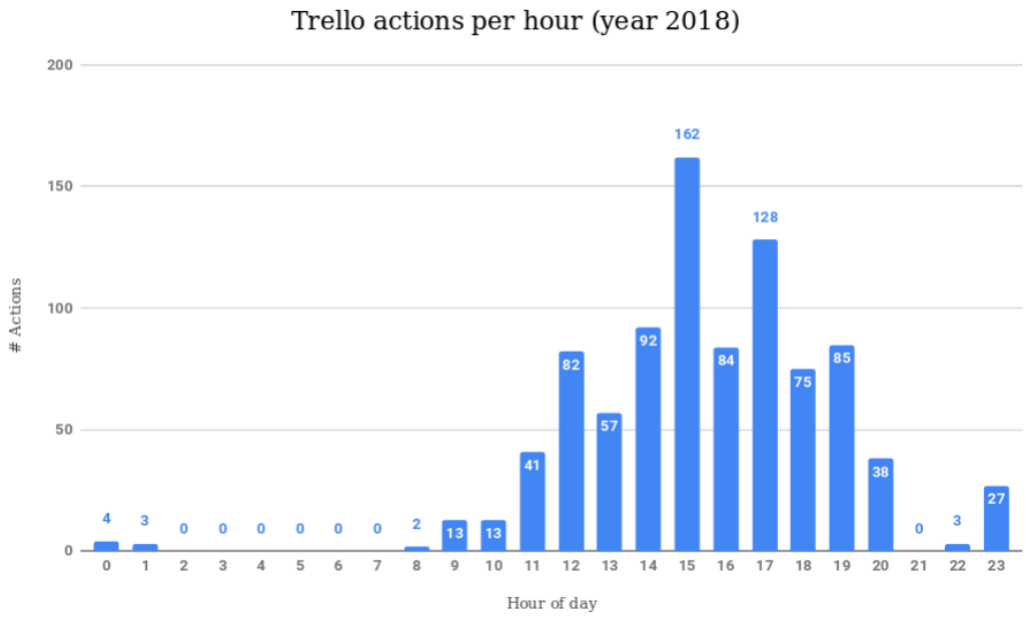


Figure 6.12: Distribution of Trello actions by hours on the year 2018.

The distribution of activity during the day was similar to 2017, as seen in Figure 6.12. The activity started at 9 AM up until 8 PM (instead of 10 AM to 6 PM in 2017). The most

prominent peak of activity was seen at 3 PM and 5 PM, which could be related to the team meetings ending around those times. An isolated peak of activity at 11 PM showed some late work, as in 2017, but less.

This year's Trello boards had a different distribution of activity than 2017. In Figure 6.13 we observed that the *Skills* board had the most activity. This board contained the development of the robot's abilities. It was decided from experience in 2017 to put much effort into having robust robot skills, and the Trello data supported that. The *Software Engineering* board followed, which was tightly related to the development of the skills. Subsequently, a board called *Challenges* contained the RoboCup Tests as "challenges". When the team started to work on them, other means of task tracking were used, namely, the post-its whiteboard.

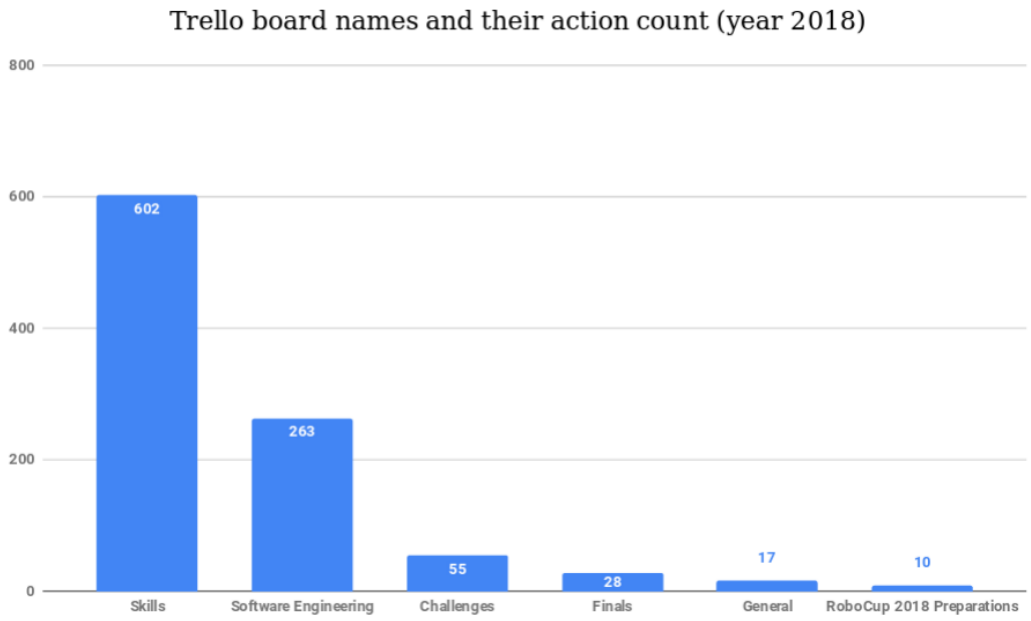


Figure 6.13: Distribution of Trello actions by Trello board on the year 2018.

Finally, from the anonymized data from authors and their action count in Figure 6.14, one team member had almost double the activity of anyone else. After carefully checking the original data, the conclusion was that this team member seemed to make a lot more use of the platform than anyone else. The following four team members had a similar amount of activity. Then the rest had a low amount of activity, participating in Trello only when requested.

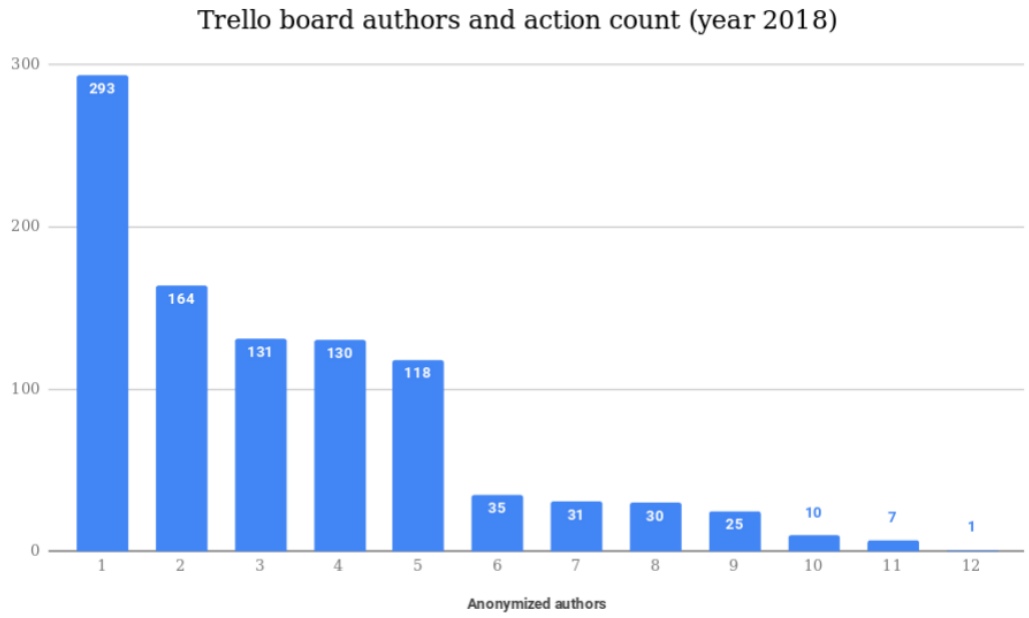


Figure 6.14: Distribution of Trello actions by anonymized authors on the year 2018.

6.6.1.2 Git Commit Data

The Git data showed, as in 2017, steady activity increasing as the competition got closer, with as much activity during the competition, as in the previous weeks leading up to it. We interpreted this to infer that teamwork increased from 2017.

The 2018 commit activity per day chart found in Figure 6.15 showed that work started or continued early on from September. The activity started increasing from the end of January until the competition at the end of June. We observed an increase in the number of commits in the last weeks before the RoboCup event.

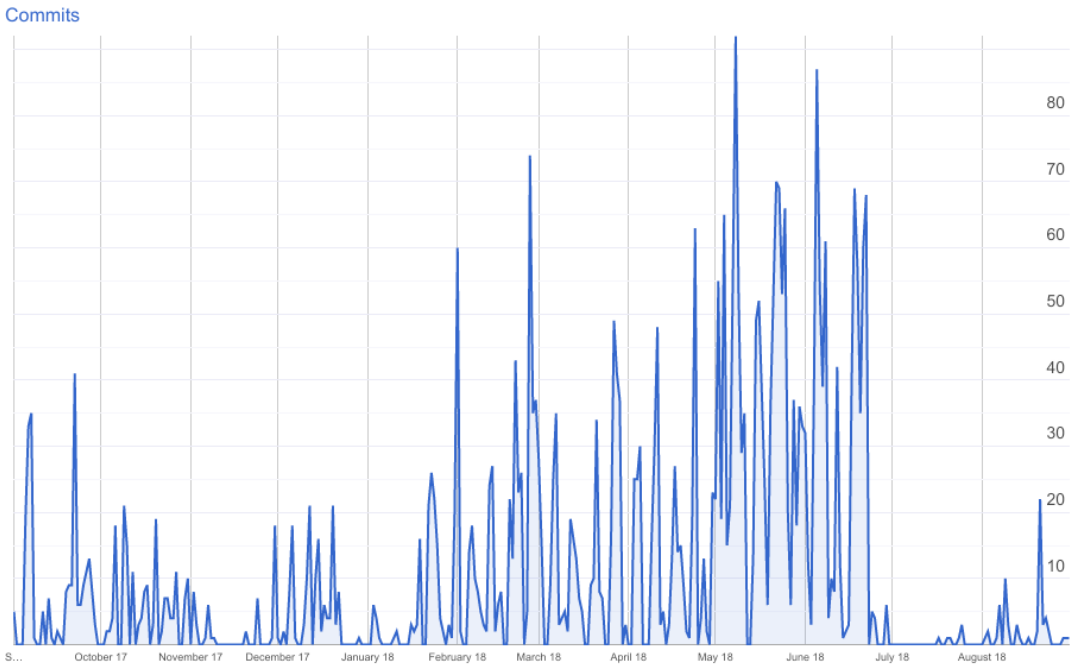


Figure 6.15: Activity (commits per day) on the year 2018.

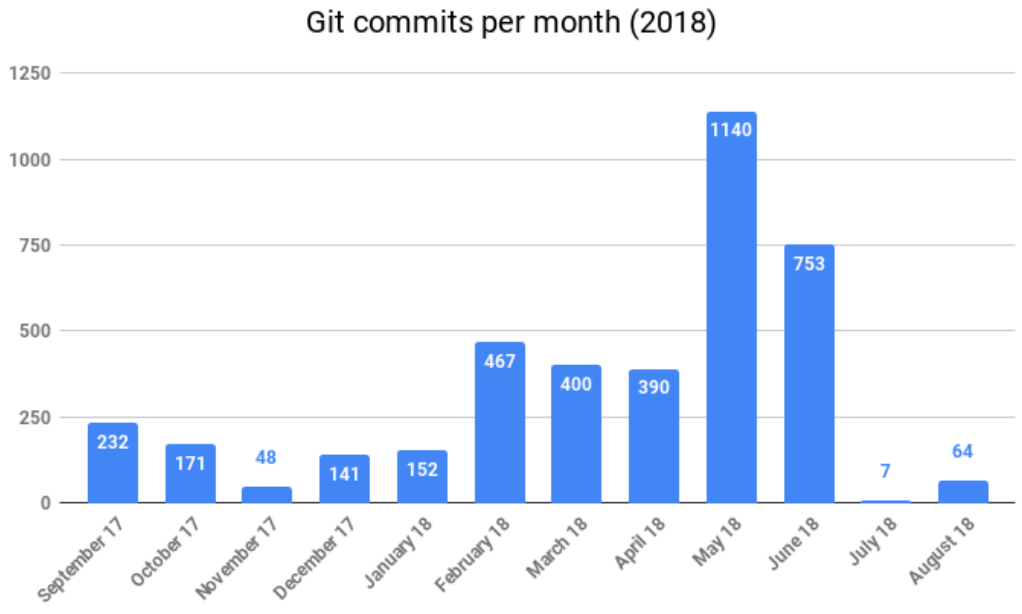


Figure 6.16: Commits per month on the year 2018.

Moreover, looking month by month in Figure 6.16, May contains the most activity followed by June. As in 2017, there was a trend of increased workload as the competition

approaches.

When checking which days of the week had more commits in Figure 6.17, Tuesdays were identified as the ones with the most activity. This could have been related to team meetings happening on Tuesdays; however, all the working weekdays have a similar amount of commits. Furthermore, some activity is shown on the weekend, as in Trello, and as in 2017.

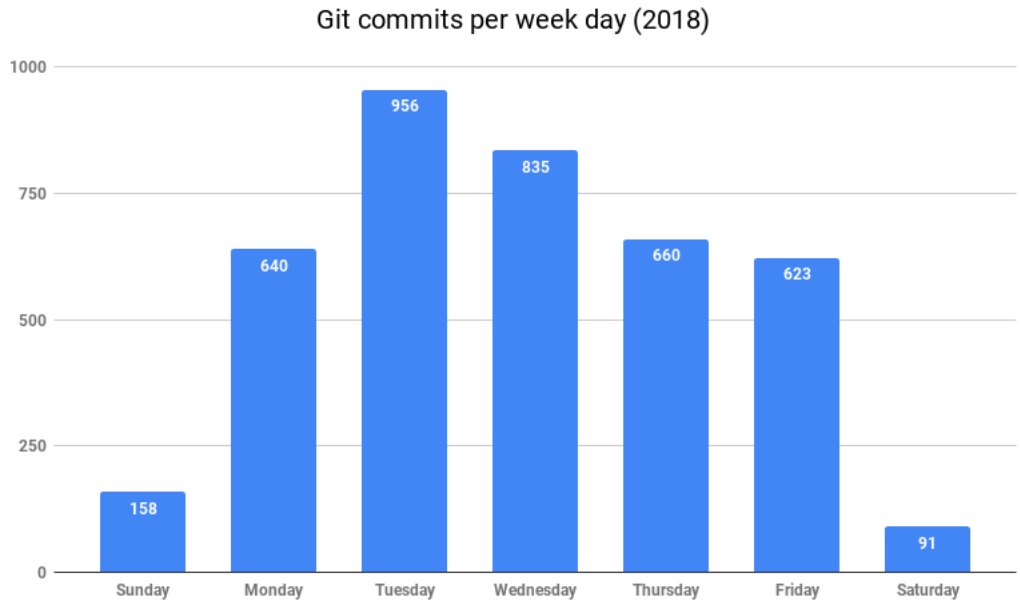


Figure 6.17: Commits per week day on the year 2018.

In the case of commits per hour of the day, as seen in Figure 6.18, work started at around 10 AM and peaked at 5 PM to fall by 10 PM. The plot looked similar to 2017 but with an increased number of commits.

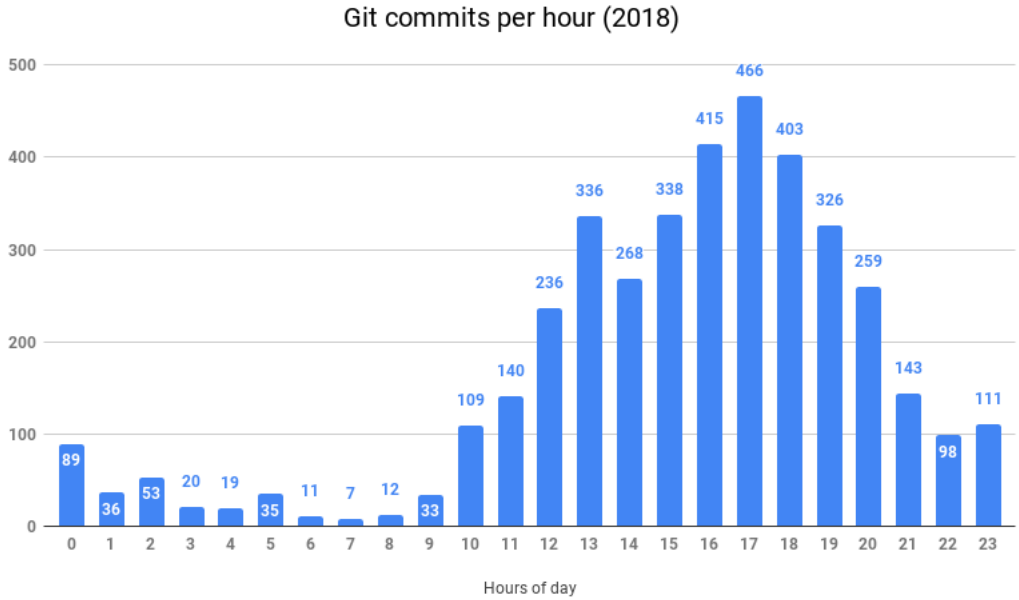


Figure 6.18: Commits per hour on the year 2018.

Analyzing this year's teamwork based on git activity by looking at Figure 6.19, we first saw a much larger amount of repositories with two or more authors and with more than one day of commit activity. This year 41 repositories made it into the plot against 11 from the previous year. This aligns with the fact that this year multiple repositories were used instead of a centralized repository using submodules (as it presented problems the previous year). The data presented brought up the following insights:

- The repository with most activity (by far) and most authors (mostly every team member developing code) was *pepper_skills*. As the name states, this repository contained the skills the robot could perform, and it was planned to have an important focus on those in this year's development. *stuffed_pepper* and *qimate* contained easier-to-use APIs to access the robot APIs frequently used in *pepper_skills* and also showed a high number of collaborators. Other repositories contained implementations for skills like *pose-detector, pepper_object_recognizer, magic_speech, pepper_tabletop, ros_object_recognition_docker, pepperception*.

- The following three repositories with the most authors were: *docs.wiki* which, as in 2017, contained the wiki with the team's documentation; *pepper_navigation* which contained most code for the robot to navigate. Almost every test needed this skill;

and *finals2018*, where most of the team worked together to implement a test for the finals during the RoboCup event, also showed great teamwork.

- The set of repositories: *uts_planning, gpsr, planner, and task_manager* were part of the effort of making every test into a GPSR test variant by using the planner interface that could be configured with a graphical web interface. Much work could be observed in them; however, this work was not finally used in the competition itself.

- Most other projects showing collaboration and activity were implementations of RoboCup tests (*help_me_carry, cocktail_party_skillsbased, tour_guide, restaurant_test, open_challenge, robot_inspection*) or experiments.

- The magic_tablet repository showcased how the team gave much importance to the Human Robot Interaction (HRI) experience as in 2017.

- The repositories *pepper_docker, on_boot_goodies* and *magic_launcher* contained code to build and run the dependencies of the team's system. A second author appeared in these repositories, but it was still, like in 2017, mostly a one expert person job.

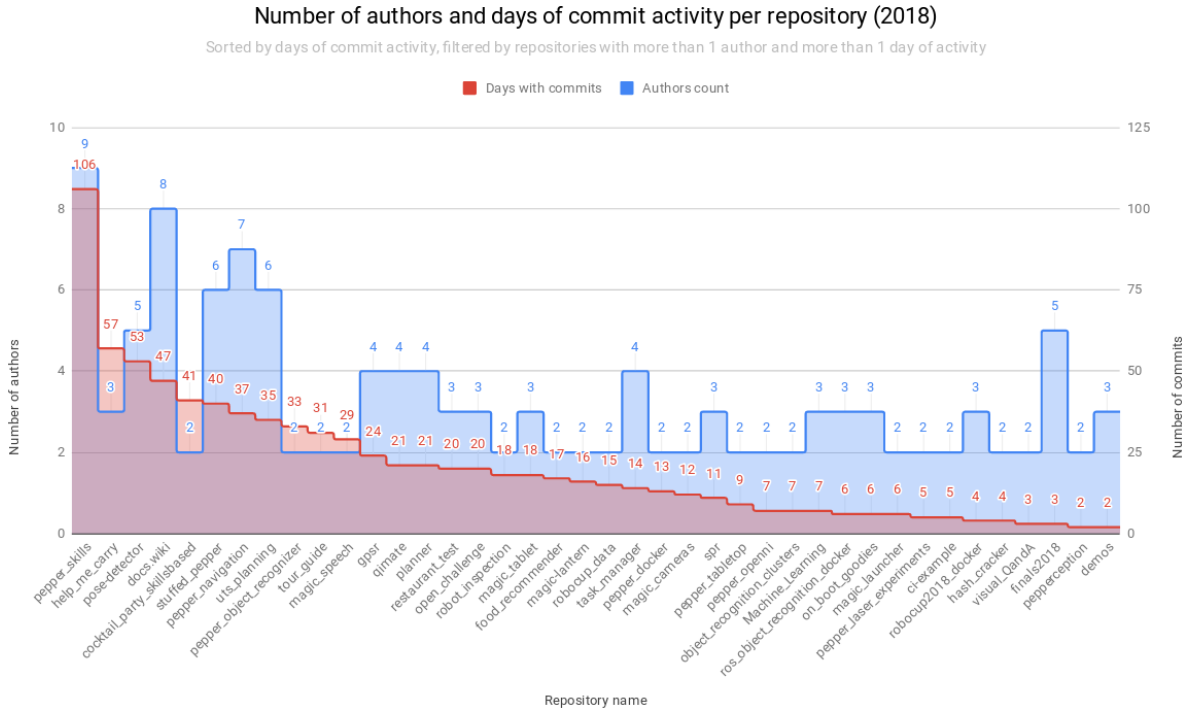


Figure 6.19: Number of authors and days with commit activity per repository, sorted by days with commit activity and filtered by more than 1 author and more than 1 activity day.

When checking repositories with only one author, found in Figure 6.20, a similar amount of repositories (20 in 2017, 18 in 2018) as in 2017 was found. The insights found from this plot were:

- A similar amount of single-author repositories could be a good sign of teamwork as many more repositories represented collaboration.
- Many experimental repositories were found in this plot (*localisation-with-machine-learning, rosduct, robocup_tests, cocktail_party_2018, tour_guide_experiment*).
- Projects with a high level of expertise needed to engage in their development were also seen (*plan-blocks, magic_sync, magic_c_helpers*).
- *ros_pepperfix* contained a re-implementation of the base system, coming from *pepper_docker*. Hence, this was mostly a one-person job.

- The rest could be considered side-projects parallel to the RoboCup development, which while not directly related to the competition, built up experience using the team's codebase.

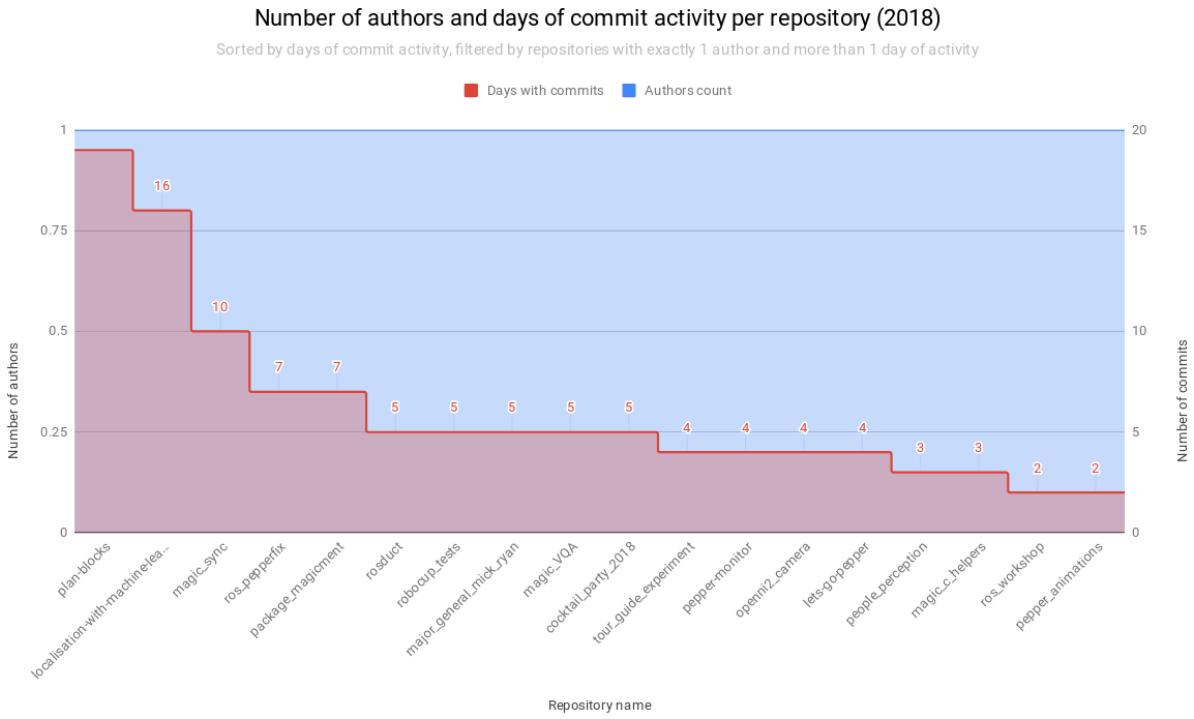


Figure 6.20: Number of authors and days with commit activity per repository, sorted by days with commit activity and filtered by exactly 1 author and more than 1 activity day.

The average of authors in repositories with more than one author and more than one day of commit activity came up to 2.55 authors per collaborative repository. This was higher than in 2017, which had an average of 2.3 authors per collaborative repository. Considering that the number of solo projects stayed roughly the same as in 2017, we interpret this as an increase in teamwork this year.

6.6.2 Team Retrospective

In this section the team's retrospective will be summarized. The original retrospective document for the year 2018 can be found in the section B.2 from the Appendix B.

The first question of the retrospective was "What worked well?" about the positive facts from the preparation process and the competition itself. Regarding the competition

event, it was reported that the on-site technology resources were well prepared with an emphasis on the offline-first capability of the team and the robot. The HRI, speech recognition, and question and answer system were regarded as well-performing skills. Also considered positive, were effective teamwork, hospitality, and sportsmanship. Concerning the development process, the team described positively: the coding practices of source code control, package management, and continuous integration; engaging in real-world testing and holding well structured, frequent and complete ORTs where "rigorous testing paid off"; and building up from modular and simple proof of concepts and iterating into refined versions, and having access to powerful computers as also contributing positively to the outcome. Furthermore, as positive notes in relation to teamwork and team management, the team praised itself on having all day and everyday activity in the lab; having a larger team; well distributed work using Trello during the first months of development, practicing sprints and feature delegation. By the last month, a particular focus on keeping close track of progress with a whiteboard including scoring progression highlighted efficient teamwork. Moreover, the team reported shared responsibility, team cohesion, collaborative decision making, the successful pairing of teammates, and enthusiastic and talented interns.

The next question was "What did not work well?" about the adverse facts in the same context. Many reports were conflicting with the previous question. Regarding issues in the competition event, the team reported problems with the HRI implementation, the navigation system having problems stemming from the localization system failing, and the development of the finals being done in a rush during the competition. Moreover, the majority of negative feedback was related to the development process, including a too rigid software design with unit testing preventing quick coding; lacking a streamlined setup for virtualized environments; and problems with testing of RoboCup tests as the components lacked isolation and only one person had the knowledge to evaluate the performance of the components separately. Interestingly, a conflict with the planner language and the planner system itself appears as it is both reported as not working as expected and as not used. Furthermore, during the development process, the team management had issues regarding communication. Specifically, it was reported that some team members needed help and reported it too late. Meanwhile, other team members did not know how to help or be involved, and, additionally, there was a lack of understanding and agreement between some team members regarding some technical decisions. Possibly stemming from the last point, a lack of visibility and clarity about the task backlog was reported. Finally, insufficient training was described which created a lack of shared

knowledge. This made the team's strategy to share code to be considered too complicated. It also raised issues with development being slowed down by team members not being able to use ROS effectively for tasks involving sensory data and 2D and 3D information.

The last question was "What should we do next?" about proposals on how to improve the next year of development and competition performance. The team identified specific improvements that needed to be researched and implemented for the following robot skills: navigation (detailed as localization, mapping, person following navigation, and safe navigation), object recognition, speech recognition and generation, manipulation, human perception, and HRI via the tablet interface. These improvements were expressed with the spirit of researching new approaches and making the best use of HRI approaches to overcome challenging scenarios and errors. Moreover, the team reported approaches towards optimizing code, creating simpler Application Programming Interfaces (APIs) with minimal dependencies, and wrapping ROS features to reduce its perceived complexity.

6.7 Reflection

The reflection is composed of the findings, which include the same substructure of three sections as explained in chapter 4, answers to the action research questions stated in this cycle, new questions to start the next cycle, and next steps to be implemented.

6.7.1 Findings

The findings stem from a review of the action research cycle and the team's retrospective. They aim to describe facts learned in this cycle.

6.7.1.1 Team Management Processes

In the retrospective document, the team showcased that teamwork was successfully built as they reported shared responsibility, team cohesion, collaborative decision making, the successful pairing of teammates, and enthusiastic and talented interns. These optimistic revelations were interpreted as caused, or at least influenced, by a set of facts: having a larger team compared to the previous year, as each member could tackle a more manageable amount of tasks; to the team working together "every day and all day" in the laboratory boosting interpersonal relations and knowledge sharing; widespread Trello usage from December to February providing visibility on what and how tasks were

worked on; structured, frequent and complete ORTs allowing the team to work together and understand it's progress; and close mentoring and supervision of the interns.

On the other hand, conflicts in the team regarding architectural decisions and their implementation arose. These problems and the difficulties associated with handling the situation damaged the team's morale, decreased the team's confidence in the proposed system architecture, and soured interpersonal relations between specific individuals. The project managers and the lead developer did act on multiple occasions using multiple strategies aiming to improve the situation, but the issue dragged on until just weeks before the competition. At that point, the team decided to abandon the previous approach and implement a simpler and more ad-hoc solution. This resolution showcased that the team could quickly work together to find a solution for a complex situation. Thus, strategies to prevent and alleviate these kinds of situations must be researched. Furthermore, it is meaningful to keep in mind the power making team-wide decisions.

Additionally, practices like sprints from Scrum[10], or feature delegation from Waterfall[11], and having product owners[12] which stem from taking reference from Software Development Methodologies (SDMs) had mixed results. The lead developer interpreted these mixed results as a consequence of some team members having a more compatible profile with those practices, and/or a lack of experience on how to effectively implement these techniques.

Finally, it could be interpreted that the team practiced a lightweight version of sprint cycles as tasks would be created as required, then prioritized taking into account time estimates. Team meetings would provide updates to the team as a whole on the state of these tasks, casual meetings in between team members would happen during working days providing closer communication, testing and integration of components would happen continuously, and, finally, team meetings would also have demonstrations and a retrospective element, as the team was open to talk about the processes that were used at any time. In conclusion, the process was less rigidly structured than Scrum's sprints, but the agile philosophy behind it existed.

[10] As described in subsubsection 2.4.2.1.

[11] Feature delegation was discussed in the team as a process where each task would have a clear description including the requirements, purpose, necessary resources, potential challenges, and a deadline and then a leader would assign this task to whomever they think is the aptest based on the context.

[12] Understanding a product owner as someone that represents a specific feature to be developed and takes the responsibility of its vision.

6.7.1.2 Team Software Development Processes

A clear direction on how to create APIs for the robot was not found. Different team members had different preferences on how to manage their code. This year's architectural approach tried to be flexible and allow the usage of common robot skills from different interfaces, but conflicts with the implementation hindered progress and adoption.

On the other hand, building modular and simple proof of concept or minimum viable products and iterating them, increasing their usefulness and robustness, was a successful approach.

6.7.1.3 Technical Approaches

Programming by configuration, and doing that via graphical interfaces, seemed like an auspicious way to ensure that the team had well-tested skills to orchestrate for the competition. However, this did not manifest in a successful implementation this year due to conflicts that arose during development. The underlying abstraction aimed to allow multiple interfaces and styles of usage of the same robot skills; however, it became the main topic of controversy. Finally, as a technical approach, it may still be valid, as demonstrated by other teams that used this strategy, but for UTS Unleashed! this approach lost support.

On the other hand, the offline-first approach used, where all the system could run without internet access, or even WIFI, by running all software in the robot's onboard computer was successful. It provided the team the opportunity to compete in all tests, even at times when there were network failures. Moreover, some resources took advantage of an external computing device running a GPU allowing the usage of deep neural networks for perception tasks. These resources did not perform as well as expected though, making this a pending topic for RoboCup's next edition.

Moreover, the work done in regards to HRI in the RoboCup tests, especially the tablet interface, provided excellent results by allowing operators and developers to understand better what the robot was doing by having immediate, visible and straightforward feedback. Dialogues were carefully crafted with simple feedback on the tablet, showing what the dialogue was about and simple animations allowed users to understand what task the robot was performing. A style guide was created to encourage best practices for social robots and to maintain a personality for the team's robot. However, it was often violated, as every test reimplemented the HRI.

Furthermore, the speech recognition system performed satisfactorily, with the ques-

tion and answering system, highlighted in the Tour Guide test, performing exceptionally well. The team identified that multiple sessions of real-world testing of the system allowed the gathering of considerable amounts of data and experience, to improve its performance.

6.7.2 Answers to AR Cycle Questions

After a review of the cycle and the team's retrospective, the proposed action research questions were revisited and answered following the same structure like in the previous chapter.

6.7.2.1 Team Management

The cycle started with the team focused on recruiting new team members by creating a workshop presenting activities similar to RoboCup tests. This approach was successful for UTS Unleashed! and it would be worth further researching as a university subject instead of a one time workshop.

Subsequently, a spreadsheet, as seen in Figure 6.1, to prioritize task implementation was created by the lead developer in collaboration with some key team members. It contained the skills thought to be needed for all RoboCup tests. These were gathered by carefully reading the rulebook and asking the team members from the last year for their input. Every skill had a proposed signature for the function calls and minimal documentation on its expected behavior. This allowed the team to prioritize these skills by how common and critical they were for the competition as a whole, which allowed for better planning for the year. The aim of this document was to be revised during the year as an additional tool for task prioritization. However, it was only used at the initial planning stage as a brainstorming tool.

Once the team was completed, and tasks were created, these were distributed to the team members. Each person was asked about their interests, experience, and goals for their participation in the project and initial tasks were provided. In most cases these were proposed by the lead developer as, especially in the case of new team members, they did not have a strong inclination towards any topic in particular. As team members manifested ease or hardship working on their assigned tasks, the tasks were adapted to fit their context. During this process, a clear distinction between team members that had a proactive attitude and others that did not arose. The management style that came out for each of those profiles was different. Non-proactive team members needed closer

guidance and tracking, increasing the management workload. In the future, research about proactivity will be done to understand better how to increase proactivity.

As the team grew and there was a considerable amount of time to prepare for the competition, popular SDMs were researched to improve the development process. However, in the opinion of the lead developer, no SDM seemed to align with the team's preferences, as standard SDMs are designed with a business model in mind, with professional paid employees under a contract, a customer, and other factors that differ from the setup and goals of the RoboCup@Home SSPL project. Nevertheless, the Agile manifesto[13] presents ideas that line up with the project's needs and the lead developer's vision, so these ideas and practices from SDMs were trialed and adapted for the team's context.

As the retrospective document and the git statistics showcased, this year's teamwork was effective. Knowledge transfer was achieved due to the team working together in the lab, often doing pair programming, or just working in pairs. Furthermore, tutorials and workshops were held to introduce major developments or tools. Finally, team meetings included demonstrations and brief explanations of how systems worked, while they also provided opportunities for team members to ask questions and ask for help.

This year ORTs were run as soon as possible when the team was able to work together and have a minimal software stack to run RoboCup tests from the rulebook of the previous year. These ORTs were adapted to the current context, and if systems were not mature enough to allow for direct scoring, other means of evaluation were created. For example, it was acceptable for tests to bypass the elements that were not available yet. New team members acquired a better understanding of what the competition looked like by participating in these events. This allowed them to better understand the team's strategy and decisions designed to improve their focus when developing.

Trello was used extensively during December to February as most task creation and research tasks were to be done in those months. The backlog of tasks was more effective at showcasing the progress of tasks, as the team added images to cards and used checklists with specific goals to achieve in a task. These simple additions provided more visibility and structure than previously, where only text descriptions were used. During the last six weeks of development, when the workload became more intense, a whiteboard with post-its was used to keep track of tasks, providing an easily accessible and understandable view of the status of the project.

[13] As introduced in chapter 2

6.7.2.2 Coding Practices and Technical Approaches

Regarding coding practices, the lead developer wanted to make the development of RoboCup tests easier and more robust, by following coding standards to improve code quality, while conforming to the team's coding preferences. Using a graphical interface to configure tests instead of coding them by traditional means was attempted, unsuccessfully. As the backbone of this work, tools that increased accessibility to complex robot behaviors in a unified abstraction layer were developed, but they were not widely adopted. Factors that affected this outcome were: issues regarding the APIs not adapting to everyone's preferences; a lack of agreement in how to implement additional interfaces; and perceived excessive rigorousness on documentation and automated testing. In hindsight, as a lesson for future implementations, the system could have been introduced in a more mature state with carefully crafted examples, templates, and simple processes that showcased how to solve complex problems that the team could relate to, thus, improving adoption and preventing conflicts.

Most code was developed from scratch this year as previous code was considered messy and lacking an architectural vision; however, successful concepts were kept, reimplemented, and improved. The lead developer encouraged ROS usage by running a workshop and working in pairs with some team members, showcasing its usefulness and ease of use for specific tasks. Subsequently, some tools as Rviz, a 3D visualization tool for ROS applications, were used while others were ignored and even reimplemented, for example, rosbags, a tool to record and playback ROS data. A few team members perceived ROS as being too complex. Further investigation on this reasoning revealed that a lack of experience working with libraries and general networking knowledge was the cause. Additional documentation was written to improve the situation, including a ten-step debugging guide for common issues related to the user's shell environment. The idea of exploring easing the learning curve for the next year was noted.

6.7.3 New Questions

This cycle raised the following research questions, based on the following observations:

- How to architect the system?

  An architecture was proposed and attempted, but it failed for diverse reasons. As seen during the cycle, further efforts were needed for the next cycle.

- How to transfer knowledge?

Knowledge transfer improved compared to the previous cycle, but the lead developer believed that there was room for improvement. The failed architecture aimed to help on this approach, and as part of a new architecture the concept must be revisited.

- What is the best shape for a robot API?

  The architecture for this year aimed to allow for a variety of styles of usage and implementation of robot APIs. These were not as successful as the lead developer wished, and this this question remains open.

- Can we improve further ORTs?

  This year the ORTs were perceived as preparing the team successfully for the experience at the competition, however, as ORTs were identified as a key element for the team success, it is worth iterating on them further.

- How to structure code for improved sharing and reusability?

  Given this year's architecture did not provide a clear improvement in this topic, it is worth further exploring in the next cycle.

- How can we distribute the work during the year so we do not have much more workload close to the competition dates?

  It was believed that as this year had a more extended period for development, compared to the previous cycle, the workload could be spread evenly during the year, but it turned out to not be like that. To allow for a lower workload in the last weeks before the competition, it is worth further thinking about it in the next cycle to improve the team members' experience.

- Can we further increase and/or improve teamwork?

  Teamwork increased this year as seen in the Git data and the retrospective document, and the lead developer believed that it was a key factor for the team's success. Given that, it is worth investing effort in the next cycle to further improve.

- How to find new team members that are proactive?

  The lead developer believed that proactive team members could be beneficial for the team and aimed to encourage this behavior with the interns but failed to do so. As it could be a personal trait looking to recruit team members that are already proactive, it was to be taken into account for the next cycle.

- How to increase proactivity in the team members?

  An attempt was made to increase proactivity in the interns, but it was not successful. Further research in this direction may benefit the team.

6.7.4 Next Steps

This section describes the aims for the next action research cycle, based on the review of this cycle and the team's retrospective, using the same structure from chapter 4.

6.7.4.1 Team Management Processes

For the next cycle, new team members needed to be recruited and a subject related to social robotics would be conducted by some laboratory members; this subject would act as an improved version of the workshop held in this cycle.

As previously reported, the concept of proactivity would be researched, aiming to get team members to be more autonomous and productive while lowering, or distributing, the management workload.

To distribute the managerial workload and also to allow for subteams to direct their work as preferred, experienced key team members would take care of setting up roadmaps for their areas of interest; these could be considered as product owners in a simile with the Scrum methodology. Regarding the backlog management, the practices of attaching images and using checklists for goals of tasks would be encouraged and, additionally, proposed deadlines for those tasks.

The current ORTs were successful, so these events would stay, however, opportunities to improve them further would be researched. Stricter focus on scoring earlier on, while maintaining trade-offs when systems are not yet ready, would be explored as previous experience showed that it did provide more robust and higher scoring.

As a new architecture would be developed, requirements, and desired features would be gathered. Care would be taken to keep the team involved in implementation and interface decisions. Once implemented, the framework must be introduced with a well-prepared presentation with meaningful demonstrations and documentation to avoid problems like those found in this cycle. Furthermore, training would be provided. This training to be held by other team members, ideally interns, not the lead developer, to ensure knowledge transfer and avoid depending on a single person's knowledge and opinion.

6.7.4.2 Team Software Development Processes

Given the issues raised in this cycle, for the next cycle, automated testing and continuous integration would only be enforced on the projects that definitely need it. Team members familiar with such an approach would help others to work in this way. Coding standards regarding naming and formatting would stay as they were not raised as an issue and provided higher quality in the code.

This year's teamwork showed that knowledge sharing was successful, but software reuse could be improved. Further effort would be made in encouraging documentation practices that were regarded as low additional workloads, such as having a README file on every package with a description of the package and examples of usage. Additionally, pairing team members for the development of RoboCup tests would continue, and pair programming would also be encouraged for those team members that enjoy it.

6.7.4.3 Technical Approaches

A new architectural vision was to be developed for the next cycle embracing further ROS elements. The implementation would provide finer control of the processing pipeline from sensor information to high-level behaviors and applications. Care would be taken to simplify APIs that were reported as complex. Finally, providing efficient network usage and transparent remote nodes would be one of the main features of the architecture while not blocking anyone from using plain ROS code.

The navigation system would be overhauled by separating its state machine in easier to test individual pieces by means of the move_base_flex [105] ROS package. Moreover, the localization system would explore using RGB and RGBD approaches.

Further effort into efficiently using state of the art deep learning approaches for perception would be undertaken, particularly, for object recognition.

Additionally, a new HRI system that would allow a centralized approach to social behaviors would be developed, allowing the team to convey a style or personality for the robot without changing the code of every RoboCup test. This system would extend further from what RoboCup needs but was aligned to the lab's interests.

Finally, the speech recognition and generation system would be reimplemented to allow for improvements that were difficult to implement this year.

6.8 Possible Guidelines

Based on the previous guidelines, the most important findings and reflections from the year of development will be summarized; these will be refined and extended every year.

- Perform end-to-end testing. Simulation of the full competition including setup on a previously unknown place, strict schedule, and timing, strict scoring, by a strict referee, naïve users and other elements that otherwise may be overseen. Examples of other elements can be networking issues, environmental noise, unavailable team members, or unusual lighting. Start performing these as early as possible, team members new to the competition will grasp a better idea of how the competition looks like. Allow flexibility when necessary, e.g., if some part of the system is absolutely not ready, allow to bypass it.

- Find a balance between what the team wants and what is believed to be the best for the team regarding development processes. This is something to be re-evaluated during the development year. Topics such as coding standards, testing standards, documentation, deployment strategies, and code sharing approaches fall into this category. Invest in understanding what is considered hard, and work on improving the situation. Example improvements can be finding alternatives or providing further documentation or tooling.

- Provide the best tools available for the job. Development tools, frameworks, simulations, and hardware are topics that fall into this category. References for these can be taken from other teams or industry. Provide documentation and training for these too. The example here is the ROS middleware, widely used in the community of the competition and industry for robotics applications.

- Ensure teamwork is possible and encouraged. Set up a backlog of tasks and distribute them, taking into account every team member's interests. Promote grouping team members to work more effectively and share responsibility. Promote pair programming for knowledge transfer and to fix difficult bugs. Consider using a physical backlog, additionally to online means, in your working environment as a clear and quick way to visualize progress.

- Test as often as possible, on the real robots and in a realistic environment. Having unit testing is a great tool; testing with real stored data is also helpful, but the only thing that will matter in the end is that the competition tests run robustly.

- Focus on scoring. Implement and test with that in mind. Over-engineering a robot skill for cases that will never happen implies that time could have been invested somewhere else of higher priority.
- Start with simple approaches and iterate improving them. This can be applied to the development of capabilities for the robot and the design of approaches for competition tests.
- Architect your system with software integration in mind. Integrating a lot of software from different sources with different mindsets can be hard, and it should not be left as a final exercise.

CHAPTER 7

RoboCup@Home SSPL: Year 2019, 1st place

2019 was the third year of the existence of RoboCup@Home Social Standard Platform League (SSPL) competition, the third year for the *UTS Unleashed!* team to participate in it, and the last year of participation for the team (at least in regards to this dissertation). The 2019 rulebook [106] presented meaningful changes.

As in the previous 2 chapters, this chapter follows the structure of the diagram in Figure 4.1 which itself follows the structure of Action Research (AR) cycles in Figure 2.2. Every AR cycle is composed of a year of preparation for the competition, participation in it and reflection on the process itself.

In each section the AR cycle setup is explained: the framework of ideas, the methodology, and the area of concern for that cycle are first discussed as per the structure explained in chapter 2 and chapter 4. The context for the year follows. Its main themes are the rule and competition changes of that year and the team composition of that year.

Following, the software development plan is presented. It is analyzed in three subtopics that will be repeated over every next subsection: team management processes, team software development processes and technical approaches.

Afterwards the software development implementation presents the same subsections but in the context of the plans put in practice and their iterations.

Then the competition participation is showcased, again with the same subsections. Also with the results, as competition outcomes, for that edition.

Near the end of the chapter we find the post-competition data collection and retrospectives. Here the available data about the year is collected.

Finally the reflection section summarizes and discusses the findings (following again, the same three subsections structure), and briefly describes the next steps. The very end of the chapter consists in the proposal of guidelines based on the reflections on the cycle.

7.1 Action Research Cycle Setup

Our research questions for this cycle come again from the previous reflection accompanied by their observations:

- How to architect the system?

  An architecture was proposed and attempted but it failed for diverse reasons, as seen during the cycle, further efforts were needed for the next cycle.

- How to transfer knowledge?

  Knowledge transfer improved compared to the previous cycle but the lead developer believed that there was room for improvements. The failed architecture aimed to help on this approach and as part of a new architecture the concept must be revisited.

- What's the best shape for a robot API?

  The architecture for this year aimed to allow for a variety of styles of usage and implementation of robot APIs. These were not as successful as the lead developer wished keeping this question open.

- Can we improve further ORTs?

  This year the Operational Readiness Tests (ORTs) were perceived as preparing the team successfully for the experience at the competition, however, as ORTs were identified as a key element for the team success, it is worth iterating on them further.

- How to structure code for improved sharing and reusability?

  Given this years architecture did not provide a clear improvement in this topic, it is worht further exploring in the next cycle.

- How can we distribute the work during the year so we do not have much more work load close to the competition dates?

  It was believed that as this year had a longer period of time for development, compared to the previous cycle, the workload could be spread evenly during the year

but it turned out to not be like that. To allow for a lower workload in the last weeks before the competition, to improve the team members experience, it is worth further thinking about it in then next cycle.

- Can we further increase and/or improve teamwork?

  Teamwork increased this year as seen in the Git data and in the retrospective document and it was believed by the lead developer that it was a key factor for the success of the team. Given that, it is worth investing effort in the next cycle to further improve.

- How to find new team members that are proactive?

  The lead developer believed that proactive team members can be beneficial for the team and aimed to encourage this behaviour with the interns but failed to do so. As it could be a personal trait looking to recruit team members which are already proactive was to be taken into account for the next cycle.

- How to increase proactivity in the team members?

  An attempt was made to increase proactivity in the interns but it was not successful. Further research in this direction may benefit the team.

7.2 Context

This year was the last year the team had plans to participate in RoboCup@Home SSPL. As a three year plan, with the previous two years achieving second place in the competition, further pressure to win built up which could have affected morale.

With seven team members on their third year of competition (albeit only three of them doing software development) and three team members on their second year, there was a lot of expertise and knowledge that could aid the team. The lead developer and project managers had an optimistic view for the year.

7.2.1 Rule and Competition Changes

This year the competition was scheduled for the 2nd to 8th of July of 2019 in Sydney, Australia. There were roughly 9 months to prepare for the competition (after a break from the previous year), similar to the 2017 year.

This year a major rework of the rulebook [106] was done. The schedule changed to adopt the concept of thematic scenario blocks. The themes were called *Housekeeper*, where tasks revolving around finding and moving objects around the home took place, and *Party Host* where emphasis was put into social tasks as in helping guests in a party.

Every day two time blocks were scheduled lasting from two to three hours. All teams were allocated at least 2 testing slots per block in which they could test any task of their choice from the block's assigned scenario. All teams got the same amount of slots, with a minimum of 2 per block. If there was extra time, another extra slot could be added. Teams must inform the organizers in advance which tests they were going to perform in each of these slots.

The Open Challenge was a test where each team would present their own research in an appealing way to an audience of team leaders and referees who would score it. The robots must perform some kind of autonomous task but the teams were free to use any technology they wanted. The Open Challenge test was made optional and non-scoring, even though the Finals kept the same format.

The biggest change was the scoring system. Each test now had a task with a main objective and a set of scoring bonuses. To score in a test a team must successfully accomplish the main objective of the task, otherwise bonuses were not scored. Previously, every test had some subtasks to accomplish that would provide partial scoring. This made it so a team could, for example, not accomplish a full task where the robot must get an order and fulfill it, but get partial scoring for understanding the order. Now the team would only score if the robot fulfilled the complete order.

A rule called *Deus ex Machina* was created to be able to bypass features with human help. This rule enabled teams to score even when a robot was not able to fulfill a partial task although it accomplished the main goal of the task. An example of this rule was presented as the robot asking for help from a human to point to the direction of whatever the robot was looking for. Making use of the Deus ex Machina rules reduced scoring of a successful task, and also made the team unable to score any possible bonus score in that task.

The new scoring, and the Deus ex Machina rule attached to it, made the approach for robustness to score even more important than ever.

In Stage I, five tests for each category of Housekeeper and Party Host were created. All tests but one (General Purpose Service Robot, as it stayed mostly the same) had familiar elements from previous rulebooks but were new.

In Stage II, four tests of each category were created. An additional test, assigned to

the Party Host scenario but treated as a major test for everyone to compete, was added, the Restaurant test. This test was not included in the slots count, but everyone was able to participate if they wanted to.

7.2.2 Team Composition

The team was composed of 17 people. Recruiting was performed during the months of October and November. The interns from the previous year stayed, funding was secured for them. Additionally, more team members joined the team partially fulfilling work for subjects of their degrees.

The team was bigger than the previous year. More than half of the team were part of it for the second or third year in a row. This year 14 members were on the role of developing software.

The interns were the same undergrad students in Computer Science and/or Information Technology from the previous years. They improved their software development skills during the previous year but were still not considered experts as they needed monitoring and guidance on their work.

The new team members were mostly undergrad students looking to fulfill subject projects with the work done in this project. The rest of the developers fit the description of the previous year being mostly PhD students and staff from the lab.

7.3 Software Development Plan

As in the previous cycles, this section explains the software development plan for this edition of the competition. It includes the team management processes, the team software development processes and the technical approaches planned.

7.3.1 Team Management Processes

This year contained a new and unforeseen situation: the lead developer had to take some time off due to mental health issues. In December and January his time commitment with the team was lower than usual. In February and March he was completely gone. He came back in April, progressively, to come back full time by the end of the month.

A new summer studio organized by some lab members that would allow the team to find new team members and to evaluate and improve the available software was held.

#	Team Role	SW Expertise	Background	Time Dedication
1**	Project Manager	–	Lab Director	On demand
2**	Project Manager	–	Project Manager	14h/w
3**	Assistant	–	Media Specialist	On demand
4**	Lead Developer	Expert	Robotics & CS	32h/w
5**	Developer	Expert	Computer Science	10-20h/w
6**	Developer	Expert	Lab Co-Director & CS	20-40h/w
7	Developer	Expert	IT & CS	20-40h/w
8**	Developer	Advanced	Computer Science	20-40h/w
9*	Developer	Advanced	IT & CS (Intern)	20-40h/w
10*	Developer	Advanced	IT & CS (Intern)	20-40h/w
11*	Developer	Advanced	IT & CS (Intern)	20-40h/w
12	Developer	Advanced	IT & CS	4h/w
13	Developer	Moderate	CS & AI	On demand
14	Developer	Moderate	IT & CS	20-40h/w
15	Developer	Beginner	IT & CS	20-40h/w
16	Developer	Beginner	IT & CS	8-16h/w
17	Developer	Beginner	IT & CS	8-16h/w

Table 7.1: Team composition table for UTS Unleashed! in 2019. Team members marked with one asterisk (*) are on their second year of participation in the team, members marked with two asterisks are on their third year with the team. [1]

This summer studio created an activity similar to a RoboCup test to be solved by the students.

7.3.1.1 Team Management Tool

Trello was used again as the online tool to keep track of work to be done. This year it contained the following 5 boards:

[1]CS stands for Computer Science, IT stands for Information Technology, AI stands for Artificial Intelligence.

- Marketing: interns took over the marketing job from the previous year.
- Software Engineering: the previous year's board was very successful so it was replicated, with some tasks being taken over from the previous board. Checklists for tasks in cards were widely used.
- RoboCup Tests: as the board was succesful the previous year, it stayed. Deadlines for cards were widely used in this board. Check boxes for tasks in cards were widely used.
- Robot Task Planning: this board was expected to be used to continue the work on the planner done the previous year by its author.
- Troubleshooting: board to share fixes and workarounds for different issues.

Creating lists with checkboxes and setting goal dates on some Trello cards seemed to help at the initial stage of the development year, so this year that practice was encouraged further as a refinement of our methodology.

Introduced this year, roadmaps were created for every major technical topic for the competition: navigation, HRI (tablet interface and user experience), speech recognition and text to speech, and perception. These roadmaps consisted of a description of the vision for the topic and a prioritized list of approaches to research.

7.3.1.2 Team Meetings

The plan was to hold team meetings every 2 weeks and as the competition got closer, every week. Additionally, ORTs were to be held monthly, fortnightly as the competition approached, and weekly by the end. This practice followed up the trend from the previous year refining it by planning them in a shared calendar.

7.3.1.3 Task Assignment

From the roadmaps, tasks were extracted and distributed via Trello cards. There was a leader for every roadmap and by meeting with this person work was distributed to the most interested or capable people. This organizational approach was decided while there was no final rulebook for this year. Once the rules were finalised, reorganization was necessary and subteams were created for every test to be implemented.

7.3.2 Team Software Development Processes

This section presents the practices and tools regarding how the team was to develop software.

7.3.2.1 Coding Standards

This year a new analytics system was to be implemented to keep automatic, or as automatic as possible, track of different insightful data.

Software packages were to be written so they could be packaged either as Python pip packages or Debian packages to ease distribution and installation refining the practice from doing it casually the previous year.

7.3.2.2 Coding Tools

A new framework called magic_ros was developed to ease the usage of Robotics Operating System (ROS) for the team. On top of that, tools to ease other areas of interest were to be created.

7.3.2.3 Social Coding Practices

The interns from the previous year were proposed to take the role of presenting new tools/libraries in workshops to the team to help them improve their transferable skills. They were to be developed alongside expert team members. This was an extension from the previous year were workshops were provided but these were presented by the authors of the tools themselves.

7.3.3 Technical Approaches

As in the previous cycles, the technical approaches, or specific decisions made for the competition itself, are discussed in this section. These are related specifically to the RoboCup@Home SSPL and describe the competition requirements, the robot's capabilities, and the software stack.

7.3.3.1 Competition Requirements

The new rules made it necessary that the robot would perform whole tasks robustly. However, as the Deux ex Machina rules could be used to overcome problematic steps, the tests were to be prepared by first programming the backup behaviors, and then, add

the totally autonomous behaviors. This was to ensure there were well tested backup behaviors and also to ensure there was always a way to, at least partially, score.

To choose the tests to be implemented, the available skills and capabilities expected to be developed for the year were taken into account. The tests were chosen by consensus during team meetings, this was a extension from the previous year. The chosen tests for Stage I follow:

- Housekeeper scenario:
  - **Clean Up**: Some misplaced objects are inside one room in the arena. The robot has to tidy up that room by throwing those misplaced objects in the garbage. The main goal of this task is to find all misplaced objects in a room and bring them to their predefined locations.
  - **Take Out The Garbage**: All garbage bins in the apartment are to be emptied and the garbage moved to a specified collection zone. The main goal of this task is for the robot take out the trash bags from the two bins in the apartment.
- Party Host scenario:
  - **Find My Mates**: The robot fetches the information of the party guests for the operator who knows only the names of the guests. The main goal in this task is to report to the operator the description and location of at least two party guests.
  - **Receptionist**: The robot has to take two arriving guests to the living room, introducing them to each other, and offering the just-arrived guest an unoccupied place to sit. The main goal of this task is to introduce and allocate two newcomers in a party.

It is noteworthy that even though the General Purpose Service Robot (GPSR) test still existed, the issues with its development the previous year (including the fact that it needed to have all possible skills for the robot ready) discouraged the team to invest their efforts in preparing this test.

As part of the effort to improve the rulebook in the interests of the RoboCup@Home SSPL, given the team perceived that some tests could be improved regarding the social aspects of the tests, the lead developer joined the technical committee. Within this role the *Find My Mates* test was proposed by the UTS Unleashed! team and accepted into the rulebook.

Additionally, for Stage II the following tests were chosen:

- Housekeeper scenario:
  - **Find My Disk**: The robot helps a blind person to locate an LP, compact disk, or audio-cassette in a shelf. The operator will ask the robot to describe what it sees in either the shelf or the media cover, but will not allow the robot to touch these treasures. The main goal of the task is for the robot to provide a description of an object that matches the operator's requirements.
  - **Hand Me That**: A guest at the party speaks English, but with only a limited vocabulary. The robot will assist them in obtaining things that they gesture for. The main goal of this task is for the robot to identify (touching or naming) each object at which the operator is pointing.
- Party Host scenario:
  - **Restaurant**: The robot retrieves and serves orders to several customers in a real restaurant previously unknown to the robot. The main goal of this task is to take and serve an order to a customer.
  - **Where is this?**: The robot has to explain and show people where they can find places in the arena (e.g. *Where is the TV?*). The robot has to tell the operator how to get there (in a summarized fashion) and then physically take the operator there in the mode of a tour guide. The main goal of this task is to give accurate directions and guide at least 3 people.

The finals were, once again, left to be developed in the competition itself. The development lead had a strong belief in their ability to prototype and test quickly.

Furthermore, even though the chosen tests are reported here, some tests, especially for Stage II, were only decided later on during the development year. A list of pros and cons for every test that team members proposed, was created. Then, a week was allocated for doing some research and development to explore how viable it was to implement these tests. Finally, the team voted for which tests to implement.

7.3.3.2 Software Stack

The software architecture was designed with strong inspiration from the lessons learned the previous year. The architecture aimed to have a basic framework from which to base

the software stack. This basic framework was called **magic_ros** in reference to the name of the lab: *The Magic Lab* and to the ROS framework.

The **magic_ros** framework is designed to implement elements that ease the usage of ROS (via autodiscovery and helpful error messages) and also provide tools to make transparent, easy and robust usage of external servers from the robot, i.e. via unreliable WIFI, when these are available. It also aims to provide automatic logging, real time state feedback, analytics, live reconfigurable variables and new data acquisition modes.

Over this framework a set of friendly-named packages[2] were created:

- magic_navigation: To contain a re-implementation of the navigation stack by using move_base_flex[3]. This change also allowed for easy experimentation with the different pieces of the navigation stack, with special mention to the local planner[4]. Different mapping and localization strategies will also be evaluated and implemented here.

- magic_apps: To contain applications and drivers that fulfill necessary roles. Aims to create new drivers for some key elements of the system, for example, a new laser driver or providers of images that can be queried for images instead of subscribed to a stream of images.

- magic_vision: To contain computer-vision related utilities. With an interest on well known working deep learning elements.

- magic_listen and magic_speak: To contain the code related to speech recognition and speech generation. Aiming to more efficiently and aggressively use Google services for speech recognition with a local fallback, thought to be PocketSphinx [107][5] at this point.

- magic_tablet: Mostly the same project from the previous year but with some updates.

[2]The packages were friendly-named compared to the previous year where many packages had creative and fun names. Those names made finding packages and reasoning over them more complicated.

[3]move_base_flex allowed custom state machines to be implemented for the navigation interface. Traditionally the state machine is fixed and implemented in C++. This package allowed it to be implemented in Python with total freedom.

[4]A navigation local planner has the job of creating and following short local trajectories that follow a global trajectory, i.e., given a global trajectory that goes from the living room to the kitchen, the local planner will be in charge of creating and following shorter trajectories that avoid obstacles and try to follow this global trajectory.

[5]PocketSphinx is a lightweight speech recognition engine, specifically tuned for low power devices as mobile devices.

- magic_hri: To implement a unified human robot interaction approach for the robot.

However, a topic that did not have a roadmap but had its necessary actions in Trello was the base dependencies of the robot build. Previously called ros_pepperfix, the project compiled all the necessary dependencies for the rest of the code of the team. For this year the aim was to automate the building in Docker and with nightly builds on free continuous integration resources in the cloud. This work was to be done open source so the community could get engaged and help. The final custom image for the team's robot ended up being called pepper_os.

Finally, a simulation for the Pepper robot via Gazebo was also to be developed (more precisely, fix the one available that was not working for navigation purposes).

7.4 Software Development Implementation

This year's software development was marked by a prolonged period of absence of the lead developer, however, teamwork kept improving.

7.4.1 Team Management Processes

As the lead developer had to take some time off, the team management went through a few hard to delimit phases. Generally, the processes learnt during the previous year were to be kept with even more emphasis on the ORTs.

7.4.1.1 Team Management Tool

The Trello boards were updated from the previous edition. The final boards were:

- Software Engineering: widely used, task oriented cards using checklists.
- RoboCup Tests: widely used, task oriented cards using checklists and also due dates.
- Marketing: quite well used by the interns working on it. Lots of checklists used.
- Robot Task Planning: this initiative was shutdown by group vote as the previous year this work created a lot of conflicts in how it was handled.

- Troubleshooting: only one team member used it as a kind of resource to review bugs that that person had fixed. Others used snippets or gists[6], some local files, the wiki or other means.

Both checklists and due date deadlines were widely used at the start of the activities on each board, but lowered in usage as time went. These seemed like a good tool to get started, but once people were comfortable working, they skipped it and kept their own personal work-tracking means.

7.4.1.2 Team Meetings

Sub teams would meet weekly meanwhile ORTs were held fortnightly. Every month there was a more rigorous ORT that involved more setup than the fortnightly ones. Focus was set on scoring as this year scoring was very hard given the requirement of performing whole tasks to be able to score.

The days when ORTs were held included lunch provided by the university to the team members. Eight ORTs were held from the month of April. They were held mostly fortnightly but as the situation made it necessary (and as the competition approached) they were held weekly. Unfortunately the scoring for these sessions was not preserved and can not be presented here as in 2018. But the team leader recalled that most sessions had the tests score 0. This was due to the model of scoring in this year's rulebook, which made it very hard to score. By the last ORT the team did score on every Stage I test and in some Stage II tests.

Thursdays were designed as work-in-the-lab days. Setting an "official" day to work together encouraged teamwork. This was a reaction to the period where the team leader was absent which made team members attend the lab less frequently.

7.4.1.3 Task Assignment

The development lead was unavailable for some time. This introduced various impediments. To start, it was not clear the level of involvement (or lack) of the team leader. Emails were still being replied to but he was not attending the lab. However, at some point the development lead was not available at all, and the team had no backup for this role.

[6]A Gist is a hosted snippet of code that has all of the benefits of an online hosted Git repository but is presented in a more lightweight fashion. Gists are the name given by GitHub, other platforms call them just snippets.

A new team leader was looked for. The role was offered to promising team members that would benefit from the experience. No one was found willing or able to perform this role. Another experienced team member kickstarted the project by creating subteams for RoboCup tests and setting clear and attainable goals. Each subteam had a team leader. Focus was put on having ORTs and rotating who would be the chair of meetings and ORTs. The shared leading effort was unsuccessful but at that point the lead developer came back into the project regaining that role.

7.4.2 Team Software Development Processes

This year's software development processes were characterized by a generalized improvement in teamwork.

7.4.2.1 Coding Standards

The magic_ros framework was implemented providing some useful tools and helpful error or warning messages. Using magic_ros was not widespread, neither was it enforced, but more encouraged in the places where it made sense.

Mindful usage of standard coding practices were done. Base libraries had unit tests and automated testing developed. Continuous integration and deployment was set up in the necessary repositories. The strictness to adhere to these practices was evaluated separately for every project and person to allow for a smooth development experience for all the team.

7.4.2.2 Coding Tools

Following the architecture planned at the start of the year, tools written on top of magic_ros were created for tasks like computer vision or testing remote servers.

Most team members started using Microsoft Visual Studio Code, some substituting it over the previously used Sublime Text, as their main code editor. This introduced some unification on the development environments.

Some team members made use of Jupyter Notebooks to conduct their research and to share their discoveries.

The team made extensive use of the powerful computers that were available from the previous year. Deep learning based approaches for a broad set of topics were tested.

7.4.2.3 Social Coding Practices

The interns led workshops over the libraries and packages created for this year. These were perceived positively by the attendees and the interns themselves. The interns from the previous year did some mentoring over the new team members as a means of improving their leadership skills.

Code reviews were done from time to time. Pull Requests were used as a learning tool in some instances.

Team members used to work in pairs in the different RoboCup Stage I tests. Once they were finished they were free to swap into working in a Stage II test of their liking or support other subteams.

7.4.3 Technical Approaches

This year's technical approaches were characterized by the introduction of the team's framework to base their system on.

7.4.3.1 Software Stack

As in the previous year, given some tests could be run at least 2 times, tests that may be implemented in a more riskier approach but could provide a higher scoring had parallel implementations.

As planned, the work on the base OS was improved (magiclab_pepper_os), streamlined and automated with nightly builds via Docker to ensure its usability at all times. The team froze a version a month before the competition.

The architecture was implemented as planned via magic_ros [7]. Base skills were either re-written or added as a wrapper on top of magic_ros.

The team started doing some analysis and benchmarking of different approaches for the different systems. Drivers and robot skills were written from scratch using the magic_ros framework to gain further control of some sensors of the robot and optimize network usage. Namely:

- magic_ros features:
  - Reduced complexity in ROS usage
  - Resource providers

[7] Details of the software stack also appear in the publication *UTS Unleashed! RoboCup@Home SSPL Champions 2019*, [108] published in the RoboCup 2019 Symposium Proceedings.

        - Subscription and query interfaces
        - Queryable buffers (get data at timestamp, get all buffer, etc)
        - Online reconfigure capability via ROS' dynamic_reconfigure
        - Efficient data transport
        - Easy and automated logging
    - Automatic logging and diagnostics

- Drivers based on magic_ros (inheriting provider capabilities):
    - Laser: more precise driver running at 10Hz
    - Cameras:
        - RGB: calibration, low resolution
        - Depth: linear filter to improve image, calibration, lower resolution
        - Unified Calibrated RGB + Calibrated and filtered Depth provider + robot transforms provider
    - Sonars: custom driver at low rate (10Hz)
    - Transforms (TF): minimal TF tree (no arms, no fingers), low rate (20Hz)
    - Odometry: custom driver at low rate (20Hz)
    - Robot base speed driver: (ROS cmd_vel Twist driver) with on the fly reconfigurable speed limits. It also supported timeout of commands, if no new command received in 0.5s the robot will safely stop. It overrides all stock Pepper safeties
    - Safety node: touching the head of the robot or the bumpers stops the movement of the robot

- Navigation:
    - Mapping: 2D via RTAB-Map using RGB+Depth
    - Localization: via RTAB-Map using RGB+Depth
    - SLAM: via RTAB-Map using RGB+Depth, running onboard at 10Hz
    - Custom navigation State Machine via move_base_flex with optimizations for special cases
    - Local planner: Time Elastic Bands planner using Laser + Depth sensor for obstacle avoidance

  - NaoQi wrapped APIs: native APIs available but monitored by the navigation system
  - Pausable computing: save CPU by only running navigation nodes when the robot needs to move

- Speech:
  - Text to Speech
    * Save CPU by caching audio generation
    * Logging
  - Speech to Text:
    * With and without grammar
    * Offline and Online: Sphinx / Google
    * Logging
    * Optimized network audio traffic via Opus codec

- Perception:
  - People perception:
    * Face detection
    * Face recognition
    * Facial features detection: hair color, facial hair, age, gender, glasses.
    * Shirt color detection: color or multiple colours and with support for text recognition.
    * Human skeleton detector: 2D and 3D pose of joints.
    * Pose detector: sitting, standing.
    * Gesture detector: waving, pointing.
  - Object recognition
  - Optical Character Recognition

- Human Robot Interaction:
  - Unified HRI by templates
  - Tablet interface

Analysis of the odometry and the behavior of the robot when sending velocity commands to the base allowed the team to experiment with the navigation stack for Pepper in simulation. It is worth noting that the available simulation at the time of writing didn't work, so a forked version was created[8]. A custom plugin for Gazebo was developed and a configuration closer to what was observed in real life with the team's robot was created.

Before finding RTAB-Map to perform correctly other options were explored, including the approach from the UChile team where they used ORB-SLAM [109]. During the development year there was a crisis where the navigation system had a bug that made the robot behave in uncontrolled ways. During this time another team member started a parallel approach in case this bug could not be resolved. This team member also took care of supporting the lead developer in this difficult time. After a huge investment of time the bug was overcome and the navigation system behaved satisfactorily.

For Object Recognition a few deep learning approaches were researched until YOLOv3 [110] was found to perform robustly enough with an affordable training time. Other RoboCup teams seemed to also use it which was interpreted by the team members involved in this topic as positive.

7.5 Competition Participation

The development lead felt more confident than any other year on being able to have a great performance given the outcomes of the last ORTs and the general team spirit.

7.5.1 Team Management Processes

This year followed a similar approach to 2018. As the team had more experience, the lead developer thought that stricter following of known-to-be-working elements could be beneficial.

7.5.1.1 Team Management Tool

The usual whiteboard with To Do, Doing and Done columns with post-its was used again. Disregarding Trello as in previous years.

All RoboCup tests were recorded in video by one or more team members for review as usual.

[8]Which can be found at https://github.com/awesomebytes/pepper_virtual.

7.5.1.2 Team Meetings

A team meeting in the morning just before entering the venue and another team meeting every night just before leaving the venue were held. Other meetings were scheduled as needed, for example, after test runs or major incidents.

This year most people had meals in their own way, with the project manager assisting organizing them.

As usual, meetings took care of optimizing the usage of the time in the venue and the usage of the arenas.

7.5.1.3 Task Assignment

Tasks were assigned by using the whiteboard. Everyone would take tasks and ask for help if they needed collaboration in a task. The team meetings made sure that everyone's time was used as well as possible.

Most team members worked in pairs for testing anything on the robot or for delicate practices (for example, labelling pictures of objects for object recognition) where two pairs of eyes are better than one.

As in other years tasks that would block others from working would be prioritized. The usual suspects appear in this practice: mapping the arena, getting the data for the different elements of the tests etc.

Multiple people were on the task of keeping the robots and the laptops charged.

7.5.2 Team Software Development Processes

As in the previous year, the goals were to minimize development and maximize testing to ensure robust and high scoring runs.

7.5.2.1 Coding Standards

This year, more than any other, only configuring, testing and fixing minor bugs was encouraged in the competition. And even more than any other year, code that was not tested beforehand was forbidden to be run in an official run of a RoboCup test. No last minute changes were allowed.

As in 2018, if any base tool needed some hack to keep going, this was done in a different branch (namely "robocup2019") to be improved later on, if there was interest in doing so.

7.5.2.2 Coding Tools

The setup of the robot was the same as 2018, as it worked as intended without issues - that is, a single compressed (.tar.gz) file contained the full robot state. As this year the robots were not rented but they were the actual robots from the lab, this setup was done before the competition. That provided some extra time for the team during the setup days.

Almost every system and RoboCup test had simple scripts to test isolated functionalities. These were usually documented in the README of every project. This accelerated development in situ.

A set of automated tests were created in some repositories that were considered critical, like the base framework magic_ros.

Moreover, a local GitLab server was available this year as in previous years.

Furthermore, the team had a 4G router ready to be used in case connectivity completely dropped. In previous years some team members had local data on their phones and they used them when needed, but this year the team addressed the topic directly.

7.5.2.3 Social Coding Practices

This year most activities consisted of configuring and testing the work done during the year. Working in pairs, with one person taking care of the robot, the arena, and the interactions with the robot, and the other on their laptop, was determined to be the optimal approach.

Experts in every field would be ready to help on the setup of the RoboCup tests that used their part of work. Moreover, a responsible team member for every subsystem and for every RoboCup test, with a backup person in case of need, was scheduled. Finally, the lead developer stayed free of work as much as possible to assist all team members.

7.5.3 Technical Approaches

This year's technical approaches were characterized by satisfactory performance with excellent network usage.

7.5.3.1 Software Stack

The tests in Stage I were well developed and tested. Minor fixes were needed.

Stage II tests were developed but needed further testing in the real environment. The Find My Disk test had a lot of new development done during the competition including a full night of work by some members.

The network was felt to perform well by our team. This was not the same feeling in other teams, but we had optimal usage of the network. Only the minimum amount of traffic was sent, even for monitoring what the robot was doing in development and debugging times.

Our main systems performed satisfactorily. Navigation had some glitches with an impossible to detect table in the livingroom. This table was low and black, with metallic shiny legs. Neither the lasers nor the depth camera could detect it. Only the sonars could detect it sometimes. Other than that the new localization approach based on RGB + Depth data using RTABMAP performed well.

Speech recognition worked well this year. There were moments where it was hard for it to work due to the high noise levels, but all in all, it went well. The Find My Disk test had open questions asked to it and it worked well, and that was one of the hardest things to accomplish. A optimal usage of Google speech recognition was used, with an on-board backup.

People perception worked well too with a lot of usage of the OpenPose package. Due to the availability of the external server to compute it quickly and a backup running on the robot that was highly optimized, the robot could easily find people and perform logic on their poses. Other deep learning networks were used in the same fashion and worked well.

Other systems in the robot worked well too. For example, there was a service to stop the computation of the full navigation system when the robot was not moving to save CPU for other tasks.

Some tests were prepared with different approaches - in other words, with multiple implementations. Taking advantage of the fact that the team had a minimum 2, and usually 3 runs for every test in Stage I, the best one for the situation was used.

Many tests had automated logging in rosbags or other systems, of the data that went through. This was especially useful in the Find My Disk test that had a 500 point bonus for delivering to the referees a report about the test.

7.5.4 Results: Competition Outcomes

The team did well in Stage I as can be seen in the scoring sheet in Figure 7.1. The team was in first place at that point. The only 0 score was in the Clean Up test. This was

a very hard test for the Pepper robot given it involved a lot of manipulation. It was intended that this manipulation be bypassed with the Deus ex Machina rule, but even so, the robot needed to perform a long set of actions to at most score just 40 points per object cleaned up. In general the Housekeeper themed tests were ill designed for the Pepper platform, only the UTS Unleashed! and LyonTech teams scored in one test in this theme in Stage I: Take Out The Garbage. Anecdotally most teams copied the UTS Unleashed! strategy to pickup the garbage bags which the team took as a compliment - the teammates working on it put great effort in finding the best possible strategy.

UTS Unleashed! scored the highest in Find My Mates (a test proposed by them) and second highest in Receptionist. Receptionist had the most participation of the Stage I tests (as every team could choose which tests they wanted to take part in). The rest of the tests all scored 0. That could prove that the tests, on one hand, were hard but, also, that some teams were not prepared well enough.

Social Standard Platform League

			Housekeeper					Party Host tasks					
		RIPS	Clean Up	GPSR	Storing Groceries	Serve the Breakfast	Take Out the Garbage	Carry My Luggage	Farewell	Find My Mates	Receptionist	Serving Drinks	Stage 1
1st	UTS Unleashed!	400	0				50			220	200		870
2nd	Uchile Pepper	400								0	250		650
3rd	LyonTech	400					50				0		450
4th	KameRider SSPl	350					0		0		0		350
5th	LiU@HomeWrec	350		0									350
6th	SinfonIA Pepper	350	0								0		350

Figure 7.1: 2019 scoring sheet of RoboCup@Home SSPL Stage I. The team was first at that point.

Some teams complained that the WIFI connectivity was slow. As UTS Unleashed! optimized their network usage in their architecture and made minimal use of it, they were not affected. Investing efforts in this direction seemed fruitful just as in 2018.

UTS Unleashed! moved into Stage II as the team expected. In Figure 7.2 it can be seen that the only 0 score was in the Hand Me That test. This was sad for the team, and especially for the team member dedicated to the object recognition pipeline, as both Clean Up and Hand Me That were the tests that would showcase this skill.

On the other hand, Find My Disk and Restaurant performed exceptionally well. They both scored highly in difficult tasks. The team also scored on Where is This? which also involved a lot of steps to be able to score, so the team was proud about this too.

Social Standard Platform League											
		Housekeeper				Party Host tasks					
	Stage 1	Clean the Table	EGPSR	Find My Disk	Set the Table	Hand Me That	Stickler for the Rules	Restaurant	Smoothie Chef	Where is This?	Stage 2
UTS Unleashed!	870.01			1500		0		900		100	**3370.01**
Uchile Pepper	650.02			0							**650.02**
LyonTech	450.00										**450.00**
KameRider SSPl	350.06										**350.06**
LiU@HomeWrec	350.05										**350.05**
SinfonIA Pepper	350.03										**350.03**

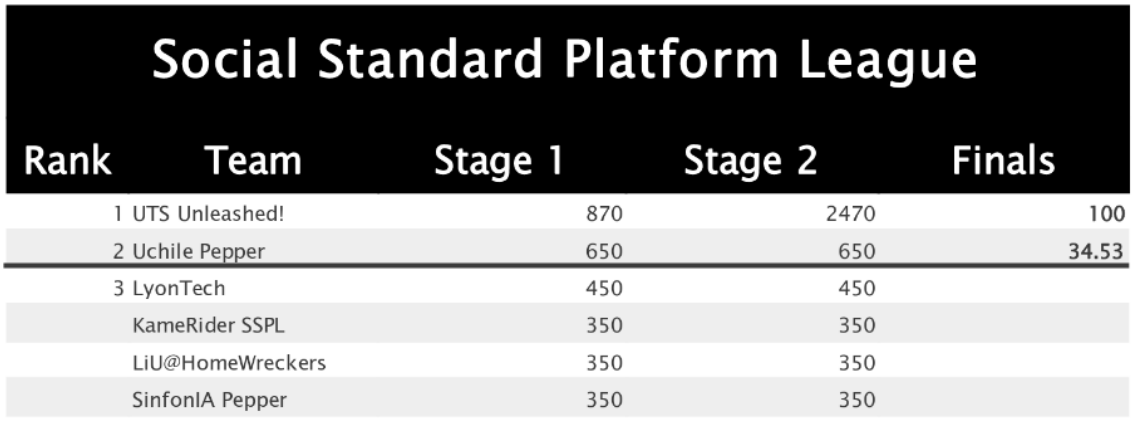
Figure 7.2: 2019 scoring sheet of RoboCup@Home SSPL Stage II. The team was first at that point.

It must be noted that the Stage 2 final scoring was modified to Figure 7.3 after a a difference of opinion in the interpretation of the rules. This decreases the scoring by 900 points from the Restaurant test.

		Housekeeper				Party Host tasks					
	Stage 1	Clean the Table	EGPSR	Find My Disk	Set the Table	Hand Me That	Stickler for the Rules	Restaurant	Smoothie Chef	Where is This?	Stage 2
KameRider SSPL	350.06										**350.06**
LiU@HomeWrecker	350.05										**350.05**
LyonTech	450.00										**450.00**
SinfonIA Pepper	350.03										**350.03**
Uchile Pepper	650.02			0							**650.02**
UTS Unleashed!	870.01			1500		0		0		100	**2470.01**

Figure 7.3: 2019 scoring sheet of RoboCup@Home SSPL Stage II, modified. The team was first at that point.

From there UTS Unleashed! qualified for the finals. The team composed a demo including the best prepared skills for the competition, showing them working all together. Given how far the second team was from UTS Unleashed! in scoring, there was confidence that the team would win. The finals were taken also as a opportunity to show some of the work that didn't perform as well in earlier competition tests.



Social Standard Platform League

Rank	Team	Stage 1	Stage 2	Finals
1	UTS Unleashed!	870	2470	100
2	Uchile Pepper	650	650	34.53
3	LyonTech	450	450	
	KameRider SSPL	350	350	
	LiU@HomeWreckers	350	350	
	SinfonIA Pepper	350	350	

Figure 7.4: 2019 final classification sheet of RoboCup@Home SSPL. UTS Unleashed! achieved first place.

UTS Unleashed! won the 2019 RoboCup@Home SSPL competition celebrated in their own city: Sydney, Australia. The team demonstrated a strong growth during the previous 2 years to finally achieve gold.

Comparing the UTS Unleashed! 2470 points, and maybe the 900 deducted from Restaurant too, with the other leagues in Figure 7.5 it can be observed that the team was on-par with teams in the podium in both Domestic Standard Platform League (DSPL) and Open Platform League (OPL). This is a very interesting outcome for not only the team, but the SSPL itself, as the Pepper platform has some perceived challenges to perform as well as robots in the OPL. Perhaps, some day Pepper robots will win in the OPL competition?

Domestic Standard Platform League

Rank	Team	Stage 1	Stage 2	Finals
1	TU/e	1940	3690	100
2	Team Tidyboy	1285	2285	68.25
3	Hibikino-Musashi	1390	1460	–
4	PUMAS-DSPL	1190	1190	
5	UNSW	870	870	
6	ORIon	600	600	
7	Team Northeastern	595	595	
8	Austin Villa	530	530	
0	RoboCanes	360	360	
	eR@sers	340	340	

Open Platform League

Rank	Team	Stage 1	Stage 2	Finals
1	homer	1715	3365	100
2	Pumas	845	2195	78.05
3	CATIE	1425	1675	–
4	Walking Machine	890	890	
5	RT Lions	650	650	
6	ACRV	438	438	
7	Tinker	367	367	
8	KameRider OPL	350	350	
9	RoboFEI	310	310	

Figure 7.5: 2019 final classification sheet of RoboCup@Home DSPL and OPL.

On a personal note, the team was understandably very happy to win after this three year journey. Being able to get gold and achieve the same level of scoring as very experienced teams in such a short time proved a great effort and great teamwork. It could also imply that the software development related practices the team practiced could be the reasons for this performance.

7.6 Post-Competition Data Collection and Retrospectives

Statistical data was collected from the Trello boards and from the Git repositories as in the previous years. A retrospective document was also collected from the feedback of the team members, these are discussed in this section.

7.6.1 Statistical Data

Every year the available data from Trello and Git was analyzed as explained in chapter 4.

7.6.1.1 Trello Cards Data

Trello usage this year was not constant, there were periods of usage and periods with lack of usage, with spikes in between. This was due the lead developer needing to take some time off, making the team look for new directions. The usage did decrease as the competition got closer, with a bigger amount of activity in December, the month where most planning and architecture design happened, just as in the previous year. The boards with most activity aligned with the expected activity during the year, similar to other years.

The Trello activity over the year of development is shown in Figure 7.6. This year Trello was used from the start of the development year, which was in November until a drop in usage happened in December. Afterwards, it took off again until the end of January. Then there was another drop in usage by early March, a spike of usage by the end of March, and some activity around May, with usage dropping again as it got closer to the competition date. This could be correlated to the lead developer needing to take some time off and the team adjusting to it.

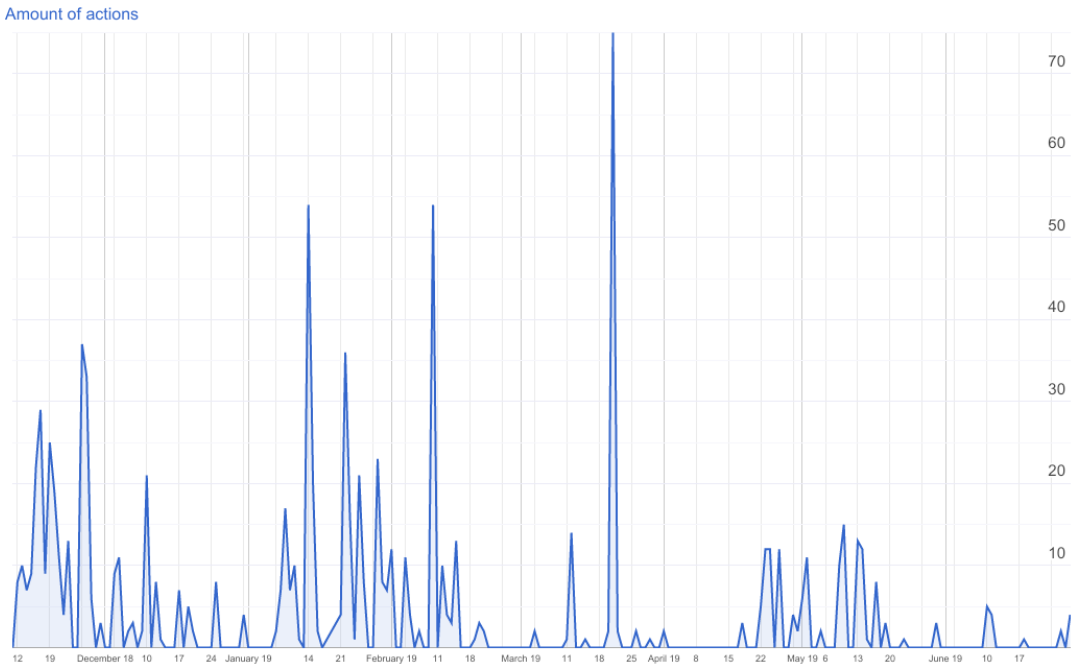


Figure 7.6: Activity (Trello actions) on the year 2019.

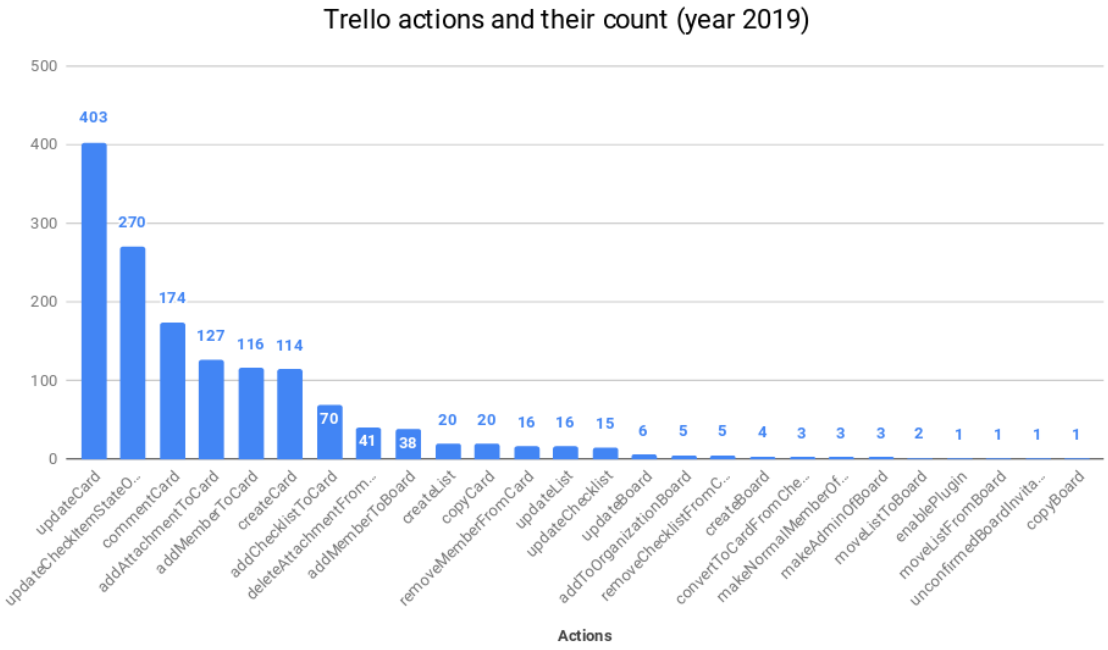


Figure 7.7: Distribution of Trello actions types on the year 2019.

Action types and their count in Trello for 2019 can be found in Figure 7.7. We observe

that updating and commenting cards keeps being the most common action as in previous years, but this year the usage of checklists increased compared to the previous year. Adding attachments, usually screenshots, kept being common.

Checking the distribution of actions month by month in Figure 7.8, December appeared as the busiest month correlating with the fact that planning and architecture decisions were made in that time. Then the usage of Trello seemed to decrease month after month as every year.

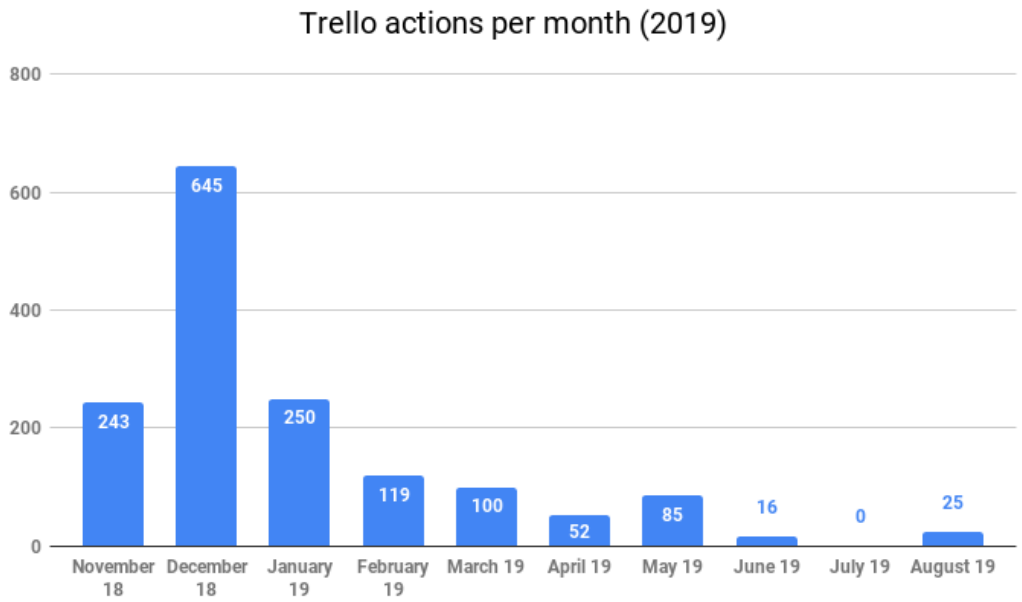


Figure 7.8: Distribution of Trello actions by months on the year 2019.

The Figure 7.9 showed that Monday and Tuesday had the most activity with the rest of the weekdays staying similar. Team meetings happened mostly on Tuesdays, so it could be interpreted that people prepared the day before, and updated the day of the meeting.

Figure 7.9: Distribution of Trello actions by weekdays on the year 2019.

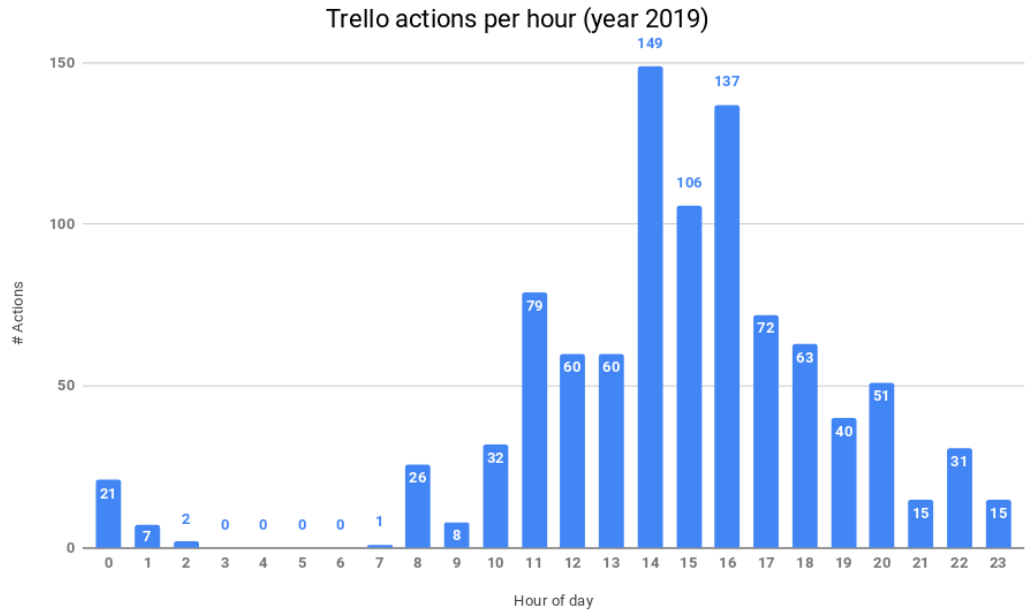


Figure 7.10: Distribution of Trello actions by hours on the year 2019.

The distribution of actions during the hours of the day shown in Figure 7.10 seemed very similar to previous years, with an earlier start with activity from 8AM but also a

decline by 9PM, as in 2018. Some activity was also shown up until midnight like other years. This year the peaks of activity happened at 2PM and 4PM, similar to 3PM and 5PM from the previous year.

Looking at the board names and their action count in Figure 7.11, we found that the *Software Engineering* board and the *RoboCup Tests* board presented the most work. In the previous year there was a board for the robot skill development itself. This year everything was stored in the *Software Engineering* board (and the board was actually copied from the previous year to keep the pending work and knowledge available). Moreover, this year the focus was put in having a robust architecture and robust scoring in RoboCup tests. Subsequently, there was a *Marketing* board with a significant amount of activity. Finally, the previous year's interns found fun in creating videos with Pepper to advertise the team and worked on that significantly in times where they lacked guidance.

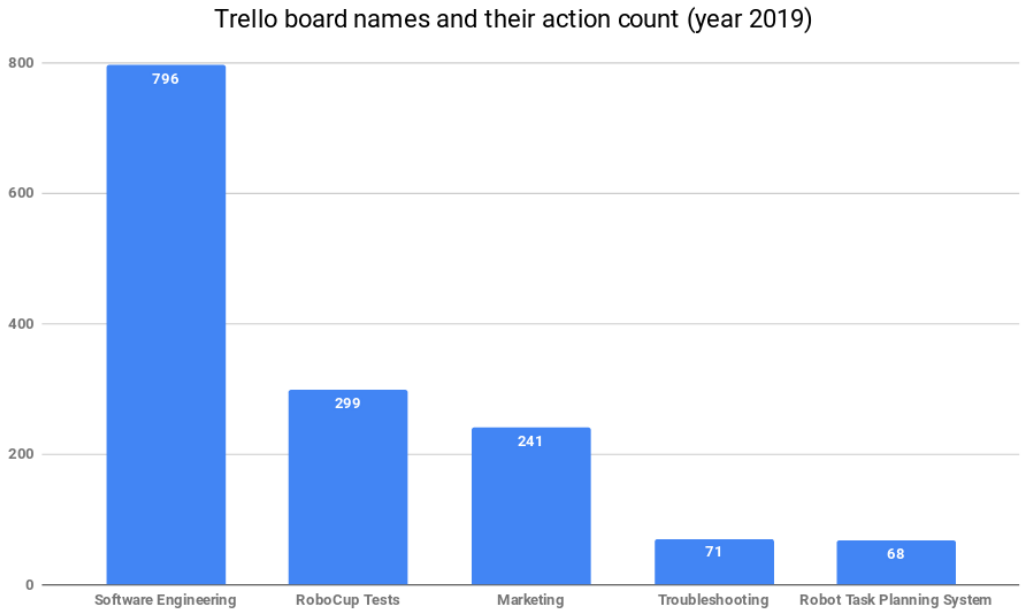


Figure 7.11: Distribution of Trello actions by Trello board on the year 2019.

Finally, checking the distribution of actions by anonymized authors with Figure 7.12, we saw a continuously decreasing amount of actions. This was different from 2018 where one author had most actions and there was a gap to 4 authors with a similar amount, and then another gap to the rest of the authors with lower contributions. The usage of Trello had a different activity profile from other years in this context. No explanation was found in that regard.

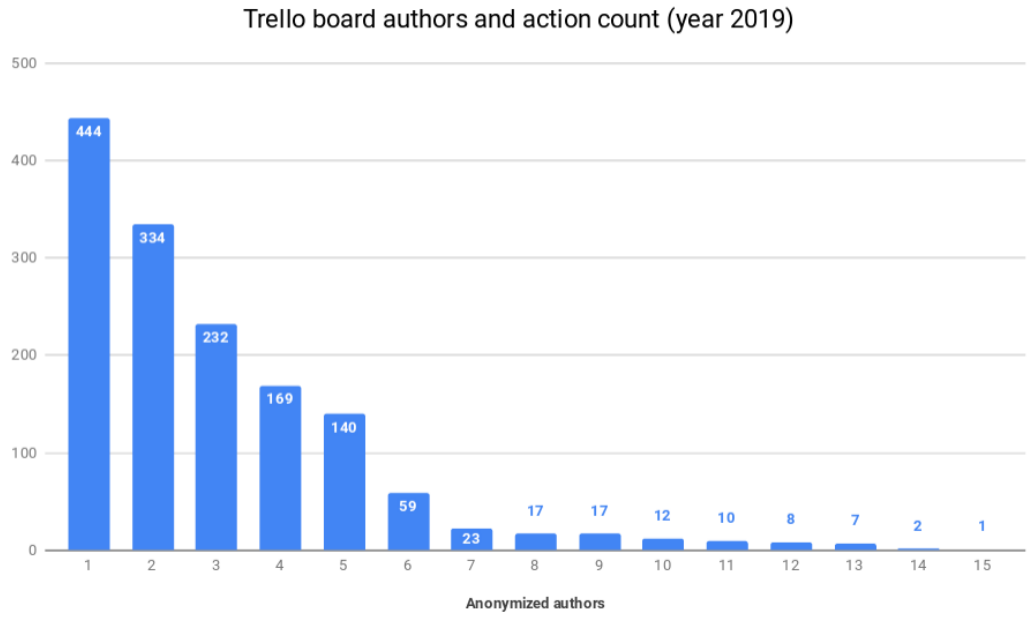


Figure 7.12: Distribution of Trello actions by anonymized authors on the year 2019.

7.6.1.2 Git Commit Data

The commit activity per day during the year 2019 as seen in Figure 7.13 showed that work started early in October and continued happening all year long with spikes in late December to early January and surrounding February. Moreover, from the start of June to the competition in the 2nd to 8th of July, work increased significantly. This was similar to previous years where the team pushed hard when the competition was near.

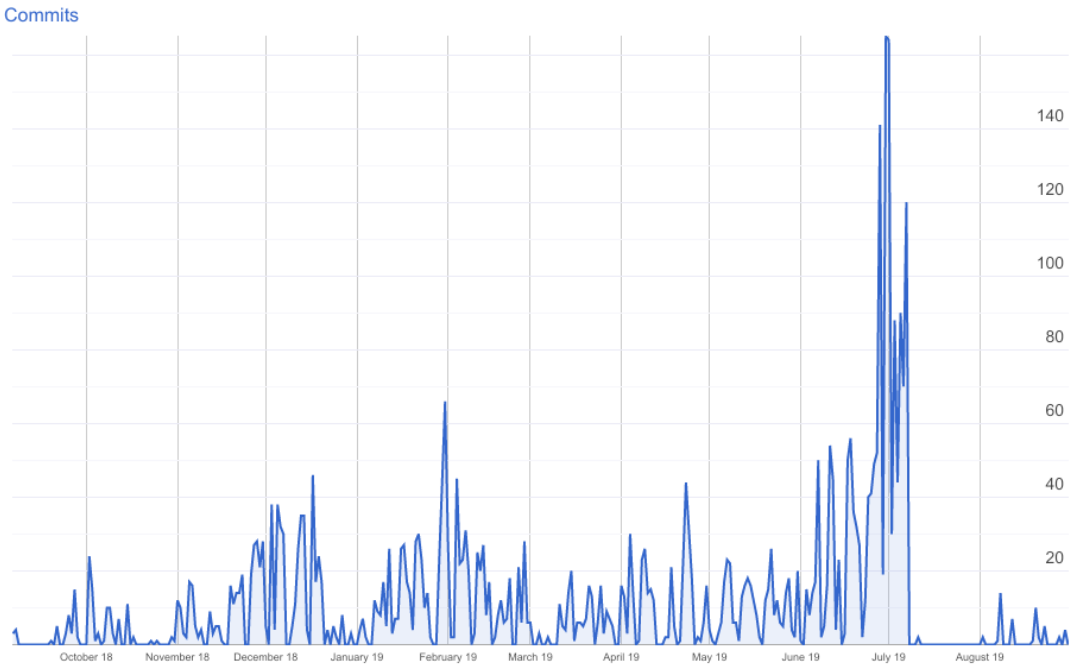


Figure 7.13: Activity (commits per day) on the year 2019.

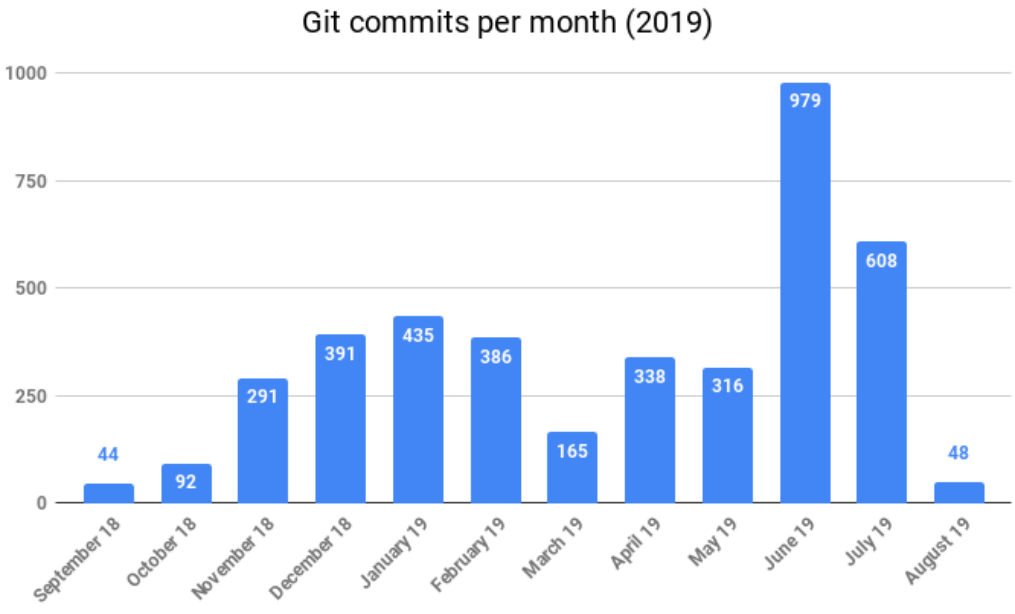


Figure 7.14: Commits per month on the year 2019.

Taking a look at the commits per month in Figure 7.14, it was noticeable how around January there was a steady amount of work, a big drop in March (when the lead developer

was absent) to come back to usual levels of work from April. Moreover, in June and July a lot of activity happened as in other years.

Checking the commits per weekday in Figure 7.15, a very similar amount of commits happened every work day of the week this year. No significant difference was to be seen by the team meetings happening mostly on Tuesdays, and the day encouraged to work in the lab being Thursday. A considerable amount of commits happened on the weekend but in a similar proportion to the previous years.

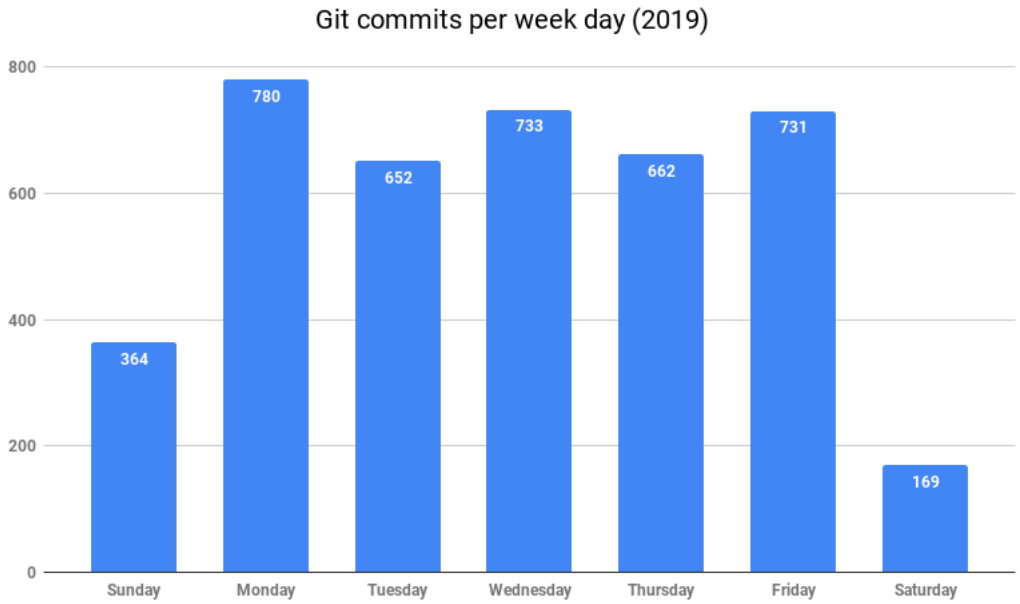


Figure 7.15: Commits per week day on the year 2019.

In the distribution of commits per hour of the day in Figure 7.16, a similar shape to 2018 was indicated with a slightly more evenly distributed amount of commits. The days started by 8AM. A peak of commits was found by 1PM (at lunch time) and around 4-6PM at leaving time, but extending until 9PM as other years, with some additional activity up until midnight. It could be interpreted that there was a flatter distribution of commits during all day.

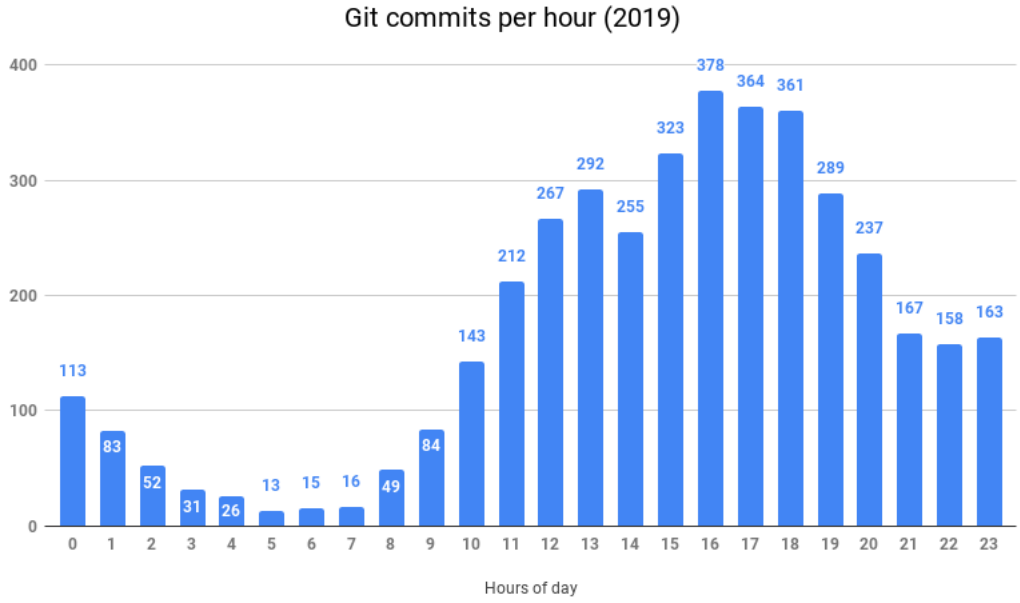


Figure 7.16: Commits per hour on the year 2019.

On checking the repositories for collaboration using Figure 7.17 it was found, as in 2018, 41 collaborative repositories. From this chart the following insights were obtained:

- The focus for the year was to improve substantially the navigation system and to create a strong base architecture for the codebase. The team demonstrated teamwork, with the repositories *magic_navigation* and *magic_ros* having most of the activity and a fair number of authors, with 4 and 6 respectively. Other repositories were also related to the architectural work by providing drivers and skills based on *magic_ros*: *magic_ros_apps, analytics_tracking, dark_magic, pepper_diagnostics, magic_pepper_bringup, magic_transformer, diagnostics_web_interface, magic_msgs, magic_dashboard, magic_stylesheets_pepper, magic_metrics, magic_launcher, linear_depth_scaler.*

- Significant activity by many authors could be observed in repositories related to developing skills for the robot: *magic_listen, magic_poses, magic_faces, magic_vision, pepper_skills, magic_speak.*

- Further significant activity showcasing collaboration on RoboCup test implementations: *find_my_mate, clean_up, receptionist, take_out_the_garbage, hand_me_that, find_my_disk, where_is_this, restaurant_2019, robot_inspection, finals_2019*.

- *magic_hri* and *magic_tablet* kept showing the compromise of the team to invest in good user experience.
- The team kept improving the documentation wiki *docs.wiki* as in previous years.

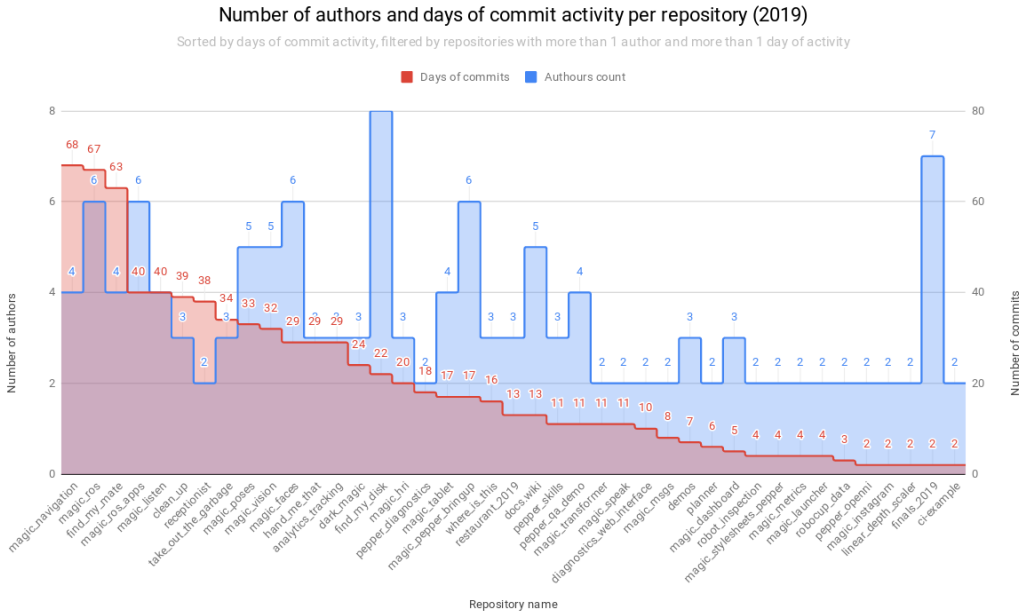


Figure 7.17: Number of authors and days with commit activity per repository, sorted by days with commit activity and filtered by more than 1 author and more than 1 activity day.

In regards of the repositories that did not show collaboration, as can be seen in Figure 7.18, their quantity stayed roughly the same as the previous year. The insights from the solo work projects were:

- Work on the building and deploying of the base system kept being the work of a single expert in the team. The repositories *gentoo_prefix_ci, gentoo_prefix_ci_32b, ros_overlay_on_gentoo_prefix, ros_overlay_on_gentoo_prefix_32b* and *pepper_os* were developed opensource to get help from experts from the Gentoo, ROS and RoboCup community with different degrees of success. A private repository *magiclab_pepper_os* contained the specific implementation and optimizations for the robots of the team. Taking into account that six repositories in this plot belonged to the same project by the same person, it was interpreted that this year, overall, there was less solo work.

- Experiments, as in other years, had single author repositories.
- Highly specialized projects also had single authors, as in other years.
- Some repositories from abandoned work from the previous year had some work.

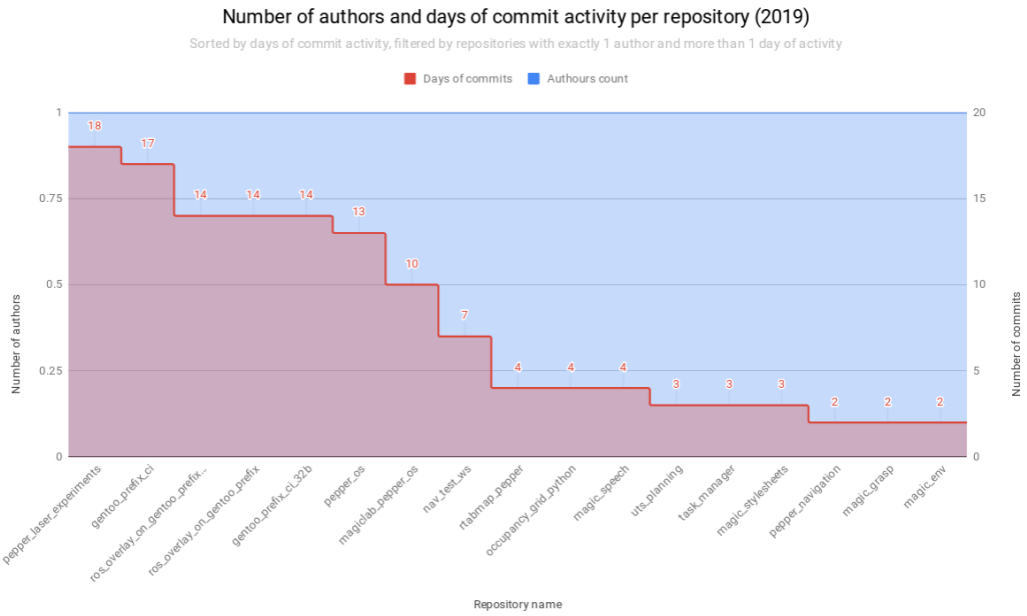


Figure 7.18: Number of authors and days with commit activity per repository, sorted by days with commit activity and filtered by exactly 1 author and more than 1 activity day.

2019's average of authors in repositories with more than 1 author and more than 1 day of commit activity came up to 2.68 authors per collaborative repository. This improved even more than in 2018 which had an average of 2.55 authors per collaborative repository. Taking also into account that the amount of solo projects decreased, we saw an even larger increase in team work in this year compared to the previous year's increase.

7.6.2 Team Retrospective

In this section the team's retrospective will be summarized. The original retrospective document for the year 2019 can be found in the section B.3 from the Appendix B.

The first question of the retrospective was "What worked well?" about the positive facts from the preparation process and the competition itself. Regarding the competition event itself, the team reported "amazing" teamwork and successful "last-minute hacking",

they also noted that having a nearby hotel to allow team members that lived far from the venue[9] to sleep nearby helped them. Additionally, having the laboratory nearby allowed the team to have a place where they could code and test in times when there was no access to the competition venue. Furthermore, the team reported great robustness in tests, highlighting the Stage I performance, with special mentions to the following systems: the usage of OpenPose [111], magic_ros providing seamless and efficient external computing, the unmodified tablet interface from the previous year, the Red Green Blue + Depth (RGBD) approach to mapping and localization with the RTAB-Map [112] framework, the speech recognition system with exceptional performance working with open grammars, the face recognition approach using face vectors, and the object recognition implementation using YOLO [110]. Concerning the development process, multiple comments about the ORTs were provided: the provided free lunch in these events "helped bring people together", having regular ORTs with realistic scoring was beneficial, and ORTs as the backbone of development was a successful approach. Additionally, the lab setup was as hard or harder than the real arena in the competition, including items like metallic shiny chairs and tables hard for the robot to sense were also regarded as positive. Moreover, the team mentioned as beneficial the reuse of the tablet interface from the previous year. They complemented the "right level of granularity" of the team's packages in regard to the distribution and isolation of the functionality of navigation, speech recognition and generation, and tablet interface. Finally, working in pairs and pair programming to work on RoboCup tests and to share knowledge about specific functionalities were regarded as positive practices.

The next question was "What did not work well?" about the negative facts in the same context. Some reports were conflicting with the previous question. In regard to issues in the competition event, the team reported that robot navigation around a black table with metallic legs was problematic as it was invisible to Pepper's sensors. Also, trade-offs between navigating safely and at speed, and between the speed and precision to arrive at navigation goals were criticized. Additionally, the Human Robot Interaction (HRI) approaches for every RoboCup test were different, particularly the robot's awareness with regard to people approaching it. With regard to the object recognition system, the training was reported as very time consuming and stressful as it was ready only very closely to when it was needed in the competition, and it was sensitive to context changes such as backgrounds and illumination. Moreover, Stage II tests were less robust due

[9] In 2019 the competition happened in Sydney, Australia, close by to University of Technology Sydney (UTS) where UTS Unleashed! was based.

to their late development. Additionally, setting up external computing devices to run remote nodes was perceived as stressful and risky - tooling could have improved the situation. Finally, a team member reported that they believe that the team should not leave early from the venue, or go to sleep the day before a RoboCup test is to be run if it is not ready. Related to this topic, sleep management was reported as an interesting topic. The trade-off between the team members wanting to show commitment to the team, and needing personal sleep to be sufficiently well rested for upcoming challenges, manifested in new team members, unfamiliar with the fatigue of the event, struggling to manage their energy well, to the detriment of their performance in the later RoboCup tests. On the other hand, regarding the development process, the navigation system was reported as left to be solved too late in the development process. This affected the implementation logic of RoboCup tests, triggering late changes which increased the testing workload. The object recognition system had the image processing work for training datasets implemented late, and was also missing tools to evaluate the trained models. Additionally, regret regarding not exploring other object recognition approaches besides YOLO was reported. Moreover, feedback about ROS adding complexity and increasing the entry barrier to contribute to the team was reported. In regards to team management, it was reported that a period lacking leadership almost froze development. This had an effect on some team members wanting to build up skills in different areas but, as development time was lost, they fell back into working in areas they were already comfortable with. Different working styles made collaboration sometimes challenging. Knowledge transfer from experienced team members towards new team members could be improved, and sharing in-depth knowledge about specific systems was not achieved. Language barriers slowed down communication in some cases, and some team members needed more guidance and their time was perceived as wasted when guidance was not available. Furthermore, Stage II tests involved elements of research combined with elements of implementation. The research aspects were not challenged and tested early or regularly enough, showcasing the need for a different approach for such topics. Also, ORTs not being adequate, was suggested. Finally, it was reported that the architecture evolved from a bottom-up process with a lack of a top-down vision; there was wasted time in subsystems that did not have direct relationship with scoring; there were few unit tests, and having more could have saved time in regards of bugs and regressions, but no setup was available to make this practice easy.

Previous retrospectives included the question “What should we do next?” and, unfortunately, because this was the last year of participation the team members did not fill

this section even though it was part of the retrospective document.

7.7 Reflection

The reflection is composed of the findings, which include the same substructure of three sections as explained in chapter 4, answers to the action research questions stated in this cycle, new questions to start the next cycle, and next steps to be implemented.

7.7.1 Findings

The findings stem from a review of the action research cycle and the team's retrospective. They aim to describe facts learned in this cycle.

7.7.1.1 Team Management Processes

By this point, being the last year of this project, the processes that the team used that stayed in some way were believed by the lead developer to be successful. But these processes were tailored to the needs of the team and the lead developer, specifically.

For UTS Unleashed!, being flexible with the preferences of its team members was a key element for good performance. Noteworthy, this was the view of the lead developer with this specific team. In order to generalize this view, further testing with different teams would be needed.

The topic of leadership and its importance came up this year. The lead developer was unavailable for almost 3 months and the leader role was missing. The team tried to find a student to take this role but this was unsuccessful. The lab co-director assumed this role temporarily since no student volunteered. However, the lead developer came back with enough time to move the project forward until the competition started. If this had not been the case, it would be hard to assess how this situation could have been better managed. The lead developer was of the opinion that it was managed as best as possible given the circumstances.

7.7.1.2 Team Software Development Processes

For the UTS Unleashed! specific case, it was more important to appeal to the profile of the different team members than enforcing software development methodologies and practices. These methodologies contain processes and tools that are believed, and in

some cases proven[10], to provide benefits in the industry. However, this project followed a different trajectory as it adapted to the feedback of the team. The motivation of each team member was different and not related to any kind of payment (even though some interns were actually paid), which context, standard software development methodologies tend to have. The agile philosophy was maintained as in the previous cycle, with emphasis on the well being of the team while still delivering working software.

7.7.1.3 Technical Approaches

This year's technical approaches seemed to be fruitful. The parts where the lead developer was involved (the base OS, the magic_ros base framework and the navigation system) saw huge improvements. Other subsystems also improved dramatically, like the Automatic Speech Recognition (ASR) system.

The opinion of the lead developer was that having a clear architecture view, and having it well implemented helped in this direction. This opinion was based not only on the technical benefits that these things provided, but also on the morale of the team by having an easy to understand architecture which approached the gaps seen in previous years.

7.7.2 Answers to AR Cycle Questions

This year the system was architected using a base framework called magic_ros to enable transparent remote nodes running at the same time as local nodes in parallel with many other features. This approach was successful. This framework offered many elements to make development easier for teammates and minimize code duplication, especially around ROS related components. Having their own drivers for the navigation part, and their own navigation state machine, gave the team a lot of flexibility. Moving to visual based localization also performed well. This framework helped reusability, at least in the parts where the lead developer was involved, in his opinion. However, the same feeling may be shared from other teammates that developed their own frameworks for their RoboCup tests. Different opinions on the setup of other subteams existed, but every subteam worked well in their own setup. Embracing the ROS distributed approach but taking care of the user's experience when using these distributed services, provided fast integration of capabilities. Capabilities that were designed to be used by all the

[10]There is a large body of literature about software development methodologies, practices and processes in different industries. Different keywords in academic search engines related to these topics bring up million of results.

team were aimed to be easy to install with minimal or no dependencies, usually just magic_ros. They were also designed with minimal setup as they were already running in the robot as a service, offering a simple interface for mainstream use. Additionally, these capabilities offered a complete interface for advanced usage, returning structured and complete information, and with descriptive error messages. A lengthy manual analysis of the available git data could be done as future work to better understand which specific parts of the code were reused the most in this project and may provide further insights.

Allowing team members to develop their systems in the way they thought best was positive too, especially while keeping attention on making them available via the magic_ros framework when appropriate. This year further dividing in subteams was embraced, especially in pairs for RoboCup tests. It worked very well given the team's feedback, the git statistics, and the competition results, showcasing that teamwork increased, making it a model to keep for the future. The lead developer believed that it was related to the team composition, as possibly with people with more independent and proactive profiles this would be either not necessary or not beneficial. Every subteam worked comfortably in their own way which allowed them to focus on scoring in their tests. They had their own interpretation of what worked "best". From the lead developer perspective, learning from each approach was beneficial. Not a very conclusive answer as to how to implement these, but engaging in conversation with each subteam to understand their needs and why they used such an approach was done. Sometimes an approach could be wrapped in multiple interfaces. Examples of these were:

- Embody the robot in a class and access its capabilities by topics. For example "robot.navigation.move(x, y, theta)".
- Create capabilities as simple functions to import. E.g. "show_in_tablet(something)".
- Create centralized services which are called by clients to do work.
  E.g. "object_detection_client.find_objects()".
- Programming by configuration by using graphical clients.

This year there was a stronger focus on scoring. This was because the rulebook made it harder than ever to score, and more team members believed in the importance of scoring more than 0 to advance in the competition. The interns were in their second year with the team, so they saw first-hand how important this was. The new team members were introduced into the team via a summer course where they were implementing

RoboCup-like tests where the scoring was very important which also helped the team to be in the same mindset.

By the end of the development year the workload increased significantly as in previous years. Perhaps this project has to take into account that the workload becomes higher in the time close to the competition. A set of factors played into this phenomenon, as not having a final rulebook until a few months before the competition, in the case of RoboCup@Home, and new team members being required to learn the team's technologies and workflow, with different amounts of time required for every individual.

Finally, for this year the concept of proactivity was to be researched. The lead developer experimented with measuring proactivity trialing psychological profile tests. However, these showed that team members that had the potential to have high proactivity, just were not being proactive. Given this initial testing was not fruitful, no further work was done in this direction. Future work in the workshop and/or survey was planned at this point.

7.7.3 New Questions

Given there wouldn't be any further action research cycles in this project, new questions could only be explored via other means. After the participation in the 2019 RoboCup, a workshop with experts was held. From this, a survey for RoboCup experts was developed to further gather insights.

The ORTs this year were strict on scoring. A lot of 0 scoring happened and this affected morale, for two months progress was perceived as stalled. There could have been approaches to improve this situation. New approaches from other teams could help and should be researched further.

It was mentioned in different parts of the previous and this action research cycle the fact that, regarding team management and coding practices, adapting to the profile and preferences of team members was important. The lead developer was of the opinion that this led to success. However, other experts could have a different opinion and believe that adhering to standard software development practices and procedures could be a better approach.

Proactivity was a topic that arose this year. The lead developer believed that it was a positive behavior to have in team members, or at least in some of them, specially if they were to lead some development. Not enough research was done this year, so it became a topic to explore further. Furthermore, some team members this year were disengaged

from the project. A question about how to keep them engaged arose. It is unclear if this situation was related to the topics of proactivity and motivation.

7.7.4 Next Steps

Given this was the last year of the project, this section is discussed in future work in chapter 9. As a brief introduction, further work on increasing the research output of the team should be done. Improving the understanding on what creates teamwork and if it is related to the concepts of motivation and proactivity would be of interest. Enrolling another team, or multiple teams, in to following action research cycles as the ones shown in this work would add new points of view.

7.8 Possible Guidelines

Based on the previous guidelines, the most important findings and reflections from this year will modify or add to the guidelines. These guidelines will be discussed further in chapter 9 as they will be compared with the outcome guidelines from chapter 8.

- Perform end-to-end testing. Simulation of the full competition including: setup on a previously unknown place, strict schedule and timing, strict scoring, by a strict referee, naïve users and other elements that otherwise may be overseen. Examples of other elements can be: networking issues, environmental noise, unavailable team members or unusual lighting. Start performing these as early as possible, team members new to the competition will grasp a better idea of what the competition looks like. Allow flexibility when necessary, e.g. if some part of the system is absolutely not ready, allow it to be bypassed.

- Find a balance between what the team wants and what is believed to be the best for the team regarding development processes. This is something to be re-evaluated during the development year. Topics such as coding standards, testing standards, documentation, deployment strategies and code sharing approaches fall into this category. Invest in understanding what is considered hard and work on improving the situation. Example improvements can be finding alternatives or providing further documentation or tooling.

- Provide the best tools available for the job. Development tools, frameworks, simulations and hardware are topics that fall into this category. References for these

can be taken from other teams or industry. Provide documentation and training for these too. The example here is the ROS middleware, widely used in the community of the competition and in industry for robotics applications.

- Ensure teamwork is possible and encouraged. Set up a backlog of tasks and distribute them taking into account every team member's interests. Promote grouping team members to work more effectively and share responsibility. Promote pair programming for knowledge transfer and to fix difficult bugs. Consider using a physical backlog, additionally to online means, in your working environment as a clear and quick way to visualize progress.

- Test as often as possible, on the real robots and in a realistic environment. Having unit testing is a great tool, testing with stored real data is also helpful but the only thing that will matter in the end is that the competition tests run robustly.

- Focus on scoring. Implement and test with that in mind. Over-engineering a robot skill for cases that will never happen implies that time could have been invested somewhere else of higher priority.

- Start with simple approaches and iterate improving them. This can be applied for both the development of capabilities for the robot and for the design of approaches for competition tests.

- Architect your system with software integration in mind. Integrating a lot of software from different sources with different mindsets can be hard and it should not be left as a final exercise.

- As processes become stricter (testing, coding practices, task progress tracking, etc) and the competition approaches, attention must be kept on the morale of the team members. E.g. stressful meetings may include snacks, free meals or a follow-up leisure activity. Having a designated day of the week agreed to work together in the lab can improve morale by being present on the progress of other team members and it opens more opportunities to collaborate.

CHAPTER 8

EXPERTS INSIGHTS AND VALIDATION

The previous chapters tell the story of the author's team in the RoboCup@Home Social Standard Platform League (SSPL) competition. Conclusions have been drawn from this project within its own context. These conclusions can't be generalized by themselves. To validate insights, find new ones and potentially find conflicting topics, the feedback from others is needed.

The most relevant feedback can be obtained from experts in the same field or community, in this case, participants of RoboCup competitions. After a set of informal chats during the competitions with other RoboCup teams, a workshop was held to further gather topics of interest and share approaches. From the information acquired in this workshop, a survey was created and distributed to members of the RoboCup community.

In this chapter the workshop will be presented first, followed by the analysis of the survey.

8.1 Grounded Theory Remarks

The concepts from Grounded Theory will be applied in the following manner.

- **Simultaneous involvement in data collection and analysis phases of research.**

  The workshop implies the usage of this characteristic of Grounded Theory. Given the nature of the workshop where the participants will be openly discussing topics

as they appear, their opinions become data and the discussion in itself is part of the analysis.

- **Creation of analytic codes and categories developed from data, not from preconceived hypotheses.**

  The survey presents this practice often. On questions where the response is an open text field, the methodology to analyze the answers implies a process where keywords need to be extracted. These keywords form codes which are organized in categories. These categories are developed into theories that become the summary of the responses. Furthermore, from the summary of the responses further abstractions into theories will be created.

  Other questions follow a different process to create these features - for example, questions where a respondent needs to sort concepts by importance. However, even these questions still result in the creation of codes, categories and theories.

- **The development of middle-range theories to explain behaviour and processes.**

  In the previous point it is mentioned that summaries from responses are created. These act as short lived middle-range theories. When thinking about the final conclusions of the survey, all the summaries and theories from the questions in the survey act as middle-range theories.

- **Memo-making, that is, writing analytic notes to explicate and fill out categories, the crucial intermediate step between coding data and writing first drafts of papers.**

  Memo-making is used throughout both the workshop and the survey. In the workshop, notes are taken from the different points participants raise. These contain opinions, facts and theories to be analyzed and discussed further on.

  During the analysis of the survey, the process where keywords, codes and categories are developed, fits the description of memo-making.

- **Theoretical sampling, that is, sampling for theory construction, not for representativeness of a given population, to check and refine the analyst's emerging conceptual categories.**

  Both during the workshop and the survey, the aim is to find theories first. Analysis of the population is done when the profile of the participants is requested via

different questions, but this doesn't guide the research. This data is used later on to check the quality, validity and applicability of theories.

- **Delay of the literature review.**

  As theories appear from the analysis of the workshop and the survey, literature review is performed to find and take into account previous relevant work.

Furthermore, an example of how the question Q6.2-Q6.3 from the survey in section 8.3 was analyzed can be found in Appendix C.

8.2 Expert Workshop

A full day workshop titled *Software Development Methodologies for Robotics Competitions and Challenges Workshop* was organized for the 9th of July of 2019, the day after the RoboCup Symposium, as part of an effort to gather insights from other experts in robotics competitions.

The idea to hold this workshop came from informal conversations during the previous years within the UTS Unleashed! team and other teams during the competitions. The workshop was held to build up evidence from multiple sources about the process of preparing for a RoboCup competition and competing in it as a team.

The topics the workshop covers stem from the ideas from the Action Research (AR) cycles held previously.

8.2.1 Overview

10-20 experts in robotics competitions gathered at University of Technology Sydney (UTS) to explore and discuss research issues related to Software Development Methodologies for Robotics Competitions and Challenges. This one day workshop aimed to discuss topics around the software development process for robotics competitions like:

- Meetings and communication
- Planning and decision making
- Team structure and management
- Coding practices
- Challenges

The goals of the workshop were:

- Gather new insights to improve Software Development Methodologies for Robotics Competitions based on experience engaging in robotics competitions and challenges.
- Create a new research agenda for Software Development Methodologies for Robotics Competitions.

Participants were invited via the following means:

- Contact via email explaining what the workshop was about months before the event.
- Contact in person at the RoboCup event explaining what the workshop was about.

The participants were selected to be mostly very seasoned members of the RoboCup community.

Initially, twelve participants were expected, including 3 members of the UTS Unleashed! team. Three participants ended up cancelling, but three additional participants were found and also two more team members from UTS Unleashed! joined. The final number of participants was fourteen, representing ten different teams.

8.2.2 Structure

The workshop was designed to be an open environment for the participants to discuss the matters proposed in the overview. The program for the day can be seen in Table 8.1.

Time	Activity
8:45	Welcome
9:00	Introduction to each other
10:00	UTS Unleashed! Approach and Discussion
12:30	Lunch
14:00	Group Questions & Answers
17:00	Workshop closing

Table 8.1: Program schedule for the Software Development Methodologies for Robotics Competitions and Challenges Workshop.

A simple presentation to guide the workshop was prepared. The points to discuss came from the previous work during the action research cycles in 2017, 2018 and, partially, 2019. The presentation had the following structure:

- Motivation
  - Performing well in a Robotics Challenge is hard
  - Extract consensus on good and bad practices
  - Obtain feedback from experts
- Plan for the day
  - Introduction to each other
  - UTS Unleashed! approach with discussion
    - Project Timeline
    - Recruiting
    - Planning & Decision making
    - Meetings & Communication
    - Team expertise
    - Coding
      - Practices
      - Tools
    - Training & Learning
    - Competition testing
  - Group questions & answers

8.2.3 Workshop Outcomes

The workshop outcomes are divided between understanding the profile of the participants and gathering insights from them. These insights are presented as ideas or questions that were answered in consensus, and questions or concepts that didn't have a decisive answer. These last ones are to be further researched by the survey found in the following section.

8.2.3.1 Participants profile

A set of questions were asked to the participants to gather some information about their profile. Their name and affiliation has been kept anonymous. Multiple team members from UTS Unleashed! attended the workshop but only one answer pertains to the UTS Unleashed! team in the data presented. This way all answers are about one participant in reference to one team unless noted otherwise.

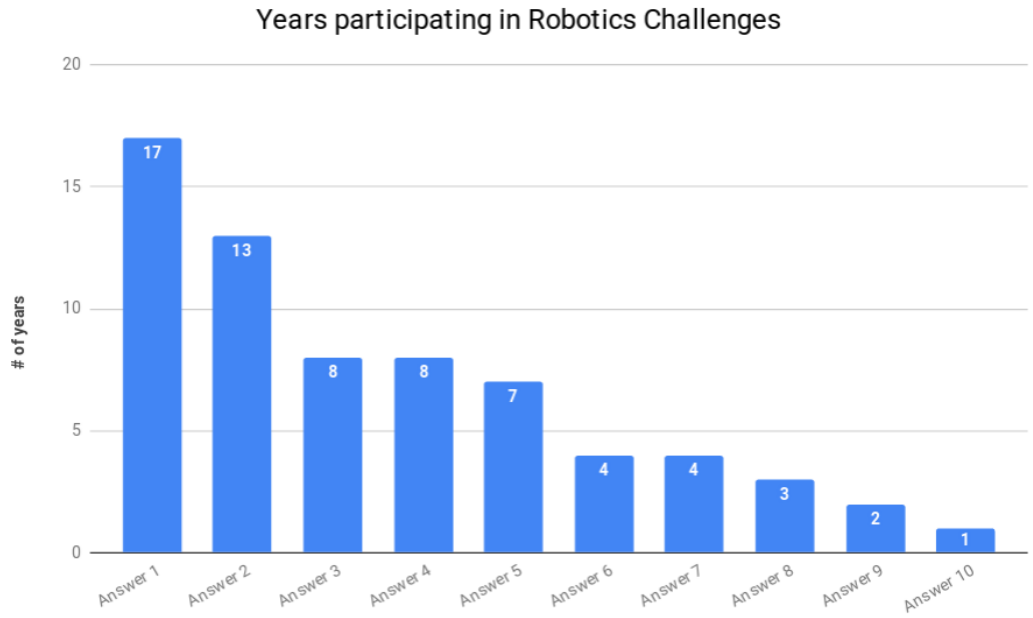


Figure 8.1: Minimum years of experience in Robotics Challenges reported by the workshop participants was 1 year. Maximum was 17 years. With an average of 6.7 years and a median of 5.5 years of experience. 67 years of accumulated experience.

Participants were asked how many years of experience they had competing in Robotics Challenges (not only RoboCup). Their answers are summarized in Figure 8.1. We can observe a wide range of years of experience with two participants having 17 and 13 years of experience and one participant having only one year of experience. 80% of participants had three years or more of experience. This should be a positive factor to get meaningful insights from the workshop.

Participants reported competing in the following RoboCup leagues: @Home, @Rescue, Soccer mid-size, 2D Simulation, Standard Platform League and Logistics. Participants also reported taking part in the following non-RoboCup challenges: DARPA challenge 2005, NASA Space Robotics Challenge, RockIN and “other”.

It can be generalized that the participants have a lot of experience in the domain of Robotics Challenges or competitions, specially on RoboCup.

It is worth mentioning that only 1 participant never participated in the RoboCup@Home league meanwhile all the rest had participated in it.

Another topic discussed was how did the participants approach the project of competing in Robotics Challenges. Everyone approached RoboCup as a continuous project. Comments over this question included: “The project is planned every year but with the aim of continuity.”, “It’s approached as a continuous project using university subjects.”, “It’s approached as a continuous project within a research program.” and “You need to approach it as a continuous project to get good results.”.

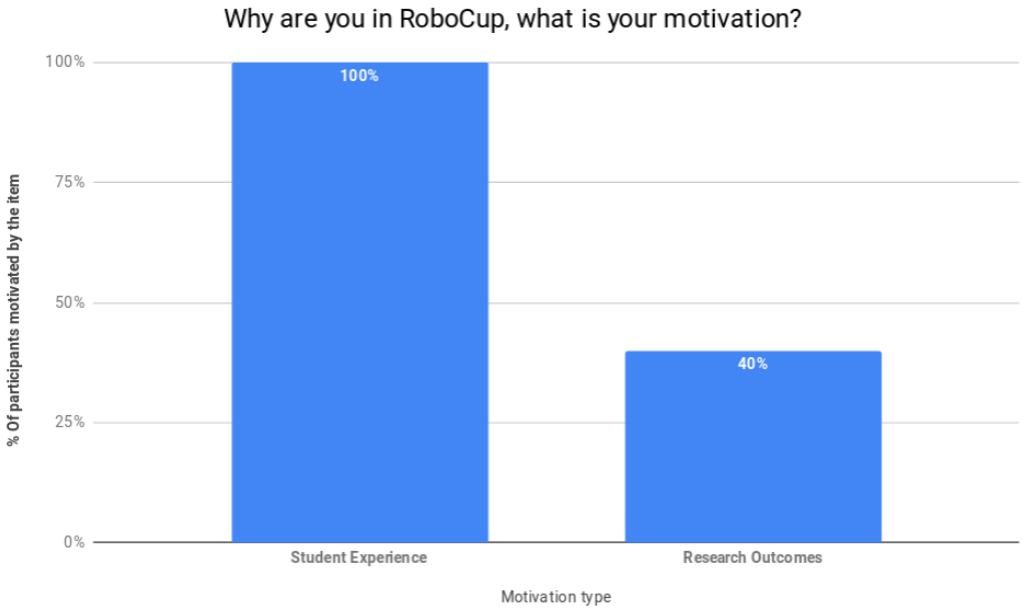


Figure 8.2: All workshop participants reported that Student Experience was one of their main motivations to participate in RoboCup. 40% reported Research Outcomes to be another main motivation.

All participants were RoboCup participants. When asked what their main motivations to participate in RoboCup were, their answers have been summarized in the chart in Figure 8.2.

Other reported motivations were: being part of a robotics club (with RoboCup being an activity of the club), being able to create robot applications, having access to robotic platforms, having fun, and learning practical skills.

Participants were asked about their approach to the competition in regards to being more pragmatic, i.e. aiming to get good results in the competition, or using RoboCup as a research platform, i.e. aiming to get publications. The summary of their responses can be found in the pie chart in Figure 8.3. 40% answered pragmatic approach, 40% answered research approach, and 20% answered both. Thus not having a majority indicating different points of view. Further comments about the research approach will be found later on in this section.

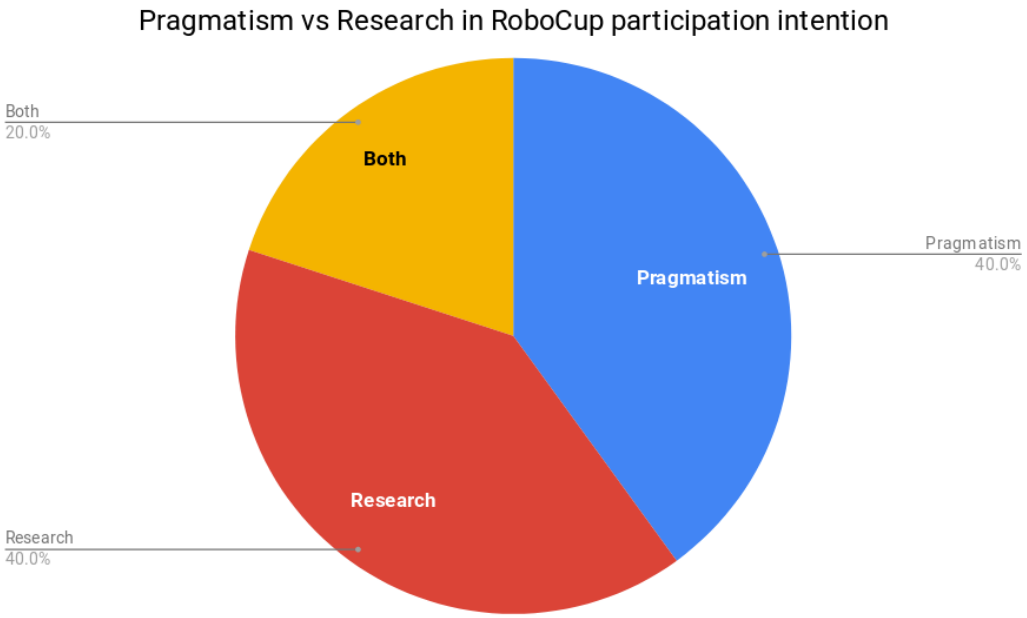


Figure 8.3: When asked about their approach on competing on RoboCup being more in the pragmatic or research line the workshop participants didn't have a clear leaning.

Moreover, the participants were asked about how many team members did their team have that year, but only seven gave an exact answer (the nature of an open discussion made it so this could happen). The answers can be found in Figure 8.4. The team sizes ranged from 14 team members to 5 team members with an average team size of 8.5 and a median team size of 7 team members.

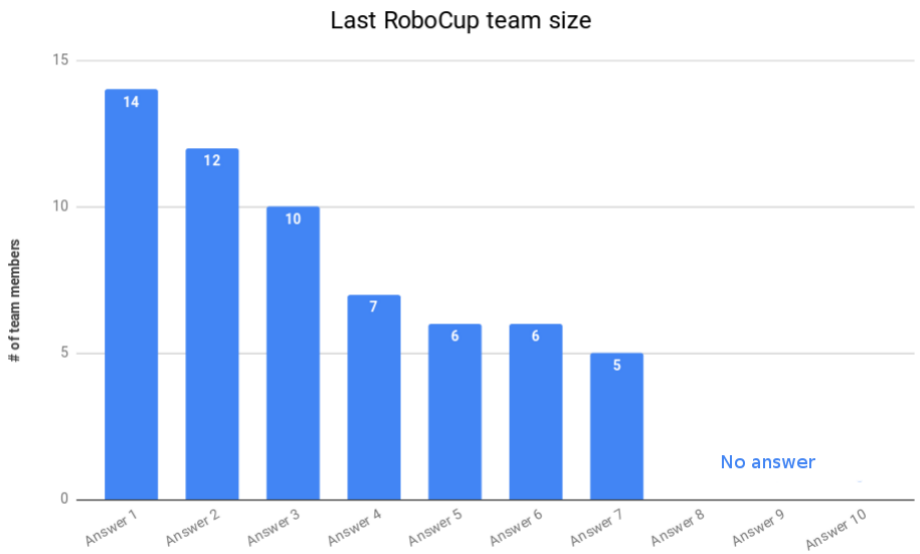


Figure 8.4: Workshop participants team size reported for the last RoboCup event. Three participants didn't answer this question. The largest team reported had 14 members and the minimum team size was 5. Average team size was 8.5 and median team size was 7 team members.

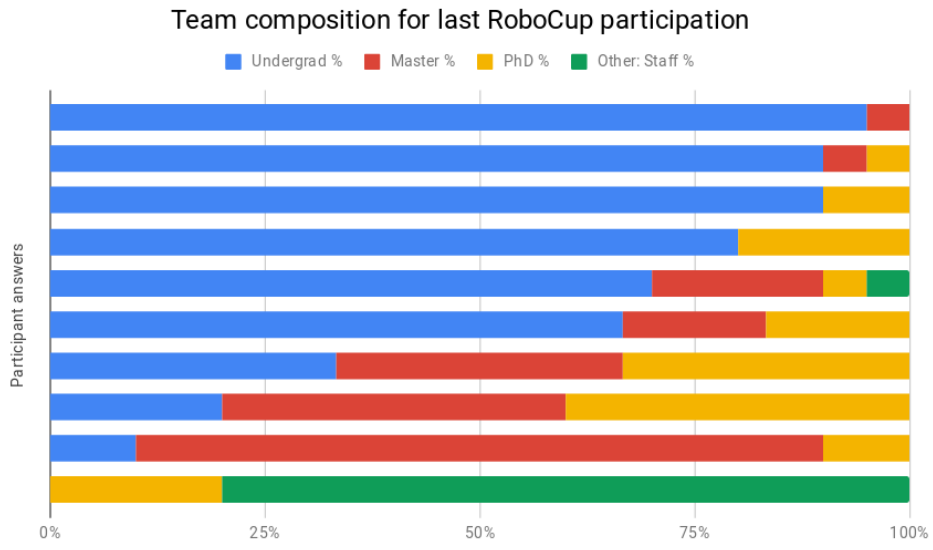


Figure 8.5: The team composition for the last RoboCup participation of the workshop participants. Every row represents the answer of a participant and each color represents the type of team members, divided by students of undergrad, masters or PhD, and another category where university staff was reported. There does not seem to be a consensus even though we see a majority of undergrad students overall.

Another question was “What was your team composition like for the last RoboCup participation?” with the answers being composed of undergrad students, masters students, PhD students, and others. Others were mostly considered as university staff. From the answers the chart in Figure 8.5 was created. No consensus on the team composition is to be observed. A majority of undergrad students can be appreciated but also one team had just PhD students and university staff.

When the participants were asked about what their optimal team size would be, the answers were both given in specific numbers and ranges. This is shown in the chart in Figure 8.6. The largest optimal team size was reported to be in between 15 and 20 people meanwhile the smallest was a specific answer of 5 team members. A wide variety of opinions were held.

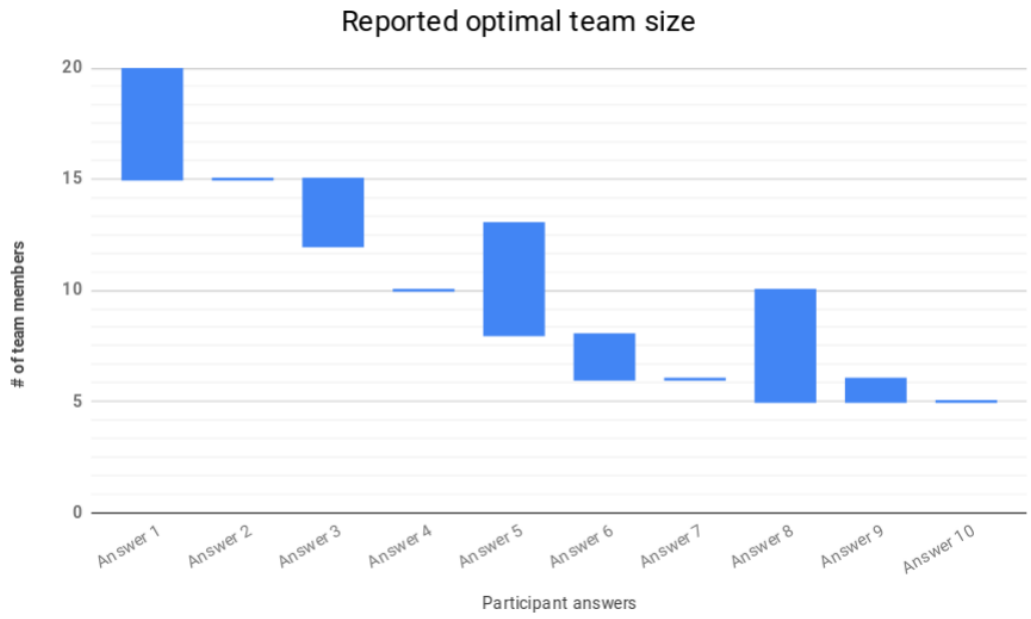


Figure 8.6: Optimal team size reported by the workshop participants. Some participants answered in specific numbers while others answered in ranges.

Further comments about the team composition, team size, and team goals were expressed. These are summarized here:

- Depending on the composition of the team, team size will differ. Having more PhDs over masters, and more masters over undergrads, was perceived as a reason to require less team members.

- Depending on the goals of the team, i.e. research or competition outcomes, the team size may differ. One participant reported that having a large team benefits research value.
- Optimal team size was reported similar to the team size of each workshop participant. This may point to the fact that every team adapts to their own context.
- One participant reported being unhappy with undergrad students as team members, because they may lack base knowledge. This participant recommended to first allow them to learn, and then join a team.
- How much effort is needed and how it is measured (person-hours, person-days, person-months?) remained unknown. Having a measure may be meaningful. One participant said that their belief was around 6 months for 1 person full time to master one skill. Other participants agreed that it depends on the field of expertise and where a team member starts from.
- One participant expressed that you need team members you can rely on. Another participant added that the members need to have accountability, they need to follow through people doing work, and they need to have work to do.
- Some team members agreed with the idea that "usually RoboCup team members are top quality students".

Most participants agreed (others didn't explicitly participate in the conversation, but no one denied such an argument) that "doing research is hard with RoboCup".

Most participants also agreed that "getting funding for RoboCup is hard". One participant noted that their argument to fund RoboCup was "RoboCup participation is research".

Other comments at this point of the workshop included:

- Some teams take part in RoboCup inside of a university environment but as part of a single laboratory project, not as part of a university founded initiative.
- Two teams participated in RoboCup as part of a robotics club.
- Multiple participants agreed that every student that got into RoboCup "got nice jobs in companies. RoboCup experience is valuable employable experience".
- Burnout came up as a common topic. Participants reported that it happened in every team at some point.

- Teams participate in the creation of the rulebooks, sometimes joining the technical committee to fix specific problems.
- RoboCup team members were reported to get access (via keys or passses) to the labs to work freely, and to a certain degree, use it as space for their daily life. Everyone agreed that having access to the robots was important. Two participants reported that even having a bed in the lab is beneficial.

The conversation went off-topic about the @Home league needing a fine balance in between pragmatism and research and how hard it is to unite both goals.

8.2.3.2 General Answered Questions

During the general discussion, just a few topics reached some kind of consensus. These are described here.

To the question “How important is experience in participating in RoboCup competitions?” there was a unanimous answer: very important, both in the specific competition itself and in the necessary fields of expertise. Fields of expertise were exemplified as robot navigation and manipulation.

Discussion about collaboration in between teams and sharing experience arose. A participant mentioned that some teams just copy other team’s approaches and code without contacting them. Another participant mentioned that some teams do ask about collaborating or using other team’s work.

There was a consensus in the fact that “there is a lack of standard software engineering practices”. The consensus went further stating that “software engineering practices were key but it is hard for everyone to learn all the important bits and tools”.

Another point of consensus was that “good communication is fundamental” with regard to a team’s internal communication.

Most participants also verbally agreed that practicing RoboCup tests in conditions as close to the real competition is a very important practice to achieve good results. One participant said that a professor acted as a “very harsh referee”. Another participant manifested that they “recreate real RoboCup competition days”.

8.2.3.3 Unanswered Questions

An activity was held where the workshop participants were proposed to write on a set of post-its, concepts, topics, or questions that they found hard or without a consensus. These are reproduced here from the image in Figure 8.7 for the interest of the reader. They

are sorted trying to keep similar themes next to each other. Comments by the author in parenthesis to make the context more clear have been added. From the discussions in the workshop and this set of concepts, a follow-up survey was created. The next section is a detailed report of that effort.

The list of concepts considered as unanswered questions follows:

Figure 8.7: Picture of the post-its on a whiteboard from the Software Development Methodologies for Robotics Challenges about unanswered or hard topics related to Robotics Challenges and RoboCup.

- Support and encourage diversity and inclusiveness in the teams.
- Automated testing and metrics are hard to implement and share.
- Stability (of the software systems).
- (How to implement, importance of) Fault tolerant systems
- Benchmarking (in the general sense of the word, benchmarking the competitions, the robot skills, the applications).
- (How to) Fair benchmarks to test skills in isolation but still aiming for the big picture
- Unit testing in Continuous Integration (CI) (How important is it)
- Testing (what kind of tests to do)
- Comparability (of team approaches and robot performances)
- Poor code-sharing (in between teammates)
- Allow students to graduate (as competitions work takes a lot of time and seems hard to get research output)
- Distraction from studies
- Large team to restart from one year to the next (how to avoid it, how to deal with it)
- How do we motivate students to work?
- How do you create accountability?
- (How to, is it beneficial) Create a sense of panic early in the process
- (How to, reported to be hard) Understanding everything as an undergrad team
- Making integration tests without proper simulation
- Getting everyone to use the same tooling
- Enforce standards for software design e.g. style, naming conventions, functional design

- Actually enforcing good coding practices (code style, testing, CI...)
- (How to get people with those skills or develop them in time) Solid programming and software engineering
- Recruiting and keeping experienced people
- Taking responsibility, no code ownership
- Simulating everything
- Debugging
- Software development practices are not taught across traditional engineering
- Fun, travel
- Real action planning
- Proper system architecture (multi-purpose)
- Burnout
- Mindset of "competition" vs "research"
- To not solve tests by just hacking
- Staying pragmatic and focused on outcomes
- Rewarding research
- Meeting RoboCup desired outputs (points) with research oriented outputs (papers)
- Recruiting researchers
- How do you distribute the work?
- How do you approach working with different levels in the spectrum of proactivity?
- Integration of different systems
- How to deal with conflicts?
- Knowledge transfer
- Data availability for training models for Social Robots

8.3 Survey

A survey composed of 60 questions was created from the feedback obtained from the workshop held on 9 of July 2019, the day after the RoboCup Symposium, as found in the previous section. This survey was distributed as an online survey using the Qualtrics[1] platform. An example of how the survey looked can be found in Figure 8.8.

The survey stayed open for six weeks. During this time possible participants were contacted. The survey was still accessible (and some partial answers were requested to be completed) during a further four weeks.

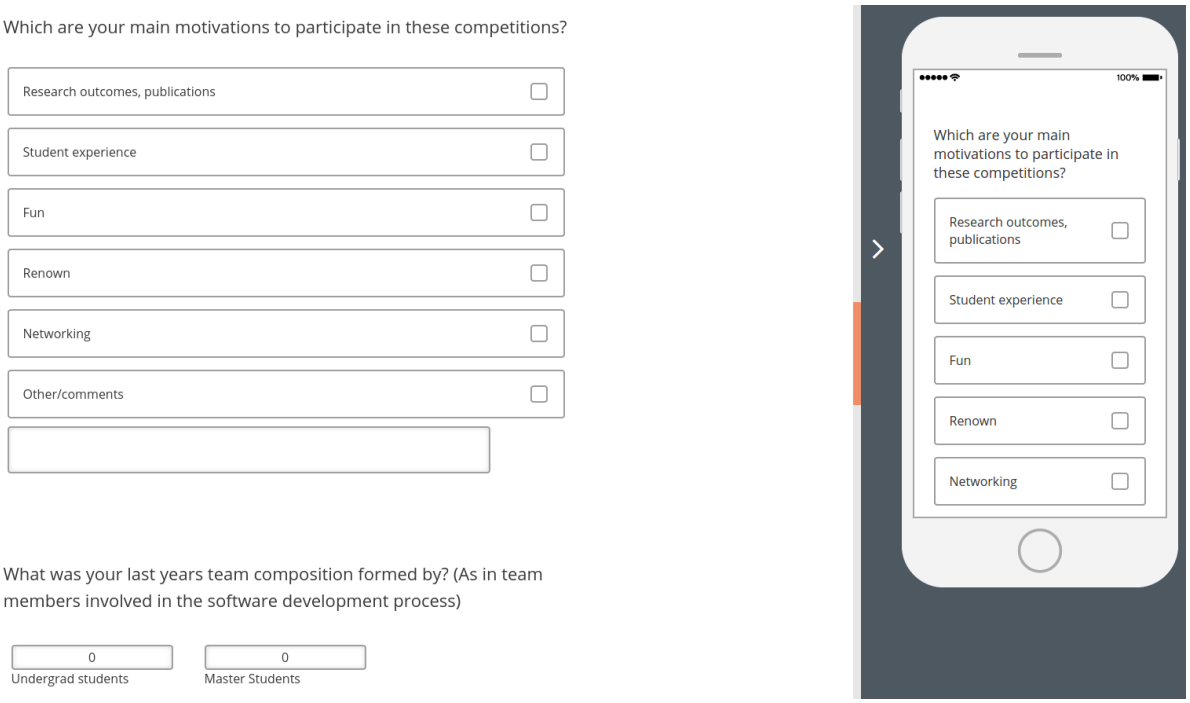


Figure 8.8: Preview of the survey in the Qualtrics platform.

A set of participants was found from the workshop. The survey was also sent to the official RoboCup@Home communication channel (Telegram group) and to the RoboCup mailing list. Participants were encouraged to forward it to other experts to also complete.

This research was conducted in accordance with procedures required by the UTS Human Research Ethics Committee (HREC), application ID ETH19-4405.

[1] Qualtrics is web based software that allows the user to create surveys and generate reports without having any previous programming knowledge.

8.3.1 Survey Aims

The survey aims were:

- Find consensus on specific topics of the software development process.
- Find mixed views in specific topics of the software development process.
- Correlate the software development style with its outcomes.
- Gather insights on this process.
- Find new questions about this process.

These topics, insights, and questions were based on the work done up until that point: the action research cycles and the workshop.

8.3.2 Survey Structure

The survey went through a series of iterations before achieving its final form. It was simplified and shortened as much as possible while taking care to gather enough interesting data.

The survey is divided into 9 sections of questions, containing 60 questions. These sections are:

1. Participant Profile. 11 items relating to the participant profile, e.g. years of experience in RoboCup, team composition.
2. General Questions. 12 items that do not precisely fit other sections like how many person-hours are estimated to be needed to prepare for a RoboCup competition.
3. Recruiting. 5 items about the recruiting process.
4. Meetings. 10 items about the meetings style of the participant's team.
5. Practices from Software Development Methodologies. 13 items about well-known software development methodologies, e.g. if using Agile.
6. Coding Practices. 8 items about coding practices.
7. Outcomes: In research and in the competition podium position. 4 items about the team's outcomes.

8. Experiences in specific situations. 5 items about specific situations that arose in different teams during the development as discussed in the workshop section.

9. Receiving results. 1 question about whether they wished to receive the anonymized results of the survey.

8.3.3 Data Analysis

The survey received 28 answers, 2 of them not fully completed.

In two cases multiple answers were found to be from the same team but from different leaders of different years. Wherever this affected the analysis, it will be pointed out.

This section analyzes the results of the questions in an isolated fashion, one question at a time. By the end of the section a correlation between the 9 top performing teams and their answers is provided.

The questions presented in the survey went through a series of iterations to maximize the interesting insights that could be extracted. Some questions were complex, designed in a manner that would allow different points of view provided by experts who participated in these iterations (including participants of the workshop). Given this conflict, some questions contain a wide range of responses. To extract meaning from those, different approaches are taken and these approaches are explained.

Furthermore, an example of how the question Q6.2-Q6.3 was analyzed can be found in Appendix C.

8.3.3.1 Participant Profile

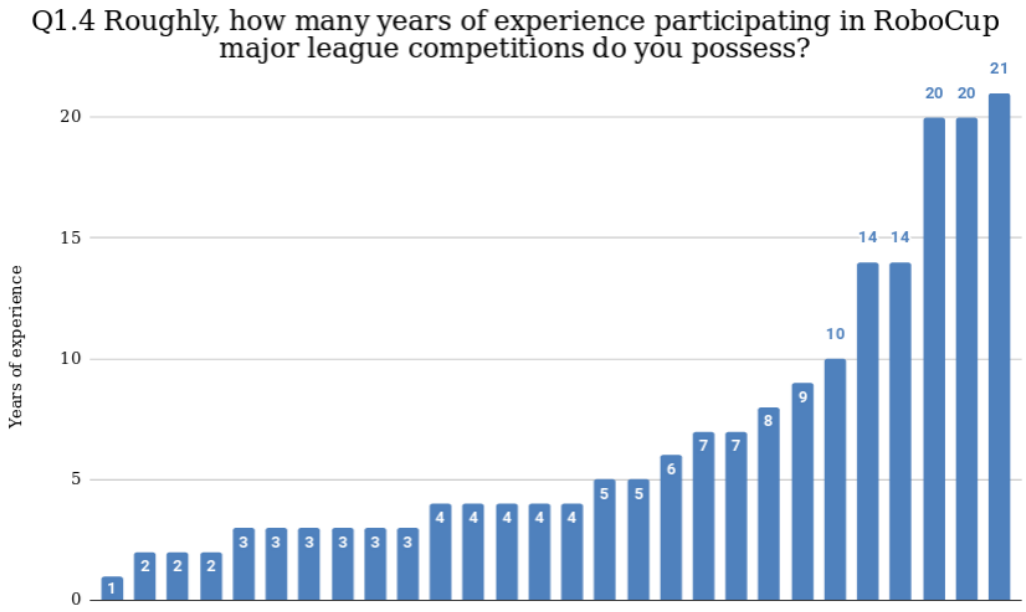


Figure 8.9: Minimum years of experience in RoboCup was 1 year. Maximum 21 years. With an average of 6.75 years and a median of 5 years of experience. The accumulated amount of years of participation is 189 years of experience.

As per question Q1.4 in Figure 8.9 we can find that most of the participants have 3 years or more of experience in RoboCup major leagues. 24 expert participants create a representative body of the niche field of software development for RoboCup. There are 4 additional answers with 1 and 2 years of experience which also provide useful feedback, even though their view of the competitions may not be as deep.

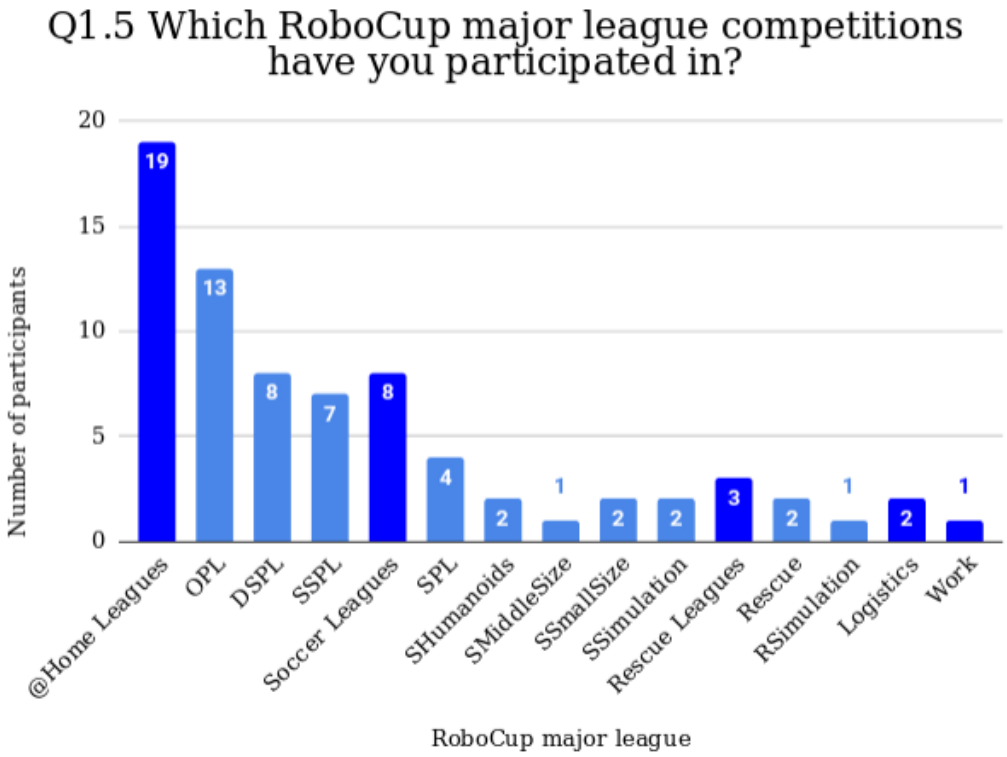


Figure 8.10: The majority of the participants have participated in RoboCup@Home (with its subleagues participation shown too). This is to be expected as the author of the survey was a participant in this league so it was easier to reach out to them. The survey was distributed to experts of all leagues.

As per question Q1.5 in Figure 8.10 it is to be observed that most participants come from the RoboCup@Home league. Given the author participated in this league and their community has an easy to approach communication channel, this seems natural. There is representation of the Soccer, Rescue, Work, and Logistics leagues too.

Question 1.6 stated: *Please add any other Robotics Competition / Challenge with a similar spirit to the RoboCup major league competitions that you have participated in (E.g.: RoCKIn, World Robot Challenge, NASA competitions...)*. 14 participants reported having participated in other competitions, these were: World Robotics Summit, Darpa DRC, Toyota Research Institute Pick and Place Challenge, RockIn, Mexican Robotics Competition, European Robotics League, Darpa Grand Challenge, Eurobot, NASA Space Robotics Challenge and local opens like Iran Open, German Open and US Open. This strengthens further the belief of the quality of their answers in the survey.

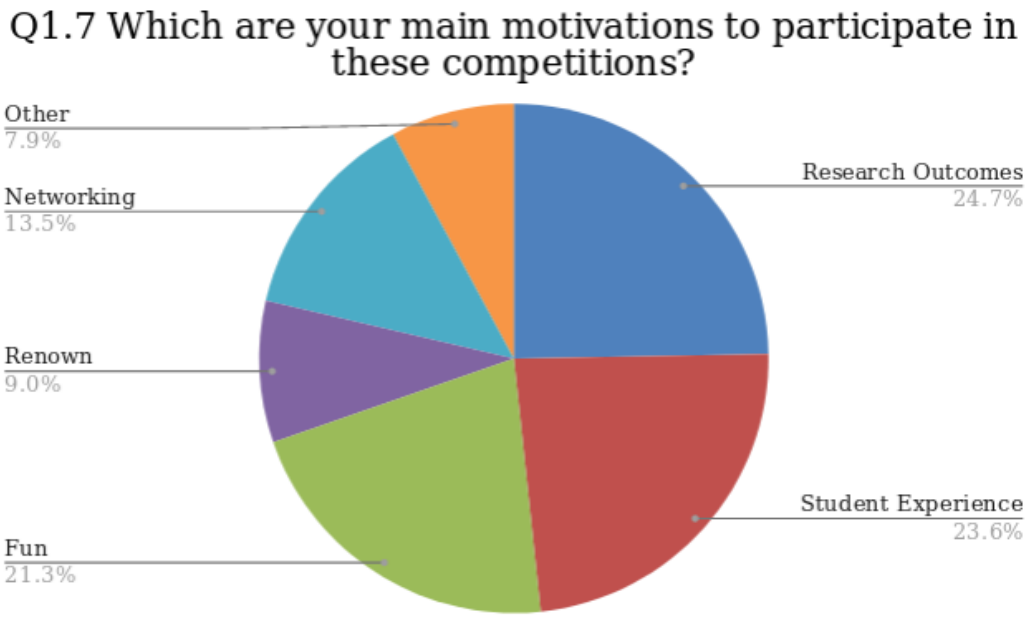


Figure 8.11: The reported main motivations to participate in these competitions are research outcomes, student experience, and fun. These motivations are followed by networking and renown.

From Q1.7 with a chart in Figure 8.11 we see that participants are likely to join RoboCup hoping for research outcomes, provide student experience, and have fun. Furthermore, networking and renown have an implication.

Participants that chose 'other' as an answer reported the topics:

- Development of robotics.
- Capabilities development.
- Using RoboCup-systems as a research platform.
- Stay up to date with the state of the art.
- Credibility.
- Validation in the real world.
- Working with particular people involved in the team.

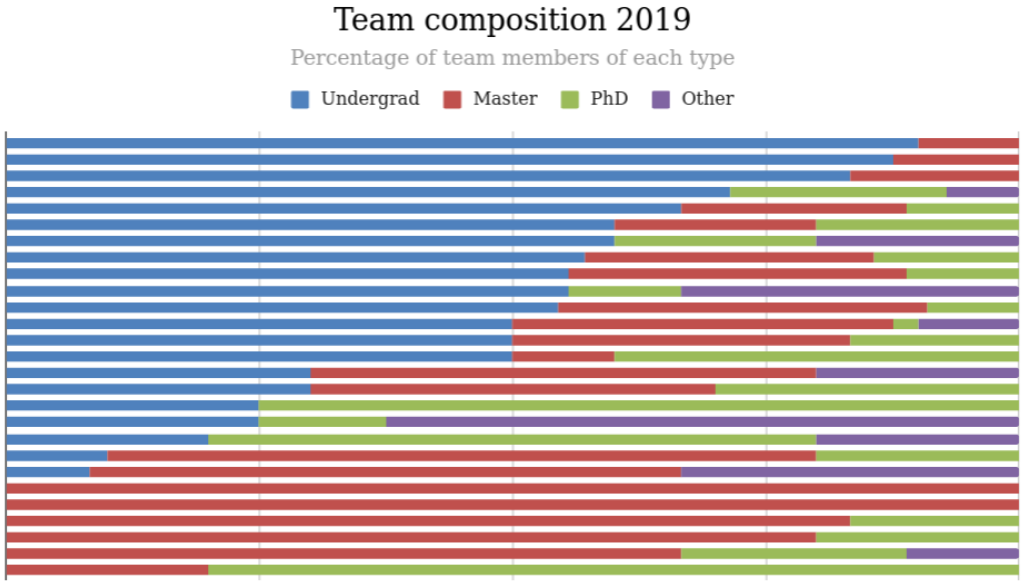


Figure 8.12: Team composition shown as percentages of the total team size. There is diversity on team compositions as per the participants reporting from the 2019 edition.

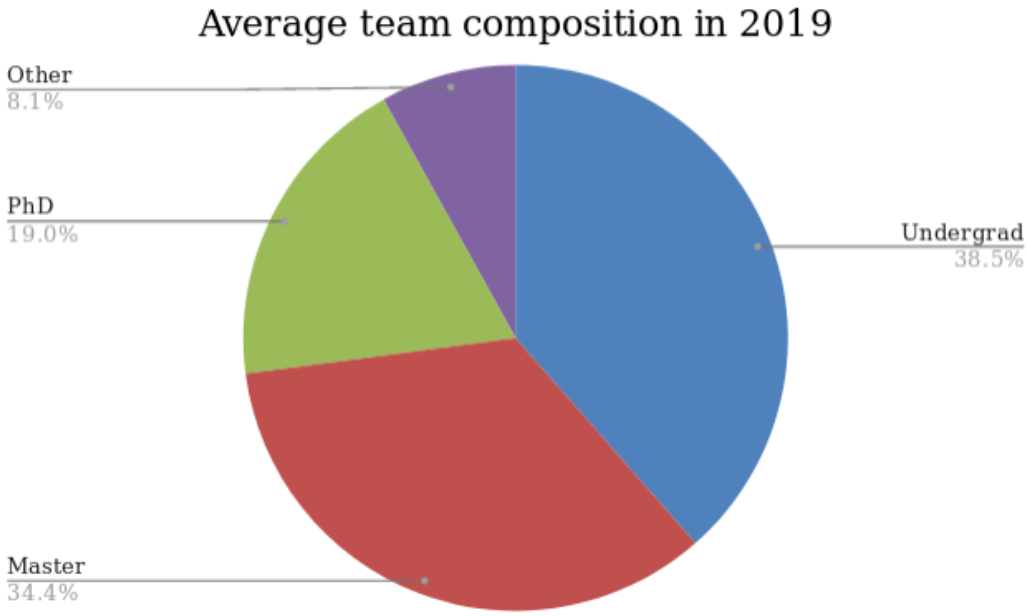


Figure 8.13: The average team composition for 2019 was dominated by Undergrad students, followed by Master students. Then PhD students and those reported as Other.

Question Q1.9, with a chart illustrating it in Figure 8.12, asked about the team composition for the last year (2019) for the survey participant teams. Diversity can be found in team compositions. From mostly undergrad students up to no undergrad students, for example.

Moreover, to gather further insight the average team composition in 2019 was computed in Figure 8.13. Here we observe a dominance of undergrad and master students at similar percentages, followed by PhD's, and, finally, other.

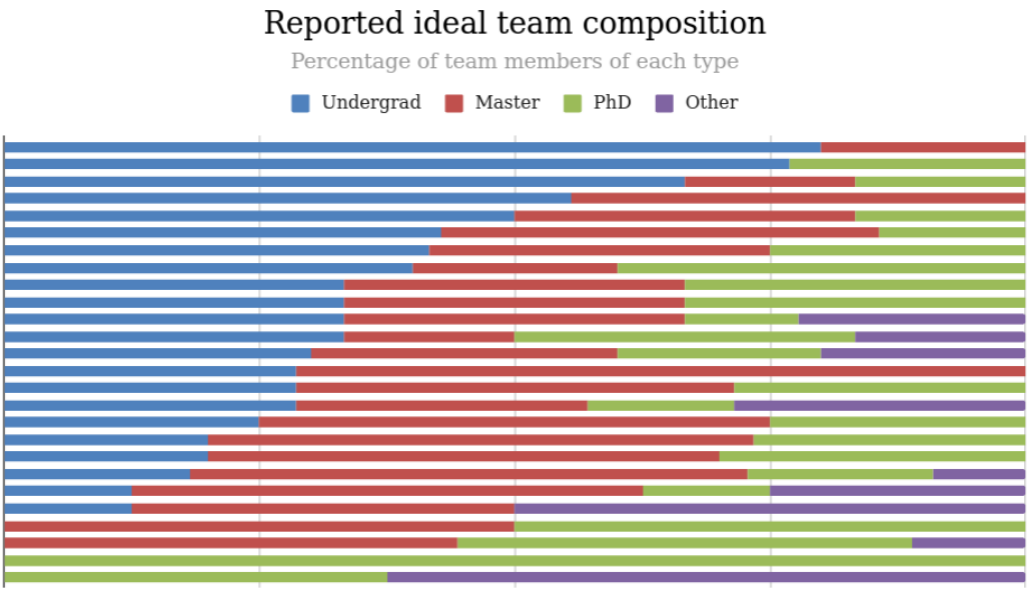


Figure 8.14: Team composition shown as percentages of the total team size. From the point of view of their ideal team composition there is a variety of answers too.

The question was asked again in Q1.10 but in terms of what would be their idea team composition. In Figure 8.14 we find again a variety of answers.

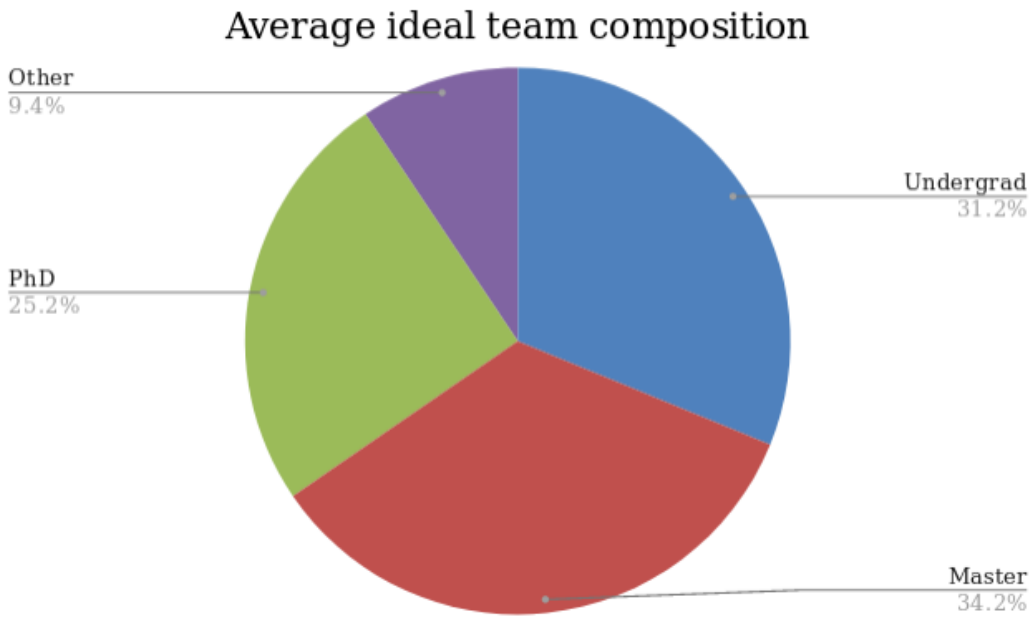


Figure 8.15: The average ideal team composition is similar to the reported one for 2019 in 8.13 but with less Undergrad students and with more PhDs filling that gap.

The average on this data is computed and shown in Figure 8.15. Here we observe a similar distribution of undergrad and masters students, but both reducing their percentages to increase the amount of PhD students. It can be interpreted as a more balanced average team composition.

From Q1.9 we see that the average team composition for 2019 was of about 3.7 undergrad students, 2.4 master students, 1.4 PhD students, and 0.8 other. This averages to about 8 team members in each team. Other roles have been reported as employee, graduate continuing with team, engineer, professor, and research engineers/staff.

In the case of Q1.10 where we asked about the ideal team composition, we obtained an average of 3.4 undergrad, 3.4 master, 2.7 PhD, and 1 other. This averages to about 11 team members. Other was reported as research assistant, professor, and postdocs.

We can interpret that, on average, an ideal team composition has more higher education tier members in comparison to the actual team composition for 2019 as determined in Q1.9..

On Q1.11 participants were asked about the rationale behind their ideal team composition. Summarising the common points of the answers here, sorted by frequency, as:

- Almost all answers say that people with experience are needed. Both RoboCup ex-

perience and engineering experience. Mostly reported as PhDs but some professors too.

- Most answers mention somehow a chain of teaching, PhD guide/teach masters, masters teach/guide undergrad.

- Undergrads to do mechanical work including testing, labeling, writing down experimental data.

- Small teams are easier to manage (small seeming to be around 5 people).

- Commitment for multiple years of the team members is desired and looked for.

- Making team members join early on (from Undergrad) to maximize their commitment.

- Take whoever that comes to the project that commits to it.

Looking at these rationales, it seems experience is a very important factor to have in the RoboCup competitions. Teams aim to maximize the time their members stay with them and want guidance and teaching to happen from these experienced team members. Nonetheless, small teams are mentioned for their supposed ease of management, while there was one mention of having larger teams to maximize outcomes (both in research and in winning).

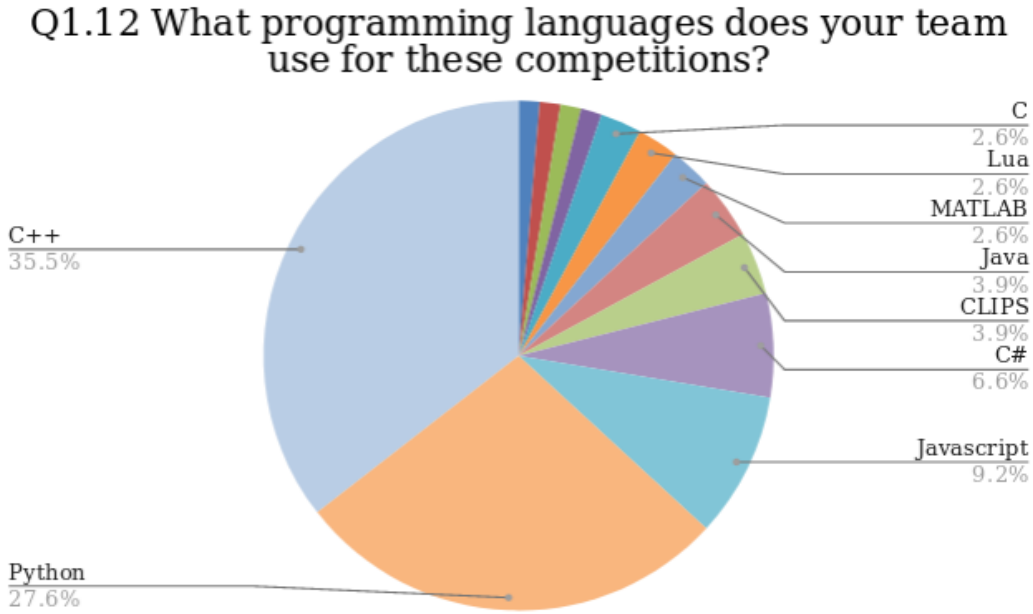


Figure 8.16: The most used programming languages reported were C++ and Python. Other languages had some usage, either by teams who were the only ones reporting using it as their main language, or as languages to solve specific tasks.

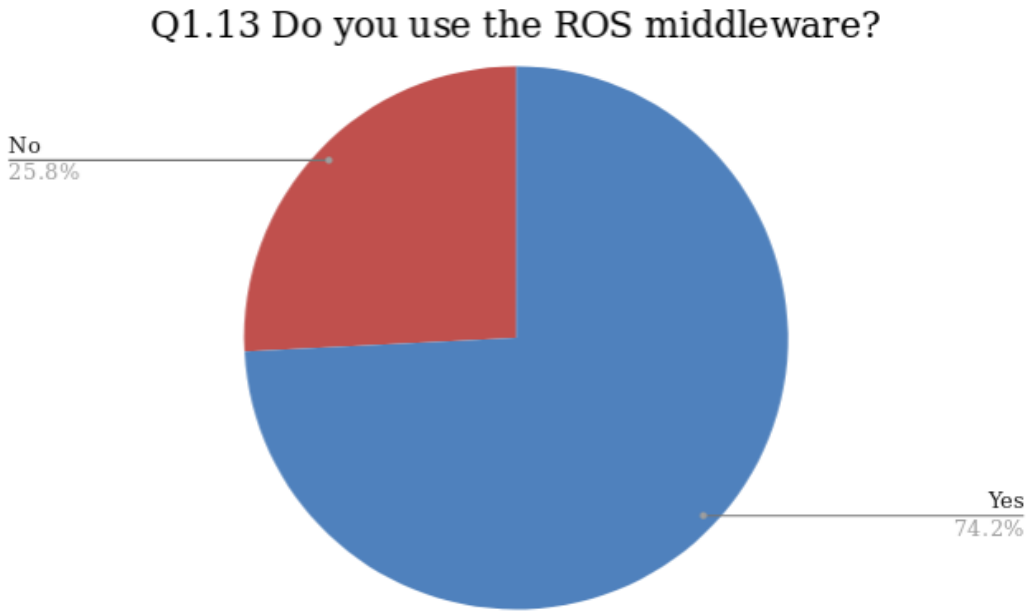


Figure 8.17: Most teams use the ROS middleware. But it is worth noting that the league in which teams participate influences this question.

Q1.12 asked about the programming languages used for the competitions. In Figure 8.16 we see a predominance of C++ and Python. This follows the trends shown in surveys of recent years like StackOverflow's [86]. Robotics software tends to need high performance components, which tend to be programmed in C++, with more higher level code done in Python. Additionally, the mainstream usage of deep learning, with widely well known libraries in Python, inclines the balance towards Python. Most teams report usage of the Robotics Operating System (ROS) framework as stated in question Q1.13 (Figure 8.17). This has many tutorials in C++ and Python, which may also influence this distribution.

In Q.13 participants were asked about their usage of the ROS middleware. As shown in Figure 8.17 most teams (74.2%) answered that they use it. Teams that answered negatively reported that it did not make sense in their league, or that they used their own custom framework instead, or in parallel, of ROS.

8.3.3.2 General Questions

For the question Q2.2 *"How many person-hours do you estimate are needed to successfully participate in a RoboCup competition?"* the answers were to be provided in terms of:

*Number of team members * Weekly hour dedication * Duration of the project in weeks*

This format was chosen taking into account feedback provided from the workshop from section 8.2. A number of people agreed that a simple count of person-hours was not a good enough metric, however, it would be meaningful to take into account different project and team setups to reach such a number.

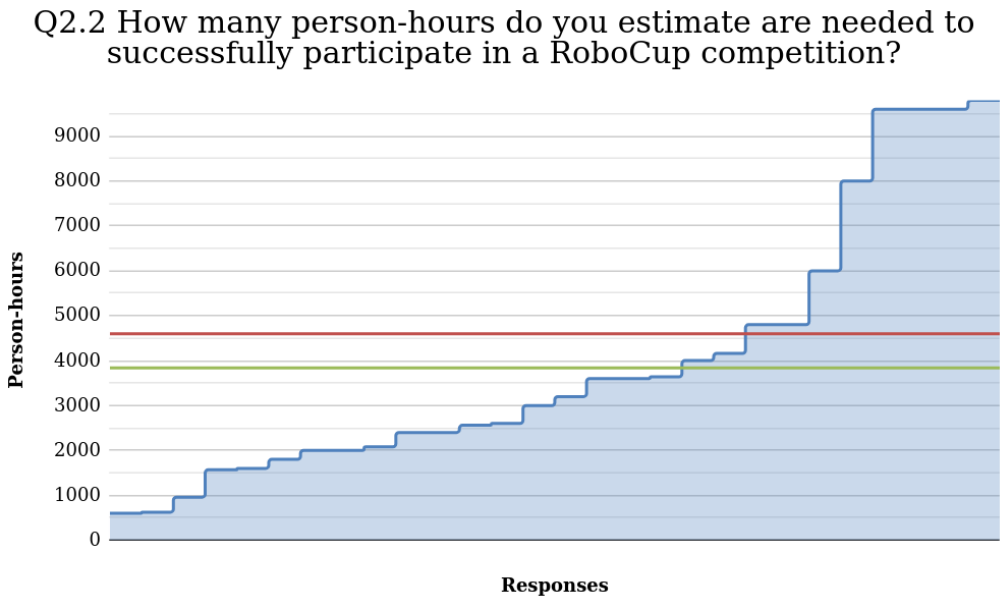


Figure 8.18: Total person-hours responses for the question “How many person-hours do you estimate are needed to successfully participate in a RoboCup competition?”. Average of 4596 person-hours and median of 3840 person-hours represented with a red and a green line, respectively. There is a wide range of responses from just 600 person-hours to 98000 person-hours.

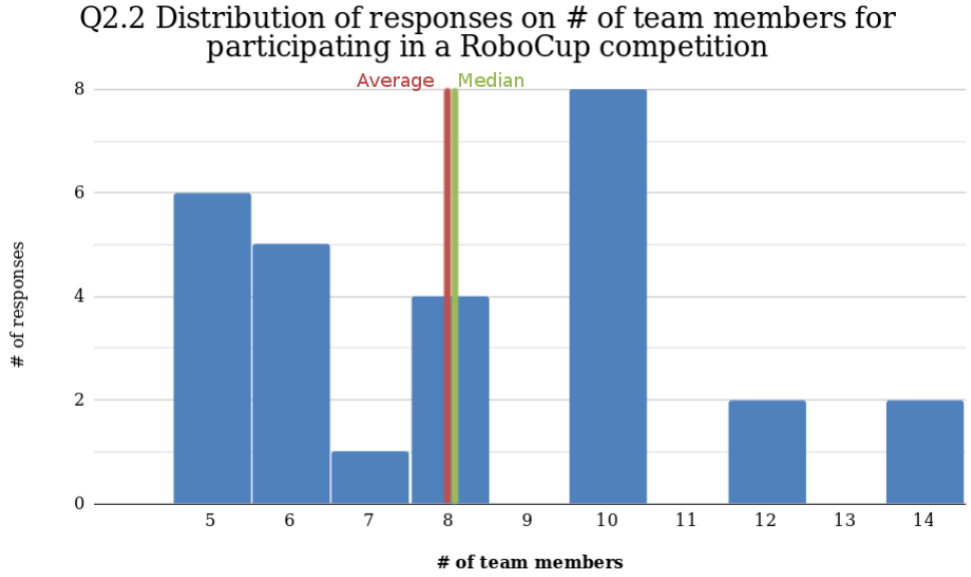


Figure 8.19: Distribution of responses on the amount of team members for participating in a RoboCup competition with regard to the total person-hours needed for the project. Average and median values fall on 8 team members.

Looking at the distribution of responses in total person-hours in Figure 8.18, there is a wide range of responses. A meaningful way to use this information is to form a consensus. Here the averages, medians and distributions of the responses are taken into consideration.

Item	Amount	Rationale
Number of team members	8	Both the average and median fall around 8 team members.
Weekly hour dedication	20h/week (part-time)	The average and mean fall in between 17 and 20h/week. In real life part-time on 20h/week is a common measure.
Duration of the project in weeks	28 weeks (7 months)	The average and mean fall in between 24 and 31 weeks. It is usual to plan in months, hence, rounding to a middle value.
Total person-hours	4480	

Table 8.2: Creating a meaningful guidance value on how much effort in person-hours an average RoboCup team needs.

The distribution of responses to the amount of team members is in Figure 8.19, the distribution of responses to the amount of weekly dedication hours is in Figure 8.20, and the distribution of responses to the amount of weeks to prepare for RoboCup is in Figure 8.21. Then, the real-life practicality of the different measures is taken into account. This process is shown in Table 8.2. The result of this process is a set of guidance values. A team of 8 members working part-time (20h/week) on the project during 7 months (28 weeks) would be considered to have a setup able to be successful in a RoboCup competition.

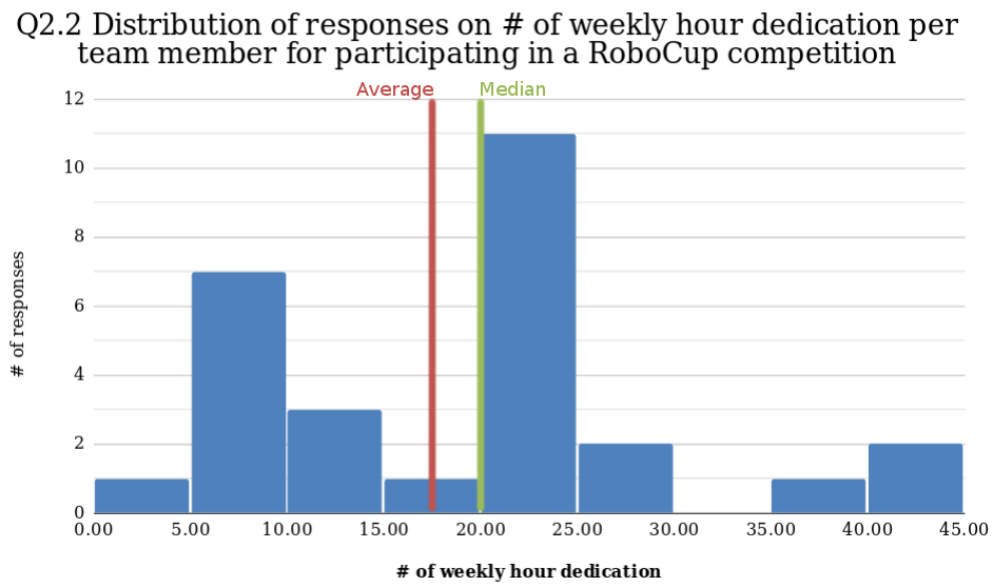


Figure 8.20: Distribution of responses to the amount of hours weekly per team member to dedicate to participating in a RoboCup competition with regard to the total person-hours needed for the project. Average value is 17.5 hours/week and median value is 20 hours/week.

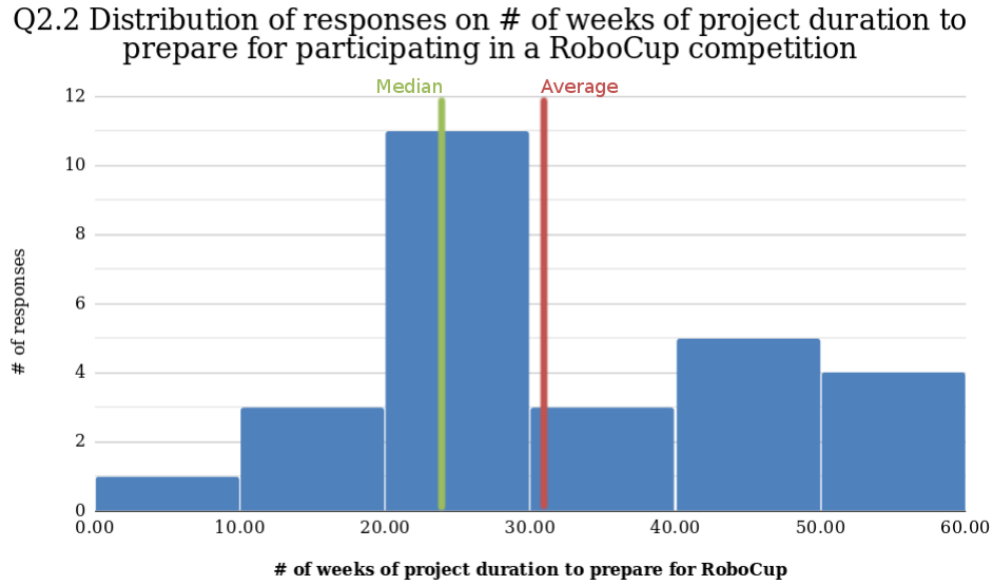


Figure 8.21: Distribution of responses to the number of weeks of dedication to prepare for participating in a RoboCup competition with regard to the total person-hours needed for the project. Average value is 31 weeks and median value is 24 weeks.

It is worth noting that for the number of team members, the average and median falls at 8, but the most frequent answer was 10 team members. Moreover, a more significant value may be the range in between 5 to 10 team members as 92% of the answers fall into that range.

The next question stated *"Do you have / create a practice environment where to simulate the competition? (E.g.: A home-like room for RoboCup@Home or a disaster scenario for RoboCup@Rescue. Shortly describe your setup if so, please."* and all responses but one answered affirmatively. On the descriptions of their setups the consensus was that they had an environment as similar as possible to the competition, as their resources would allow.

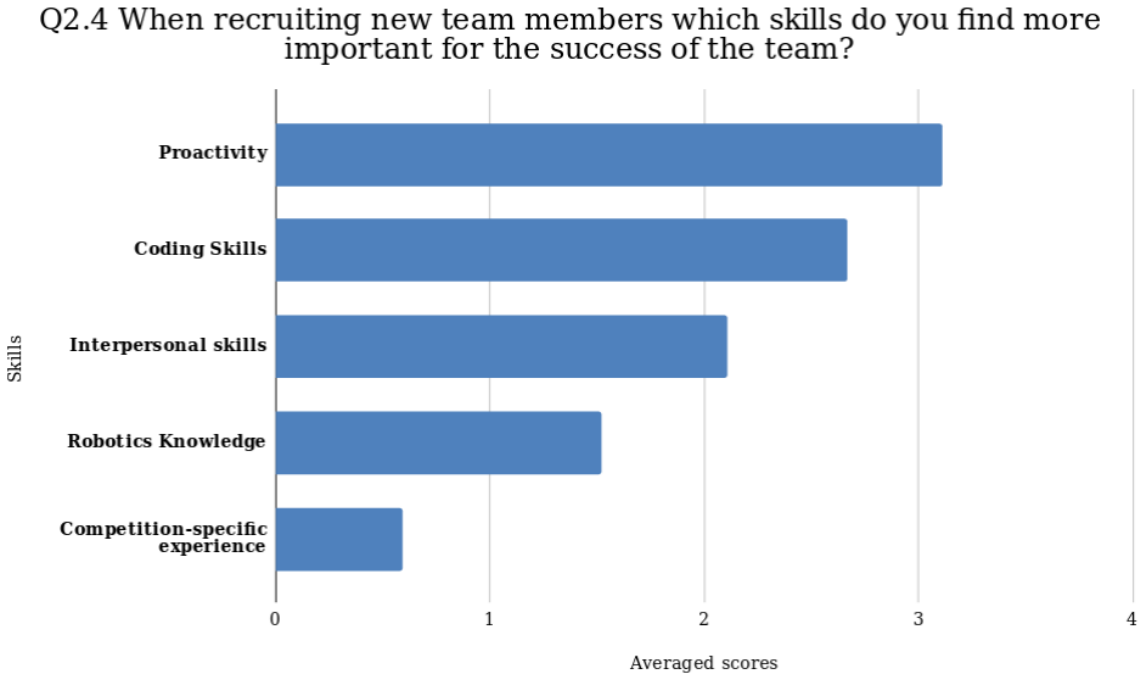


Figure 8.22: Responses on which skills were found to be more important for the success of the team when recruiting new team members. The skills ordered by importance became: proactivity, coding skills, interpersonal skills, robotics knowledge, and competition-specific experience.

The following question Q2.4 stated *"When recruiting new team members which skills do you find more important for the success of the team? Please sort by importance the following items."* and showed a list of 5 skills to sort: interpersonal skills, coding skills, robotics knowledge, competition-specific experience, and proactivity.

During the workshop it was impossible to reach an agreement on what was more valued but those skills arose as important. The question was shaped in this manner from a previous iteration, where a participant would just rate how important were these skills in a 5-scale from "Not important" to "Very important". This trial questionnaire was not effective and this approach, where participants needed to engage in a trade-off, was used instead. A follow up free-text question allows for additional notes.

In Figure 8.22 we can see the outcome of this question. Proactivity is the most wanted skill, followed by coding skills, then interpersonal skills, then robotics knowledge, and finally, competition-specific experience. Additionally, the data showed that half of the respondents chose proactivity as the most important skill, and 70% of respondents chose competition-specific experience as the least important skill.

The result that competition-specific experience was rated the least important skill might be surprising taking into consideration that previous responses indicated that it was important to have team members with competition experience. However, it may be that the question was in the context of recruiting new team members and therefore was assumed by the respondents that the team already had sufficient experienced team members.

A text field followed stating *"Please add any comment you may have on your answer of the previous question."* for additional notes. 42% of participants added some further comment. These comments are summarized as:

- Motivation is reported as related to proactivity and is considered very important by three participants. Autonomy is also reported as related to proactivity by one participant.
- "Cannot be too selective on recruiting new team members" is expressed by two participants.
- "Some team members must have experience" is reported by two participants.
- Discipline and following the team leader decisions is also reported as important by one participant.
- The shape of the question is criticized as simplistic by one participant, but no improvements are proposed.

Question Q2.6 was *"How many team members do you fly to the competition?"* with an additional field stating *"What is the decision on how many team members to fly to*

the competition based on? Please explain.". As can be seen in Figure 8.23, the average number of team members flown is 7, the median is 8, and the most frequent response being 10. The range of responses went from 3 to 10.

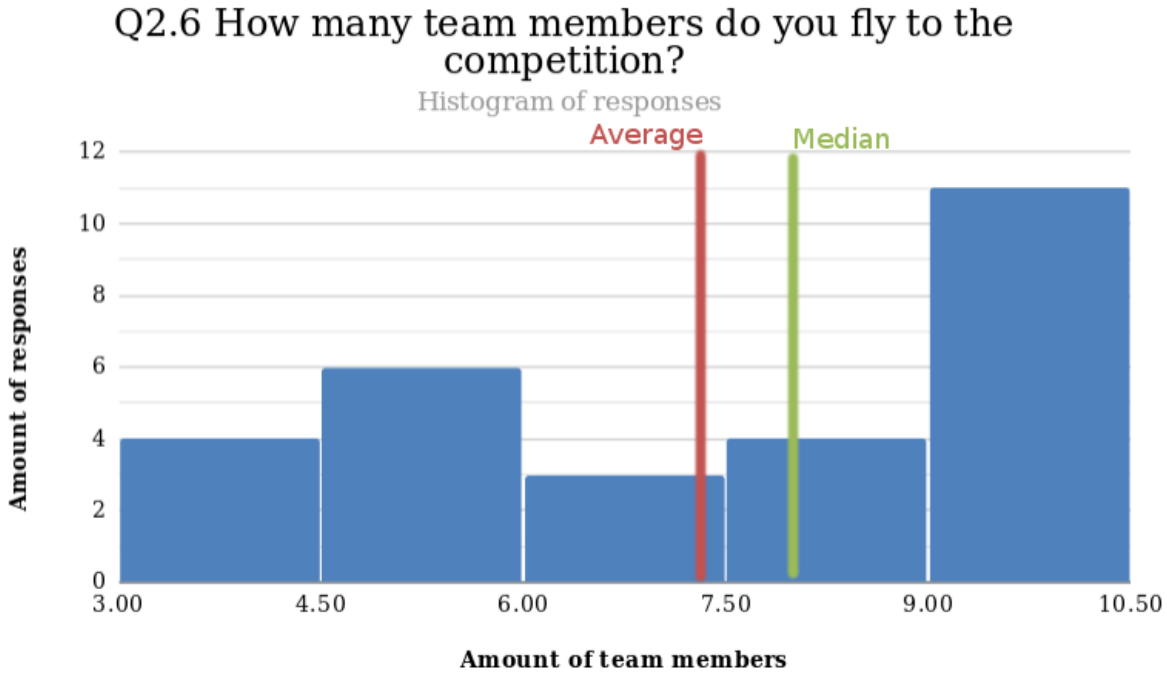


Figure 8.23: Responses on *"How many team members do you fly to the competition?"*. Average is 7, median 8 and the most repeated response 10.

The arguments about why the reported amount (and implicitly, whom) of team members to fly are summarized as:

- First and foremost: Funding. It is mentioned by almost all responses.
- Availability.
- Prioritize by most important skills or responsibility on the project. Also prioritize by the most capable people and their merits.
- With the previous factors taken into account, try to have everyone in the team.
- But also keep the size of the onsite group manageable.
- And one answer added that for the Middle Size League you need two persons per robot.

We find in Q2.8 *"How important is it to have an always fully working version of the software of your robot at any time?"* and its results in Figure 8.24. Most respondents answered it's in between extremely important and very important.

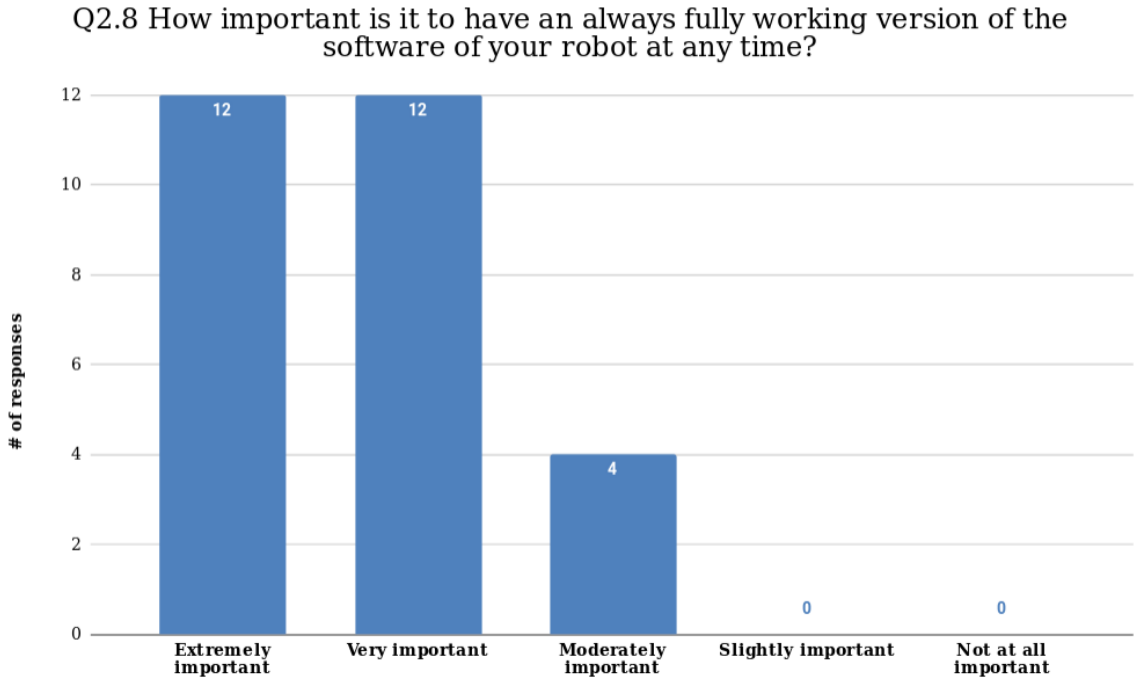


Figure 8.24: Most respondents to *"How important is it to have an always fully working version of the software of your robot at any time?"* answered that it was in between extremely important and very important. This shows less emphasis than the previous question.

Q2.9 follows with *"How important is it to have automated hardware checks? (E.g.: Having some software automatically tell the user a sensor is not working correctly)"* and the plot with the results can be found in Figure 8.25.

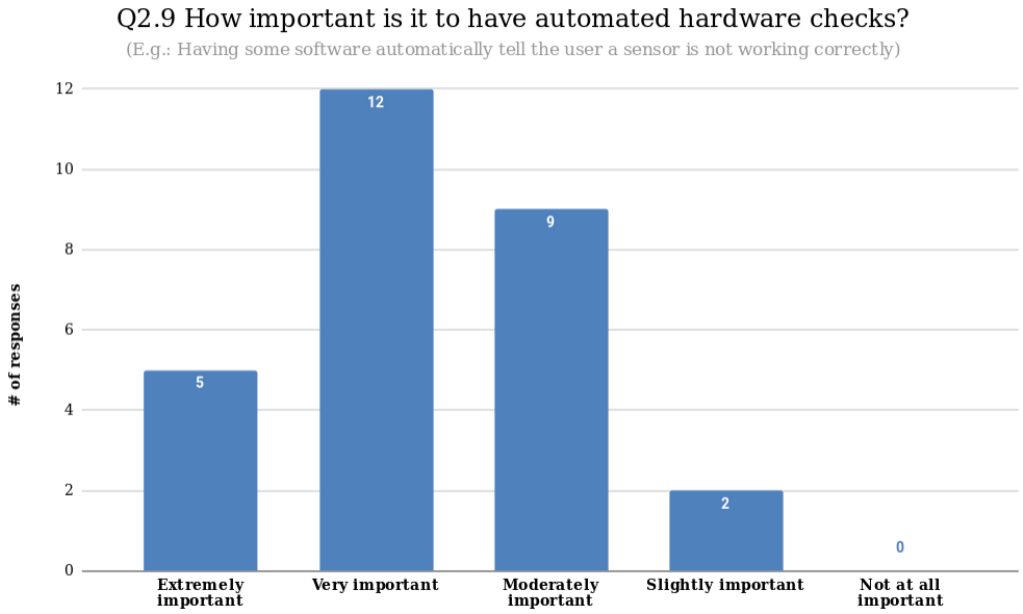


Figure 8.25: *"How important is it to have automated hardware checks?"* has most replies in the range of very important and moderately important.

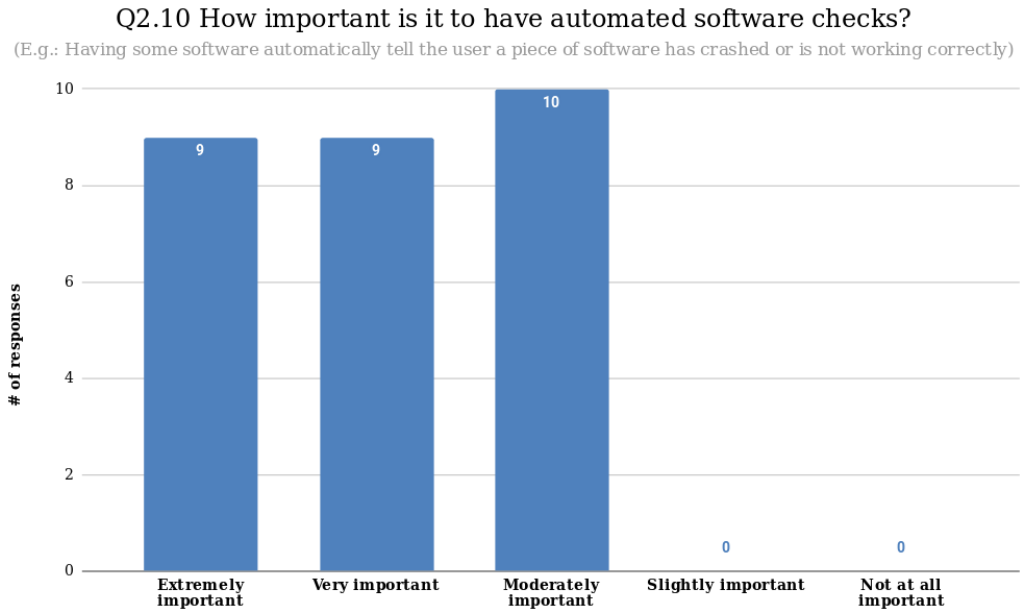


Figure 8.26: The responses to *"How important is it to have automated software checks?"* are found almost equally distributed in the range in between extremely important and moderately important.

In Q2.10 *"How important is it to have automated software checks? (E.g.: Having some software automatically tell the user a piece of software has crashed or is not working correctly)"* with its chart in Figure 8.26, the answers are evenly distributed in the range in between extremely important and moderately important.

Q2.11 contains the question *"How important is to have a simulation of your robot available for development?"*, which has a follow up text field to give further explanation. The responses are plotted in Figure 8.27 having a mix of answers. Taking into account that different leagues take part in this competition with different levels of available simulations, this may not be surprising.

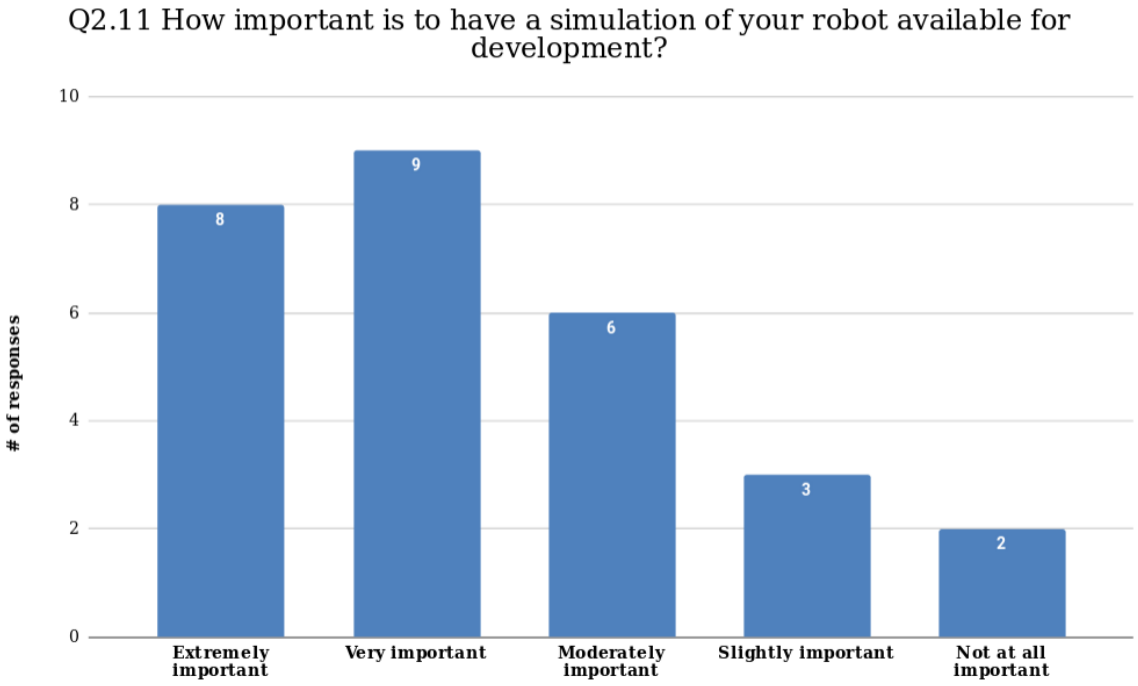


Figure 8.27: A wide range of answers for *"How important is to have a simulation of your robot available for development?"* is found. Most answers fall around being very important, but there is no consensus.

On the section for comments of this question, summarized, responses said:

- Robots are a limited resource, so simulation is needed.
- Simulation is necessary to minimize breaking robots.
- Using a simulator allows for development and testing in variable scenarios without a robot.

- Testing first in simulation before testing in the robots.
- Simulators may not be realistic enough.
- Some responses say that they want to use more simulation.
- Some responses say that they will only use a simulator if it's already available.
- Testing in some leagues is necessary to be done in simulation.

8.3.3.3 Recruiting

As in other parts of the survey, the choices available for these questions were based on discussions in the workshop from section 8.2.

The first question of the recruiting section is coded as Q3.2 and states *"How do you recruit new team members?"*, the responses can be found in Figure 8.28. 'Other' included the answers: internships, word of mouth, flyers, the initiative is presented during orientation and "they come to us".

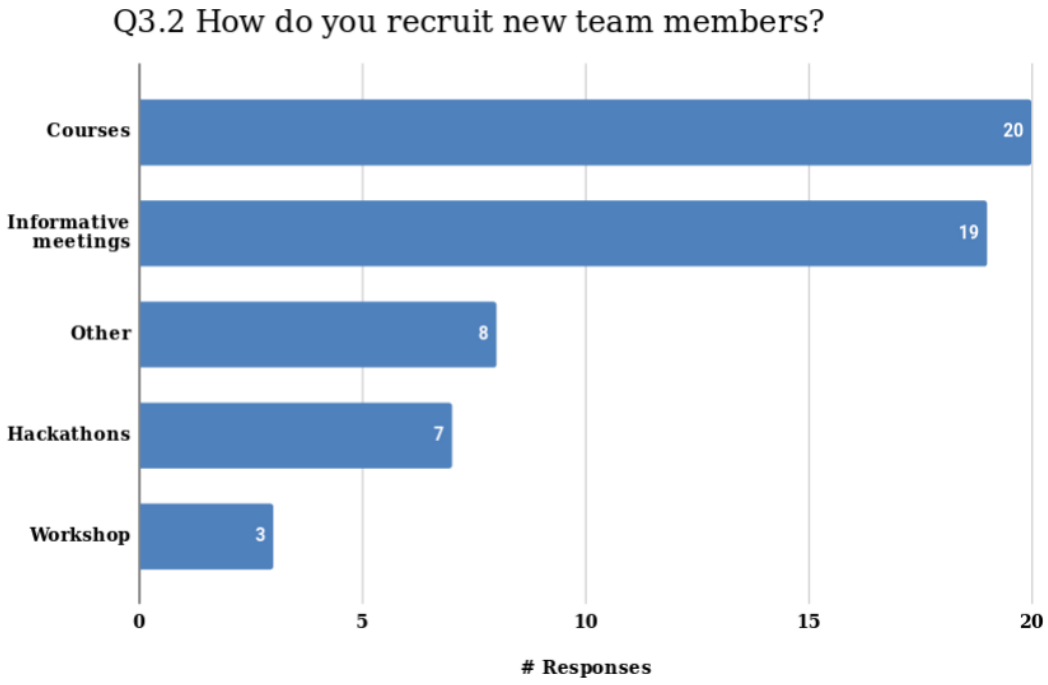


Figure 8.28: *"How do you recruit new team members?"* responses. The most popular answers were Courses and Informative meetings. Other follows. Then hackathons and workshops.

On Q3.3 *"What incentives do you offer to your team members to participate in the team?"* the incentives with more responses were publications (they get the opportunity to do scientific publications) and future job (future job prospects, they ease their way into a job) with an equal number of respondents. Coursework (coursework, credits, they advance their studies), and 'other' follows. Comments for the 'other' response were: paid travel, build robots and work with new robots as motivation, learning experience, fun, and fame. Lastly, free goodies (as in food/drinks in social events) and money (they are paid).

Focus on their personal growth goals seem to be the most important reasons. Hence, publications, future job, and coursework were the most frequent answers, compared to: other, free goodies, and money.

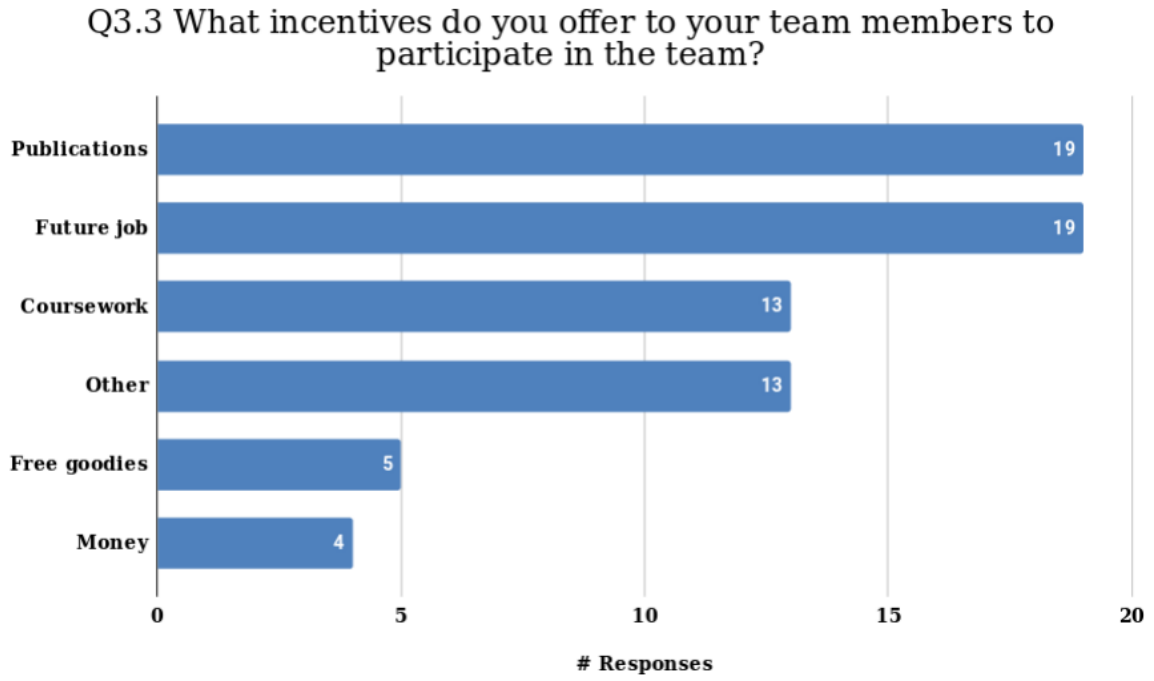


Figure 8.29: Distribution of responses for *"What incentives do you offer to your team members to participate in the team?"*. Most responses went into publications and future job. Coursework and other follows. Finally free goodies and money. Comments for the other response were: paid travel, build robots and work with new robots as motivation, learning experience, fun, and fame.

Question Q3.4 *"How do you teach the basics needed to participate in the project (or make sure the new team members have the necessary knowledge)?"* with the plot with the responses in Figure 8.30, had the following responses ordered by number

of responses: personal work and supervised personal work had the highest number of responses. Course (as in, being part of a university subject) follows. With a lower number of responses workshop and others finish the possible amount of options. For other, answers were: thesis work as part of their enrolment, could be improved, pair programming, focused experiments, and instructions by seniors.

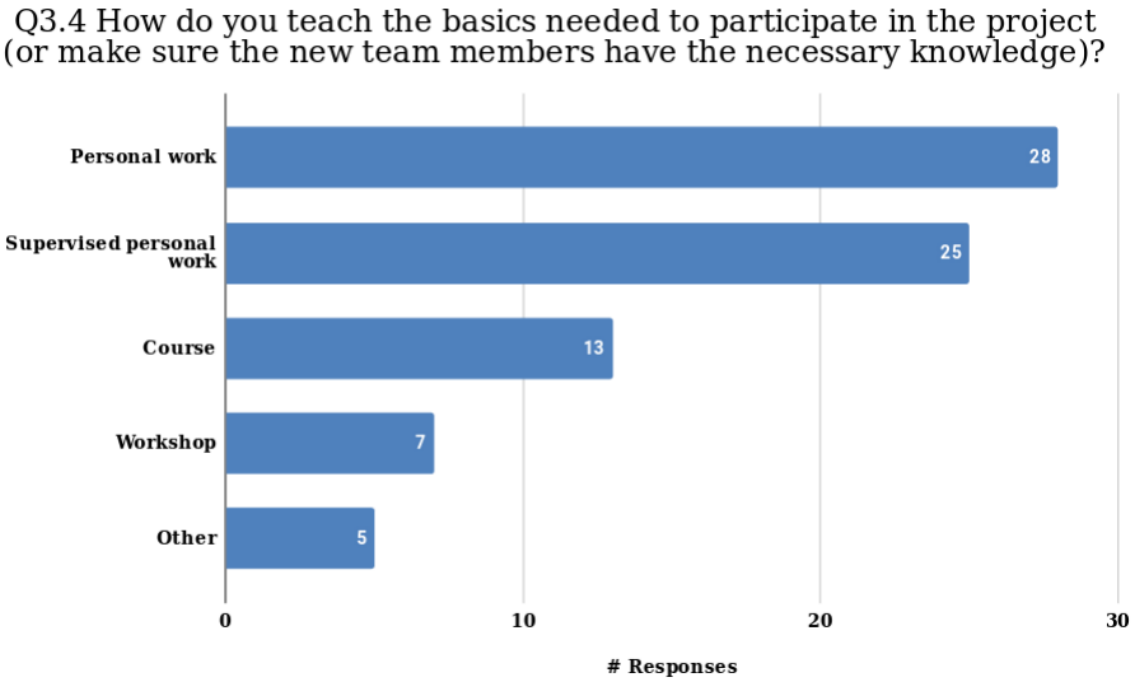


Figure 8.30: Distribution of responses for *"How do you teach the basics needed to participate in the project (or make sure the new team members have the necessary knowledge)?"*. Most responses found in personal work and supervised personal work. Course got roughly half the responses than the previous two, but still roughly double the two last options: workshop and other. Comments for the other field were: thesis work as part of their enrolment, this process could be improved, pair programming, focused experiments, and instructions by seniors.

The last question of this section, Q3.5, is specified as *"How long does it take to a new team member to make a non-trivial contribution to the team's repository? (Please add the unit, hours, days, weeks, months...)"*. The reasoning behind this format is that, during the workshop, it was hard to come to a consensus about this question. There were participants with setups where contributions were not allowed until team members were very familiar with the codebase, and others with an opposite mindset where they encouraged contributions from as early as possible. The concept of a non-trivial

contribution is also conflicting. Moreover, a wide range of responses was found even when trying to setup the same assumptions. This question was left open to gather further insight.

The responses were, unsurprisingly to the author, very varied. Responses included “it varies too much from person to person”. It can be implied from the responses that also what is being considered "contributing" may be interpreted very differently, as there were responses ranging from twenty hours to one year. Some answers also considered just learning ROS as the necessary requisite to contribute.

Noteworthy, was that most answers were provided in months, with some in a range of months. Moreover, with the goal of aiding the extraction of meaning from this data, a histogram was created. This plot can be found in Figure 8.31, and it shows two schools of thought: teams that expected new team members to make a non-trivial contribution in between 1 and 3 months, and teams that expected it to happen in between 5 and 8 months.

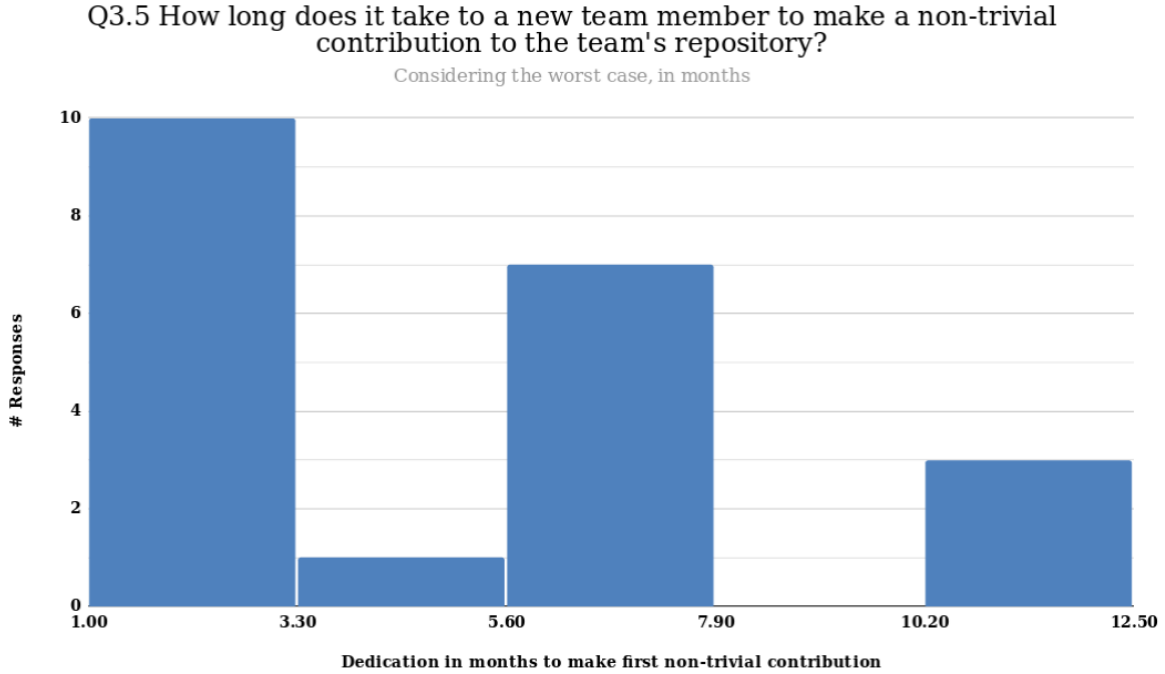


Figure 8.31: For *“How long does it take to a new team member to make a non-trivial contribution to the team’s repository? (Please add the unit, hours, days, weeks, months...)”* the distribution of responses interpreting results in the worst case in months in a histogram is used. Using whole month figures, most answers fall into the 1 to 3 months range. The next most answered range is in between 5 and 8 months.

8.3.3.4 Meetings

The first question of the meetings section has the code Q4.2 and states *"What kind of periodic meetings does your team hold?"* with the (multiple options allowed) options of: daily, weekly, fortnightly, monthly, and other. The responses are summarized in Figure 8.32.

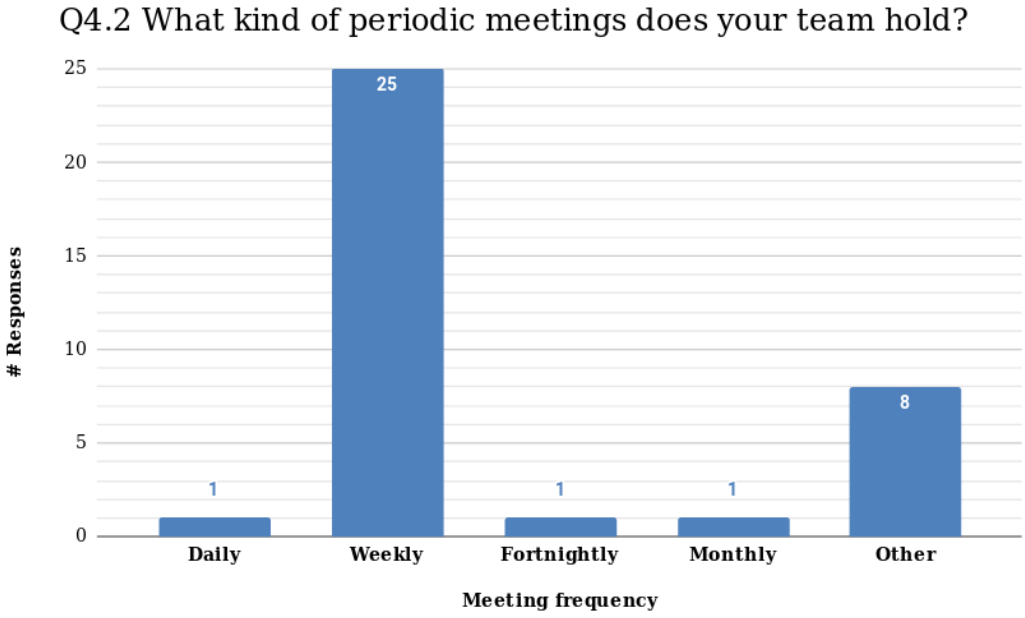


Figure 8.32: Responses for *"What kind of periodic meetings does your team hold?"*. Most responses indicate Weekly meetings. 8 responses also indicated Other: changing frequency (to more often) as the competition approaches, irregular meetings.

From the people that chose weekly as their meeting type, Q4.3 asked *"How many meetings do you have a week?"* and the summarized responses, as can be seen in Figure 8.33.

The questions Q4.4 to Q4.7 were about the duration of the meetings and were shown based on the previous answers about the periodicity of meetings. E.g., if a respondent marked they met weekly, they would have responded to the question about how long their weekly meetings were. The summary of the responses can be observed in Figure 8.34. There is a wide distribution of answers on the most common type of meetings (weekly) ranging from 4 hours to just 10 minutes.The average weekly duration was 75 minutes with the median being 60 minutes.

Figure 8.33: Distribution of responses to *"How many meetings do you have a week?"* from respondents that indicated that they meet weekly. 80% of respondents meet once a week and 20% meet twice a week.

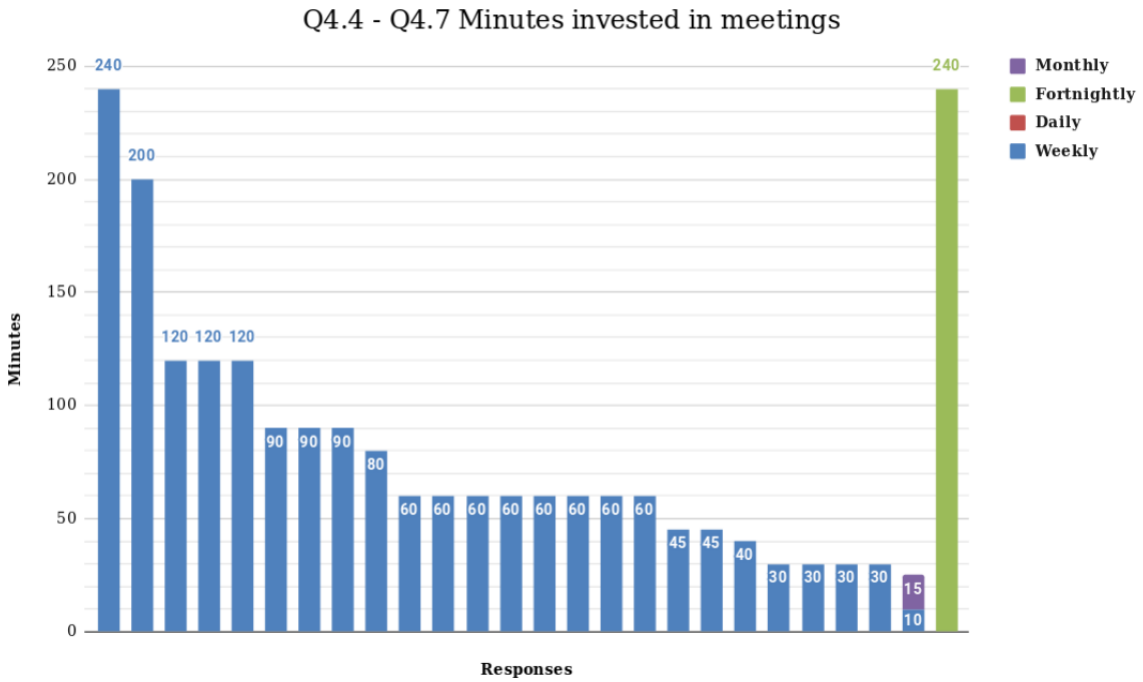


Figure 8.34: Summary of responses to duration of meetings. Average weekly duration was 75 minutes and median was 60 minutes.

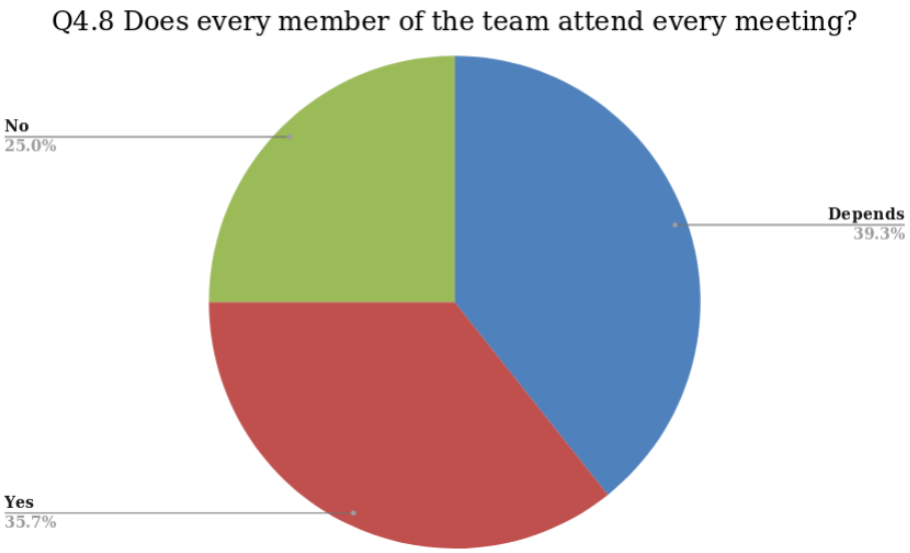


Figure 8.35: *"Does every member of the team attend every meeting?"*.

Question Q4.8 stated *"Does every member of the team attend every meeting?"* and the responses were roughly divided in thirds. The three options yes, no, and depends were distributed as seen in Figure 8.35. On the options no and depends, an optional explanation was offered. The explanations provided were mostly that the spirit of the meetings was to have everyone in the meeting but availability of the team members made it hard. A few answers manifested that there was a subdivision in groups and it was not needed for all to attend. Also another answer indicated that it depended on the type of meeting.

In question Q4.9 participants of the survey were asked *"Do you modify the frequency or duration of the meetings during the length of the project?"*. As seen in Figure 8.36, 71% replied yes. Participants could add an explanation in a following open text field. They stated that as the competition gets closer the meetings become more frequent. It is also mentioned that depending on the progress (good or bad progress) and the topic to be discussed, the meeting duration may change.

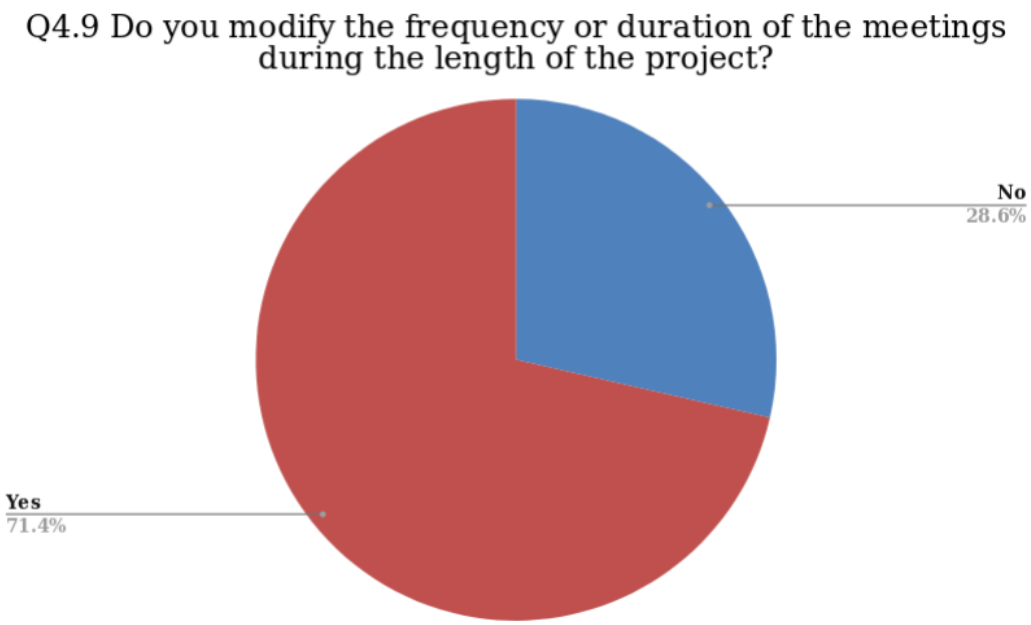


Figure 8.36: Participants were asked if they modify the frequency or duration of the meetings.

Finalizing this section about meetings, the question Q4.10 *"Add here any other comment you may have about your meetings style that may not have been reflected in your previous answers."* had 16 responses. These are summarized in the following points:

- Meetings include testing reported in 6 responses.
- Meetings include code review or programming support in 6 responses.
- Different types of meetings reported, including informative meetings, programming meetings, testing meetings, loosely structured meetings, well structured meetings, meetings one to one.
- Instead of meetings 4 responses indicate "team evenings" or "team days" to work on topics related to the team: planning, coding, testing, etc.

8.3.3.5 Practices from Software Development Methodologies

The first question on this section is Q5.2 stating *"Do you follow any of the following Software Development Methodologies?"* and allowing the choices of: Waterfall, Scrum, Kanban, Rapid Application Development, Lean, eXtreme Programming, Agile (Other), DevOps and other. The answers are distributed as shown in Figure 8.37.

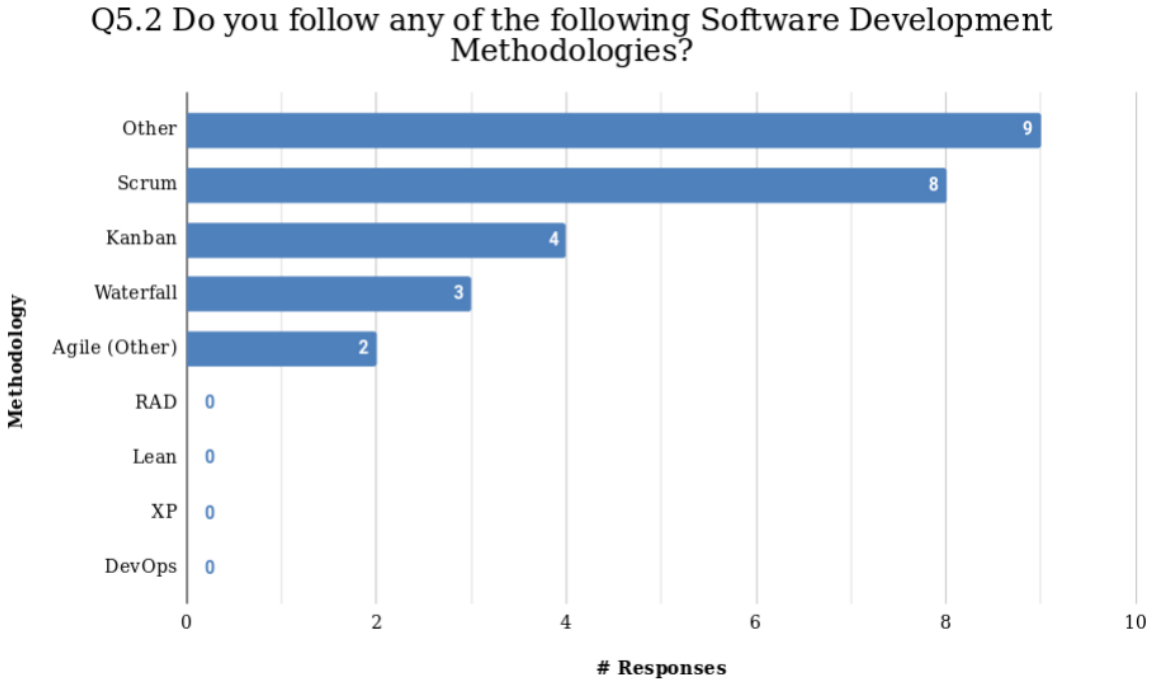


Figure 8.37: Most participants responded Other and Scrum. The next most frequent responses were Kanban, Waterfall and Agile (Other).

For the 'other' option, the related text field included the following summarized comments:

- Standard software development practices as seen in DevOps reported in 2 answers while also referencing resemblance to Agile. DevOps practices reported as: branch-based development, auto-formatted code, issue tracker, reviewed Pull Requests, automatic builds, linters and end to end tests.
- Three responses saying no methodology at all.
- Mix between Agile and DevOps with an initial Waterfall-like phase in one response.
- Spiral development mentioned in one answer.
- Alternate focus between theoretical (theory, reading, planning, etc) and real-world activity (implementation, testing concretely) in one response.

When asked on Q5.3 if *"Do you follow this methodology strictly?"* the responses, as shown in Figure 8.38, the majority (84%) answered that they did not follow their methodology strictly.

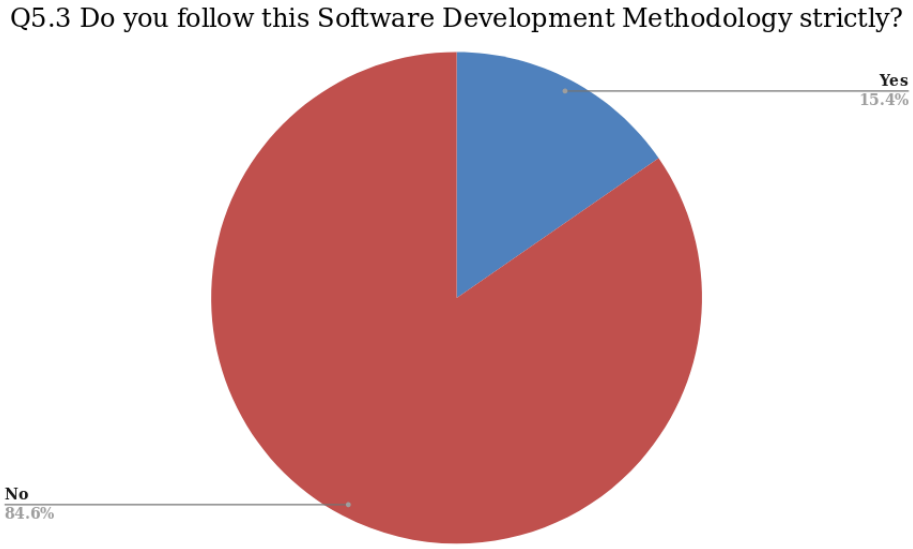


Figure 8.38: 84% of respondents said that they did not follow their methodology strictly.

On the text field to explain why they answered no, the summary is:

- 7 responses mentioned that they cannot be too strict with following the methodology.
- 6 responses mentioned that they needed to adapt to the needs of the team and to the available resources.
- 4 responses mentioned that they did not use some elements of their methodology.
- 2 responses mentioned that they tried to use it but after some time the use fell off.

The following question Q5.4 stated *"Do you use some kind of backlog for the tasks that need to be done during all the project length?"* and the answers can be seen in Figure 8.39. Most responses stated that they used some backlog for this purpose.

The next question Q5.5 stated *"Do you use some kind of backlog for the tasks that are currently being worked on?"* and the answers can be seen in Figure 8.40. Most responses stated that they used some backlog for this purpose.

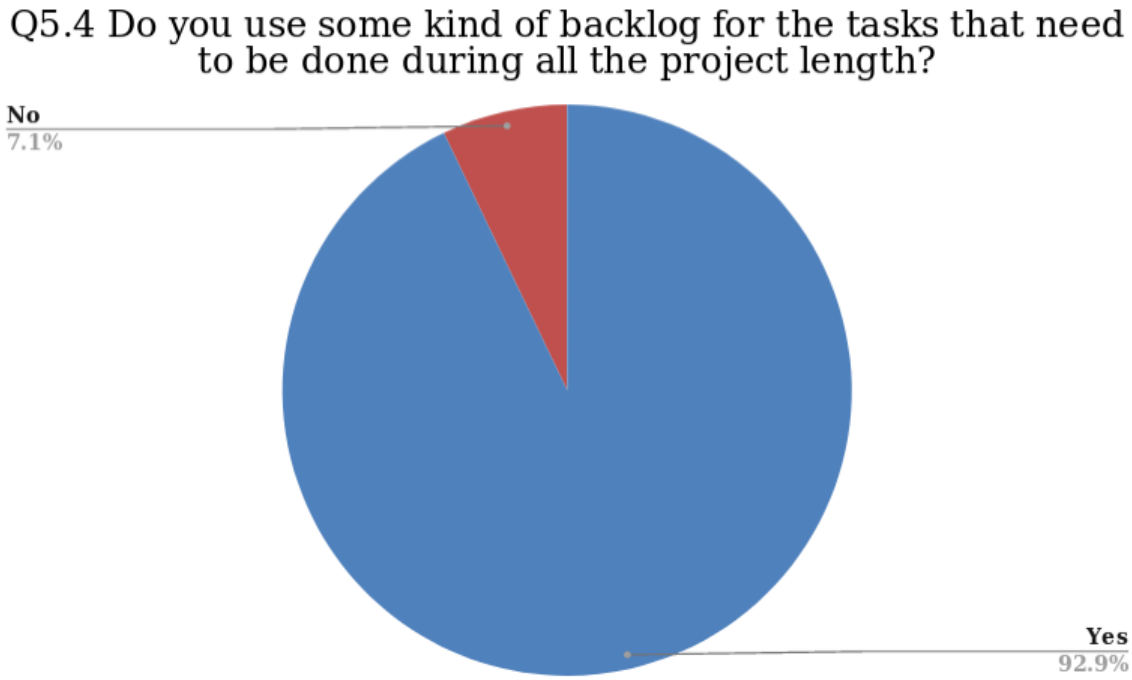


Figure 8.39: Most responses for *"Do you use some kind of backlog for the tasks that need to be done during all the project length?"* state that they used some kind of backlog.

When asked on Q5.6 *"Where do you store these backlogs, if any. (For example, Trello, or post-its on a whiteboard)"* most participants reported some kind of online means of storing them. Online resources that were reported:

- Github/Gitlab issues.
- Trello.
- Google Drive.
- Notion.
- Yammer.

From these online resources, GitHub and GitLab with their integrated issue trackers, were mentioned by 10 responses. Trello was mentioned by 8 responses too. Additionally, 10 answers mentioned some physical form of backlog, including: a whiteboard, post-its on a board, and notebooks. Respondents used these tools in combination.

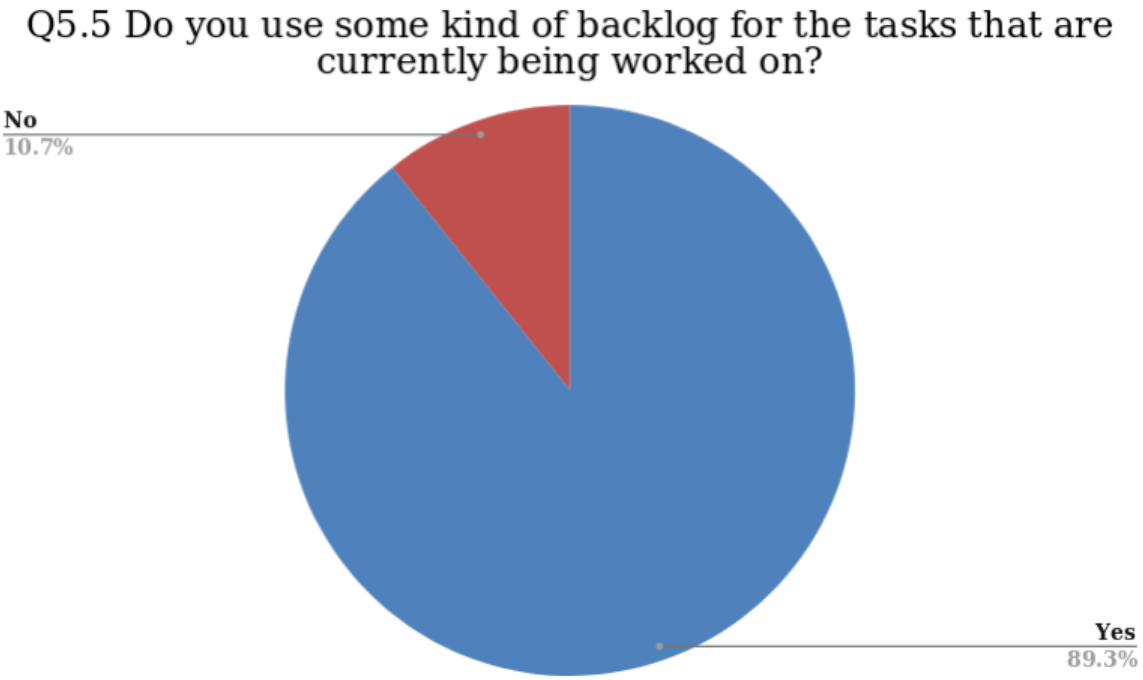


Figure 8.40: Most responses for *"Do you use some kind of backlog for the tasks that are currently being worked on?"* stated that they used some kind of backlog.

The following questions were Q5.7 and Q5.8 and stated:

- *"Do you work in 'sprints' as in the Scrum methodology? (A 'sprint' can be defined as a defined time period for developing features for a product).".*
- And as follow up question for positive answers: *"How long are these sprints? - Length of sprint in Weeks".*

The answers are summarized in Figure 8.41. Only 35% of the responses indicated they worked in sprints with the distribution of these indicating most sprint lengths were between 1 and 2 weeks.

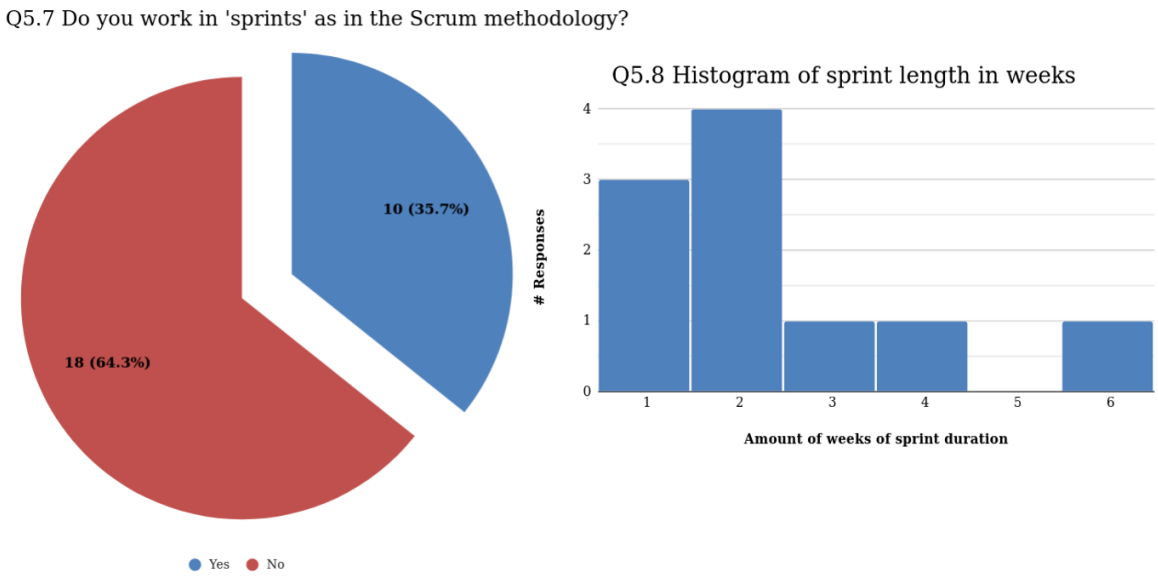


Figure 8.41: 35% of respondents to *"Do you work in 'sprints' as in the Scrum methodology? (A 'sprint' can be defined as a defined time period for developing features for a product)."* indicated that they worked in sprints. From those most had a sprint duration of between 1 and 2 weeks.

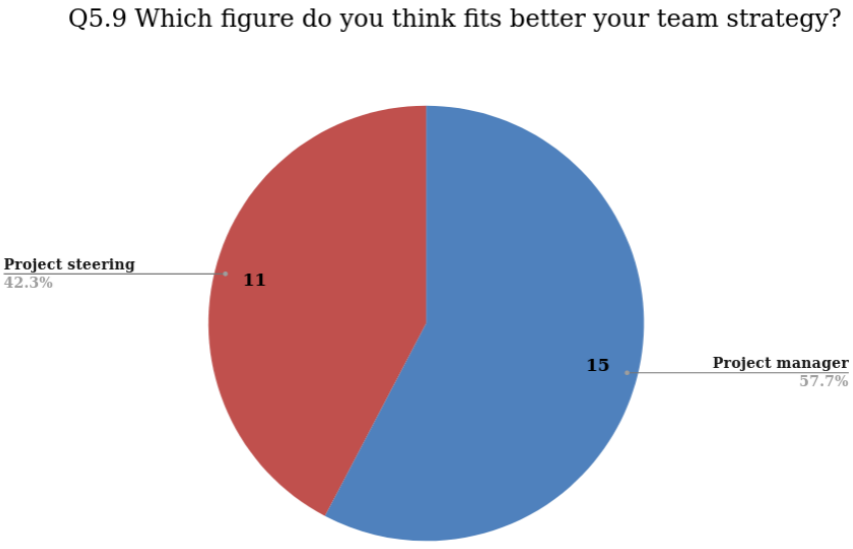


Figure 8.42: Responses to question Q5.9.

Question Q5.9 *"Some previous works (Lightweight Management - Taming the RoboCup Development Process & Elements of Scrum in a Students Robotics Project - A Case Study)*

on managing a RoboCup team talk about the difference between having someone acting as a project manager or having someone steering the project. Which figure do you think fits your team strategy better:", with its responses in Figure 8.42, just over half of the responses (57.7%) indicated the project manager figure.

However, when asked in Q5.10 if *"Do you have a project manager figure in your team?"* we find that that most (82.1%) participants do have a project manager figure as can be seen in Figure 8.43.

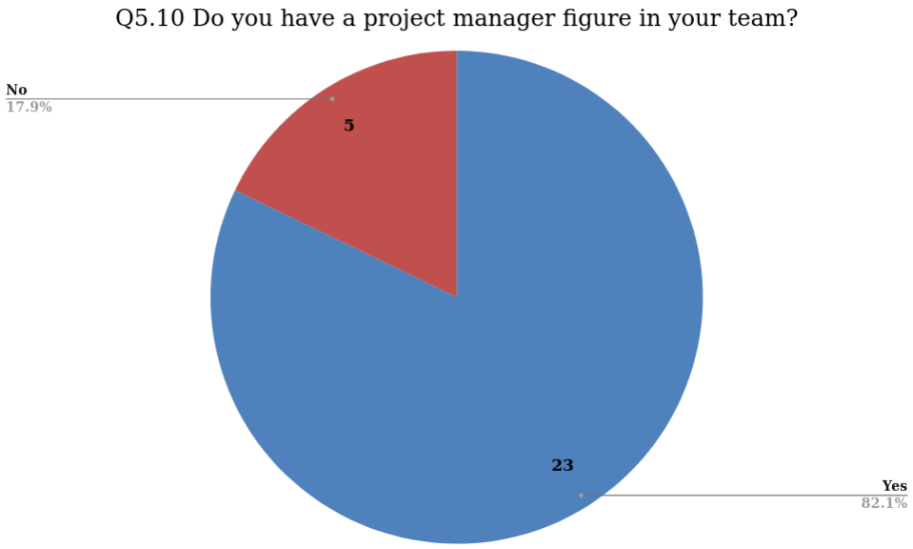


Figure 8.43: Responses to question Q5.10.

Question Q5.11 stated *"Do you hold specific planning meetings? How do they look like?"*. The responses, as seen in Figure 8.44, have a slight majority (59.3%) stating that they do not. Further explanation (11 responses) on what those meetings look like were (summarised):

- The planning meetings are used to discuss milestones and timeline was mentioned in different ways in 4 responses.
- The planning meetings are used to generate issues and tasks to work on.
- One answer mentioned only done when needed and with only the key members
- Two answers mentioned to hold planning meetings only at the beginning to review rules and motivations of team members.

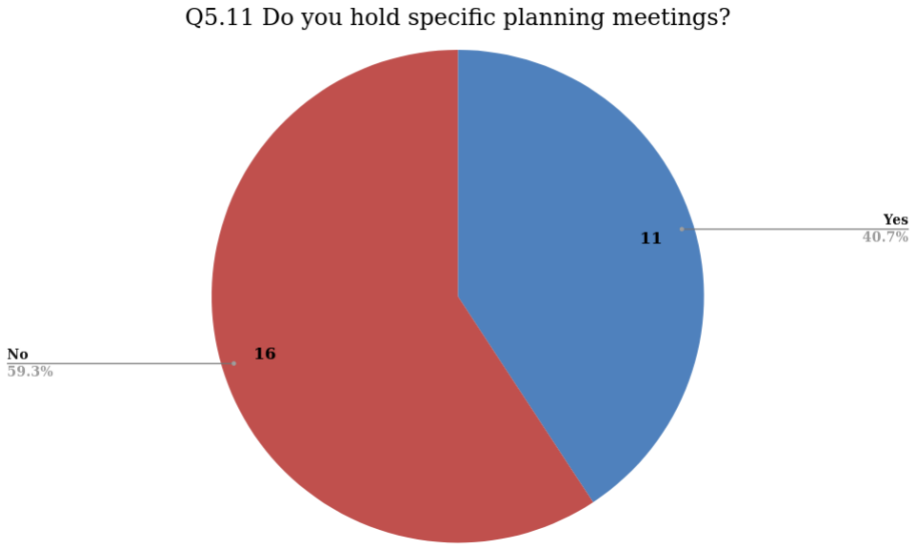


Figure 8.44: Responses to Q5.11.

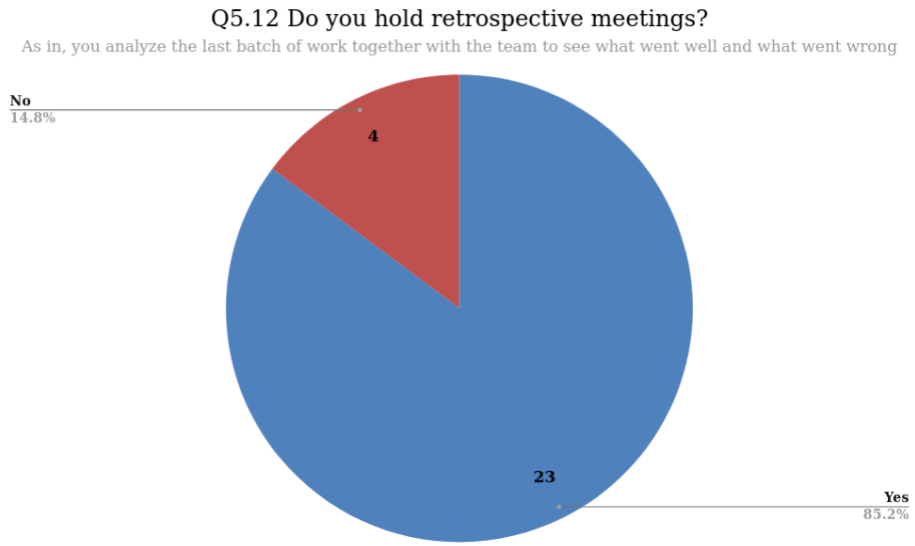


Figure 8.45: Most respondents (85.2%) stated that they did hold retrospective meetings.

The following question Q5.12 is *"Do you hold retrospective meetings? (As in, you analyze the last batch of work together with the team to see what went well and what went wrong)"*. The answers are summarized in Figure 8.45.

The last question of the section is Q5.13 *"How important do you think is it to hold retrospective meetings?"*. Participants responses were distributed as shown in Figure 8.46.

Most answers were between moderately important and extremely important, with a peak on Very important.

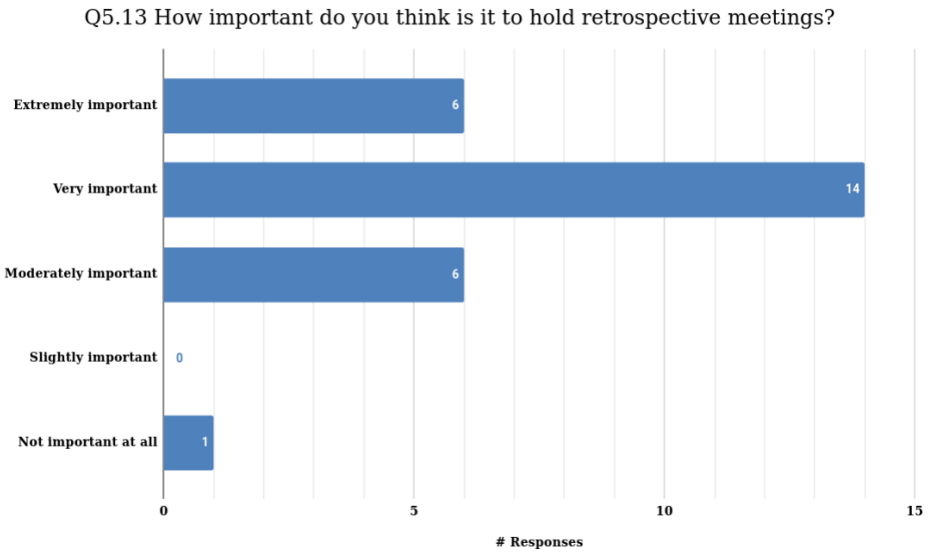


Figure 8.46: Answers to Q5.13.

8.3.3.6 Coding Practices

The first question about coding practices is coded as Q6.2. The question states *"Please order these concepts by importance to address them for the success of your team in the competition."* with the options: *"Naming conventions, Commenting code, Code simplicity, Code portability, Unit testing, Continuous integration, Pair programming, Shared code ownership"*. The question is shaped in this manner to force respondents to make trade-offs between these concepts. Previous iterations of this question just asked about the individual level of importance of these concepts as a trial survey showed that the concepts were mostly just considered very important. During the workshop it was also discussed that while every one of these concepts is desirable, there is a trade-off between investing time and energy in some of them rather that in another.

The following question was a text field for participants to optionally provide further explanations.

Every response ordered the concepts from 1st to 8th. For every concept, the average of the responses was found and it was converted to a 0 to 10 score scale. Other approaches were tested and they provided the same order of importance between the concepts. The scores can be seen in Figure 8.47.

Code simplicity, commenting code, and naming conventions head the importance of these concepts. Code portability fits in between in the third position. It seems like having easy to read code is considered very important as these three related concepts head the importance ranking. It is noteworthy that code simplicity had the most amount of responses, making it the most important concept (10 over 28 responses).

Shared code ownership is an interesting concept as its the only one to present a conflicting distribution of responses with 4 responses classifying it as the most important and 11 classifying as the least important (the concept with the highest number of least important responses). Differences in the way to manage the codebase may have to do with this.

Continuous integration falls in the middle of this importance scale, with unit testing even lower. It appears that, compared to having easy to read code they are less important, even though in the workshop it was generally considered that they were very important practices. Maybe teams take the quality of the code for granted.

The last concept was pair programming. Having easy to read code and an automated environment in which to build code and catch bugs, seems to overshadow this concept.

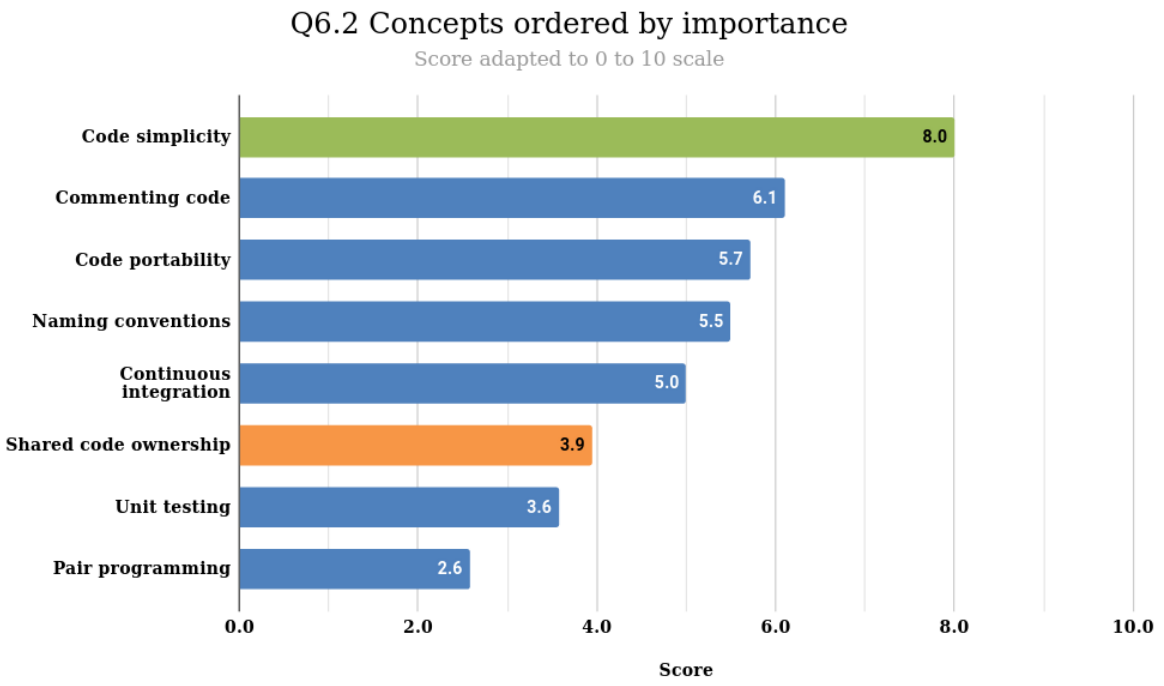


Figure 8.47: Concepts ordered by importance, on a score from 0 to 10. Code simplicity was considered the most important. It's also marked in green to denote that it's the concept with the most responses, marking it as the most important (10 responses over 28). Commenting code follows as the next most important. Code portability and naming conventions follow closely. Then continuous integration. The next concept is shared code ownership, marked in orange to note that there is some conflict in this item. This is because 4 responses classify it as the most important and 11 classify it as the least important, becoming the item with the highest number of least important votes. Unit testing follows closely behind shared code ownership. And finally pair programming has the least importance.

A set of 9 comments were left in the following question in Q6.3, these have been summarized here:

- 2 responses added that good and complete documentation is important.
- 2 responses added that clear interfaces and well understood architectures are important.
- Human factors are more important than specific technical practices was said by one respondent.

- One response added that they have not been disciplined with any coding standard other than some unit tests.

- One response added that code reviews to increase the quality of the code is important.

- One response criticized the question itself considering the concepts incommensurable. While this is fair criticism, the question arose from looking for a way to compare the trade-offs between these concepts.

- One response was *"Something is missing for us. We follow a rule very strictly using event based programming only and exclusively with many independant projects."*. Which is hard to generalize with the provided concepts.

Question Q6.4 *"How important is the concept of dependency hell in your team?"* had the distribution of responses as seen in Figure 8.48. Most responses classified dependency hell[2] in between moderately important and extremely important.

Dealing with a sane system configuration avoiding dependency hell seems important for most respondents.

Question Q6.5 states *"Which types of testing do you perform in your team? Note that if you aren't familiar with some of them, just leave them unchecked."*. The set of possible options was taken from an informal chat with a few experts in RoboCup taking a large list of types of testing and selecting the ones that were familiar at least for some of us. An option of 'other' was left in case any were missed.

[2]Dependency hell is a concept that can take several shapes. A software package could have many dependencies, long chains of dependencies, conflicting dependencies, circular dependencies or diamond dependencies. These problems can cause a lot of trouble to address. Namely, for a full ROS installation up to a thousand software packages may be needed. Many leagues in RoboCup make use of ROS.

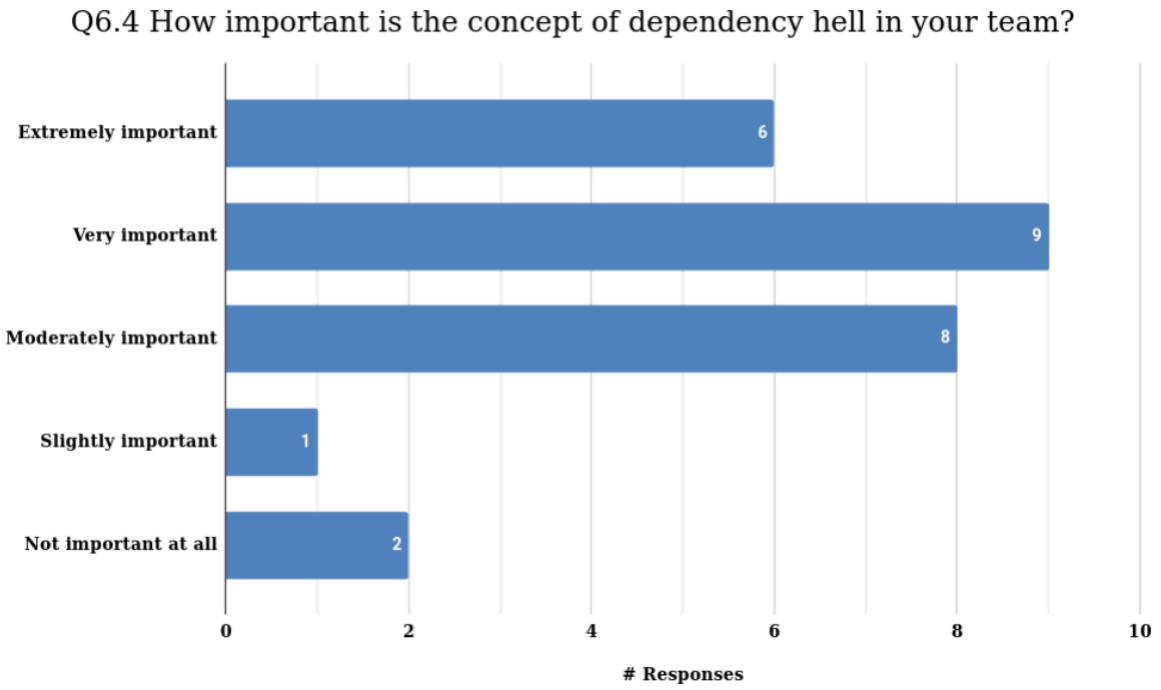


Figure 8.48: Most answers to *"How important is the concept of dependency hell in your team?"* fall in the range of moderately important to extremely important.

The distribution of answers can be seen in Figure 8.49. The most performed testing types were integration testing and ad-hoc testing. This is unsurprising given the large amount of software that needs to be integrated in most RoboCup competitions.

Unit testing follows closely, with one less respondent. It is a widely used technique, and this seems unsurprising to the author too.

System testing and functional testing are next with 12 respondents marking them. This seems aligned to the need for doing a lot of integration testing.

Performance testing follows showing the need for tuning the systems for specific platforms in many RoboCup leagues with limited computing capabilities.

For the two 'other' types of tests the responses were:

- *"Sadly we do not test much at all. Starting with unit- and install testing for some software modules at the moment."*.
- *"Demos and Integrated test in real world."*

The question Q6.6 stated *"How important is the usage of end to end testing in your team? (Consider end-to-end testing as in running a full test / mission of your competition)"*.

The answers are summarized in Figure 8.50.

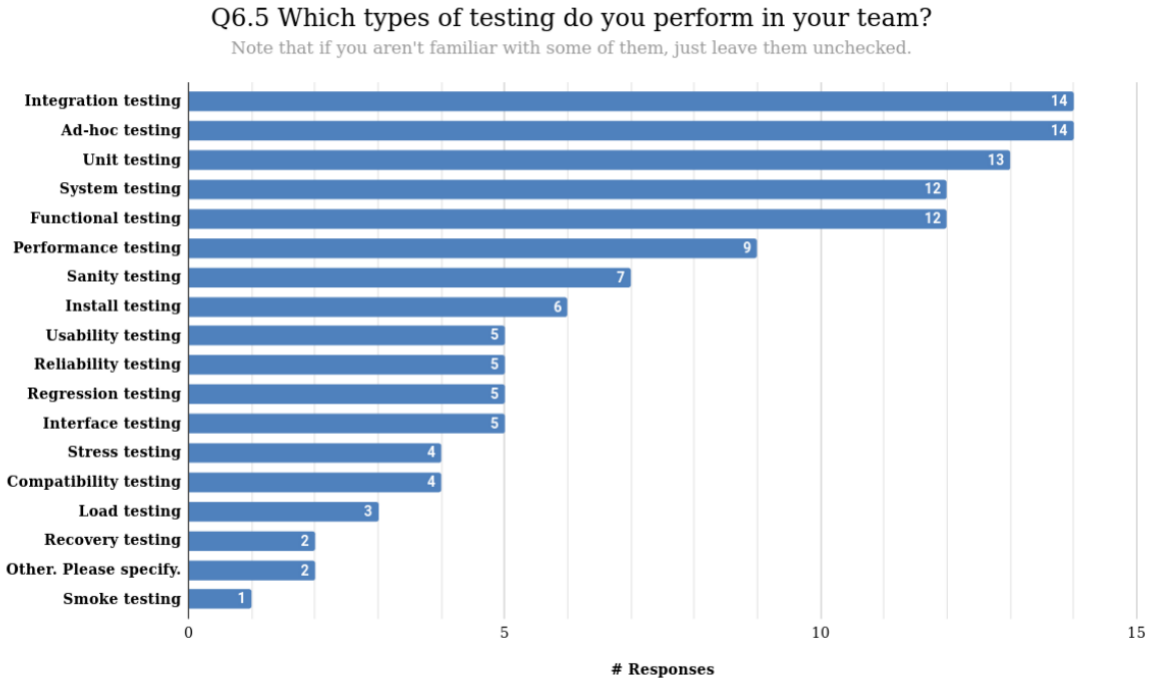


Figure 8.49: The most performed types of testing were integration and ad-hoc testing, followed closely by unit testing, system testing, and functional testing. From there the quantity of responses decrease.

Q6.7 follows with *"Do you use Pair programming in your team?"*. There is a match in between yes and no as shown in Figure 8.51.

From the people that answered yes, everyone added an extra comment in the following text field. These comments are summarized here as:

- Pair programming used to share knowledge, added by 8 responses.
- Pair programming used sometimes, depending on the people themselves, added by 4 responses.
- Pair programming used to have more than one responsible person for some item, added by 2 responses.
- Pair programming used for teaming up to prepare specific tests, added by 2 responses.
- Pair programming used to introduce people to the project, added by 2 responses

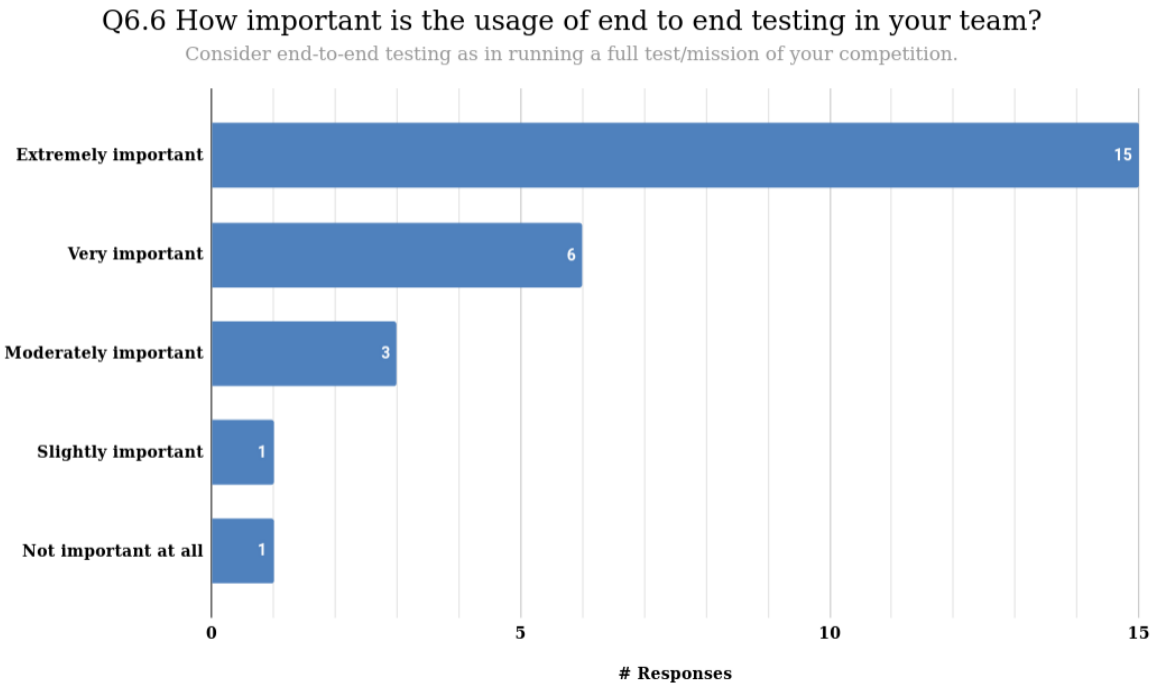


Figure 8.50: Responses for *"How important is the usage of end to end testing in your team? (Consider end-to-end testing as in running a full test / mission of your competition)"*. Most answers considered it extremely important.

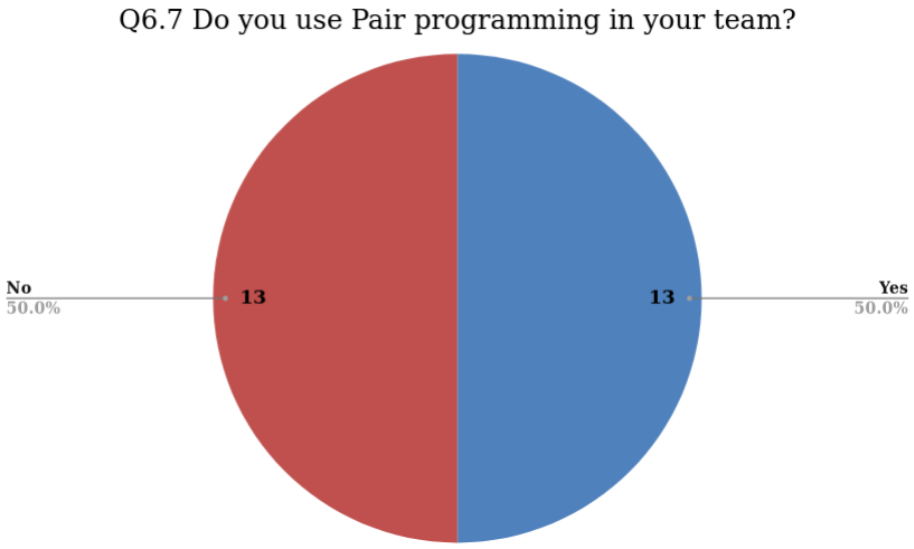


Figure 8.51: Exactly half of the respondents to *"Do you use Pair programming in your team?"* answered yes.

For the last question of the section Q6.8 *"How important is the choice of programming language?"* the distribution of answers can be found in Figure 8.52. Most answers were in the range in between very important and moderately important.

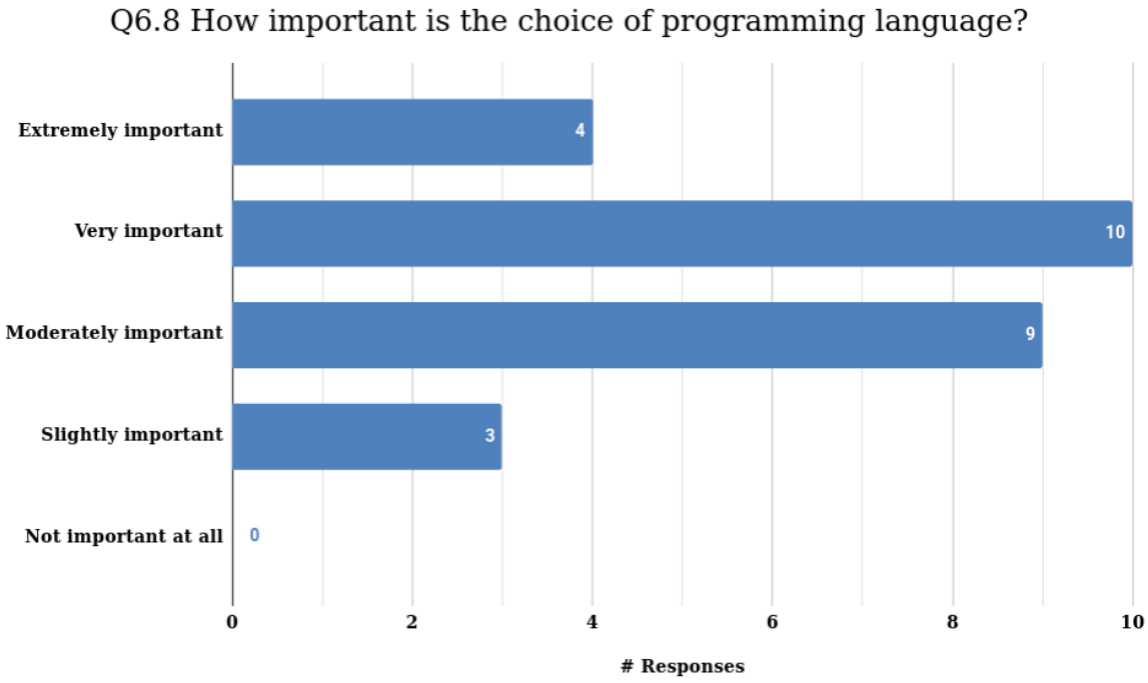


Figure 8.52: Most responses to *"How important is the choice of programming language?"* were in the range in between very important and moderately important.

8.3.3.7 Outcomes: Research and Competition

Noteworthy, this section starts with the question Q7.2 *"Please summarize your team outcomes from your participation in the RoboCup major league competitions. (As in, which place did you achieve and which awards)."*. An analysis of the top performing teams and their answers on the rest of the survey is to be found further in the chapter in subsection 8.3.4.

For this question the count of respondents that attained first place in some year of participation in a world RoboCup competition instance was 25. For second place, 16, for third place 10, and for fourth place 6. Many other first, second and third placed results were also provided for regional RoboCup opens (German Open, Iran Open) and some other competitions. This number of first to fourth places reported is more than enough to consider that this survey has a set of strong participants with a long story in RoboCup.

Q7.3 states *"Which were your research outcomes from your participation in these Robotics Competitions? Please describe them. (As in, did you publish papers from it? How many? Did they have a big impact?)"*. A summary has been made based on the provided comments. The comments included lists of publications, specific amounts of publications, generalizations as "some publications" or "many publications" as well as others. The summary of comments follows:

- 11 responses were considered to express "some publications".
- 5 responses were considered to express "many publications".
- 2 responses were considered to express "very few publications".
- 3 responses reported to have indirect publications, e.g. by using the software created for RoboCup to create experiments unrelated to RoboCup.
- 2 responses reported that publishing in "serious" or "big impact" venues is hard from RoboCup related work.
- 2 responses indicated that RoboCup provided content (including publications) for PhD and Masters thesis.
- 1 response indicated that they do not aim to publish research from RoboCup participation.

The following question Q7.4 stated *"Was there any other outcome from participating in these Robotics Competitions? Please describe them. (As in, was there a product created? A company or spin-off? Did team members improve their employability?)"*. The responses have been summarized in the following list:

- 12 responses considered that future job prospects improved (giving examples of students going to work into companies like Google, Toyota, Honda, Uber, Microsoft, Amazon...).
- 5 responses reported that team members created spin offs or startups from the experience in RoboCup.
- 5 responses reported that RoboCup provided the students with valuable hands-on experience.
- 3 responses reported that RoboCup created the opportunity and interest for students to start a PhD.

- 2 responses reported RoboCup provided media attention.

- 1 response reported that the software developed in RoboCup powered other research.

8.3.3.8 Experiences in Specific Situations

The first question Q8.1 stated *"Have you found yourself in a situation where a team member was not doing their part of the work? How did you deal with the situation? Was it resolved? Please describe it"*. The summary of the responses follows:

- 11 responses reported that talking about the issue with the team member was the first step. Some responses reported to also talk about the issue openly with the rest of the team.

- 11 responses reported that the team member was either actively dropped out of the team or the team member dropped out themselves. This may happen after a different number of approaches to improve the situation depending on the team.

- 5 responses reported to provide this team member with non critical tasks if the situations didn't improve.

- 1 response reported to add precise goals and deadlines to try to improve the situation.

- 1 response reported to pair the team member with some more experienced member doing pair programming to improve the situation.

- 1 response reported using a custom "karma system" and having a minimum amount of hours to invest in the project. Appraisal is given to the team member contributing the most and other team members can measure their amount of work against others. It is reported that such a team member will either get motivated to keep up or realize they do not have enough time.

- 1 response reported that the criterion to apply is different, quoted: "It is not *did he/she contribute as much as others?* but *did he/she contribute positively to the project?*. Experience shows that the criterion is always met, often with surprising circumstances".

It seems like this issue appeared often in the teams, and that it is approached in steps to try to improve the situation. When the situation did not improve as much as the teams would like, different paths seem to have been taken. Either the teammate was dropped out of the team or assigned non critical tasks that were "good to have". Some teams had a stronger focus on keeping the work output high and others to have the students have a positive experience.[3]

The following question, Q8.2, states *"Have you found yourself in the situation where a single team member had the knowledge on how a vital part of your system worked? How did you ensure this was not a problem? Was it resolved? Please describe it."*. The summary of the responses follows:

- The most repeated approach was to dedicate more time to sharing knowledge in the team with 7 responses explicitly stating that.
- Also 7 responses reported addressing it by improving their documentation.
- 1 response added that they held events a couple of times a year as documentation days where the team gets together to add or improve documentation.
- 4 responses reported taking advantage of GitHub/GitLab features to address this by making use of the commit history and Pull Requests and Code reviews.
- 2 responses reported that they needed a self motivated team member to take over the system.
- 2 responses indicated the use of pair programming to address this issue.
- 2 responses reported the use of coding style and coding conventions, and with them, automated checks of those to increase readability.
- 1 response mentioned to have a team leader take over the system to transfer knowledge.
- 1 response mentioned to make the system very modular.
- 1 response mentioned to rotate every team member through every task.
- 1 response mentioned to try to make the team bigger.

[3]I must make clear that I'm not saying that these are contrary views here. From the responses it can be interpreted the overall spirit is to provide positive experiences to the team members, but some teams may provide more effort in keeping a team mate even though they provide less output than expected.

- 1 response mentioned to have a redundancy of solutions for the different parts of the system.
- 1 response mentioned to rewrite the system.
- 1 response mentioned to ensure to have at least 3 team members with knowledge of every system part.
- 1 response mentioned that it is addressed by having a good architecture.

The next question states *"Have you found you had a new team member that lacked a lot of necessary basic skills to contribute to the project? How did you ensure he learnt what was needed? How did you make the best use of this person efforts? Did this person come up to speed? Please describe the situation"*. The answers have been summarized in the following list:

- To provide simpler tasks, by 11 respondents.
- To team up with more experienced team members, either in pair programming or in a subteam to work on a part of the system, by 7 respondents.
- Tutorials or coursework provides the basic skills needed, by 5 respondents.
- Create tasks that focus on the interests of the team member, by 4 respondents.
- Create tasks that are not critical work, by 4 respondents.
- By not letting students join the team if they do not have enough knowledge, by 3 responses.
- By providing alternative tasks to coding, by 2 respondents.
- By talking through the issue with the team member. This is explicit in one answer but could be understood implicitly in more answers.

It is worth noting that a few responses argued that ROS was the only requirement. The reasoning is reported to be related to the advantage of the system being modular when using this middleware. Additionally, another team reports the same advantage with their custom middleware. A few answers either explicitly or implicitly stated that education is the first goal for them.

For Q8.4 which stated *"Please explain how do you approach succeeding both in the competition and also getting research outcomes. Do you think there is room for improvements in your approach, which?"* the summarized answers are:

- To approach RoboCup competition challenges research first. Create systems that tackle those challenges in a principled way, not with hacky solutions. The spirit of this answer was shared by 5 respondents.
- To have more Professor or PhD team members to have a stronger research orientation was shared by 5 respondent.
- A few perspectives about time management were provided. Reducing time for practical RoboCup solutions and moving it into research, investing more time into research, or just investing more time in general were shared by 4 respondents. It is worthy of note that one respondent added that just investing more time was not healthy.
- 3 respondents shared that they can't do both competition outcomes and research. They either focus on one or the other.
- 2 respondents added that the rulebook of the competition should be improved to approach this question.
- 1 respondent replied that they identify what is publishable after the competition.
- 1 respondent replied to have one team member focused on research and writing.
- 1 respondent added that having the work done by team members be their goal for a Bachelor or Master thesis.

The last question, Q8.5, states *"Which do you think is the biggest challenge regarding the software development process for a RoboCup competition?"*. The responses are summarized here:

- The most repeated challenge reported is to create software robust in all situations, resisting variability and uncertainty of the real world. 7 responses included it.
- Grouping together concerns about teamwork, team management, ever-changing team members and new team members integration, the concept of team management is repeated in 5 responses.

- The concept of integration of the different software parts is mentioned in 5 responses.
- Knowledge transfer, between team members and from previous teams or code bases is mentioned in 4 responses.
- The complexity of the competition is mentioned also in 4 responses.
- Having a huge software stack is mentioned in 3 responses.
- Testing all the software is mentioned in 3 responses.
- Having a good software architecture is mentioned in 2 responses.
- Keeping the software architecture conventions is mentioned in 1 response.
- Automating workflows as Continuous Integration, Deployment and Testing is mentioned in 1 response.
- Time being the biggest challenge is mentioned in 1 answer.
- Bad robot hardware from a standard platform is mentioned in 1 answer.

8.3.4 Top 9 Teams Review

In this section we will review the answers of the top 9 teams that answered the survey to find insights. These may be insights on when they all seem to answer in similar way or in a conflicting way. Not every question will be discussed as some questions simply didn't provide any further insight from what was found from analyzing the full set of answers.

To choose the top teams a ranking was performed based on their reported achievements in RoboCup. A simple heuristic of giving a higher amount of points for 1st place to 6th place (the lowest position reported) taking into account the last 5 years. Care was taken to not exclude any high performing team because of this heuristic. The top list ended up having 9 teams. It may be an unusual number, but it is so based on two facts: on one hand, two high performing teams had 2 answers for the survey each, the most complete answer was taken from those while checking that they were aligned in their responses. On the other hand, the following team (10th) in this heuristic had one single podium position (while the upper 9 had 2 or more) with just 2 years of participation in RoboCup (the average being 8 years).

Other heuristics were tested and they either ended up with the same list or were harder to reason about.

8.3.4.1 Participants Profile

These 9 teams have a span of years of experience from 3 to 21 years. The average amount of years of experience is 8 years.

The teams achieved multiple top three positions in, at least, the last five years of RoboCup participation. The leagues they participated in were: seven teams in RoboCup@Home variants, one team in RoboCup Standard Platform League, and one team in RoboCup Logistics League.

When observing their last year team composition in regards of distribution of PhD, master, undergrad, and other, comparing it to their optimal team composition, a wide set of approaches is observed as can be seen in Figure 8.53.

The amount of team members of each answer stays similar between ideal and last year team composition for every answer. Team composition consistency between ideal and last year is up for debate.

When observing the answers to the rationale behind their team composition, the answers stay in line with the ones found when the full survey is taken into account. The team composition should take into account:

- Having experienced team members, which implies keeping team members for as many years as possible.

- Chain the knowledge transfer from more experienced up to the newcomers, usually from PhD to Master to Undergrad students.

When asking what programming languages do these teams use, everyone answered that they used C++. All but one also said Python. It is noticed that two teams also added CLIPS for their rule based systems.

Most possibly being influenced by the previous question, all teams reported the usage of ROS with two teams stating it was used in parallel with their own middleware.

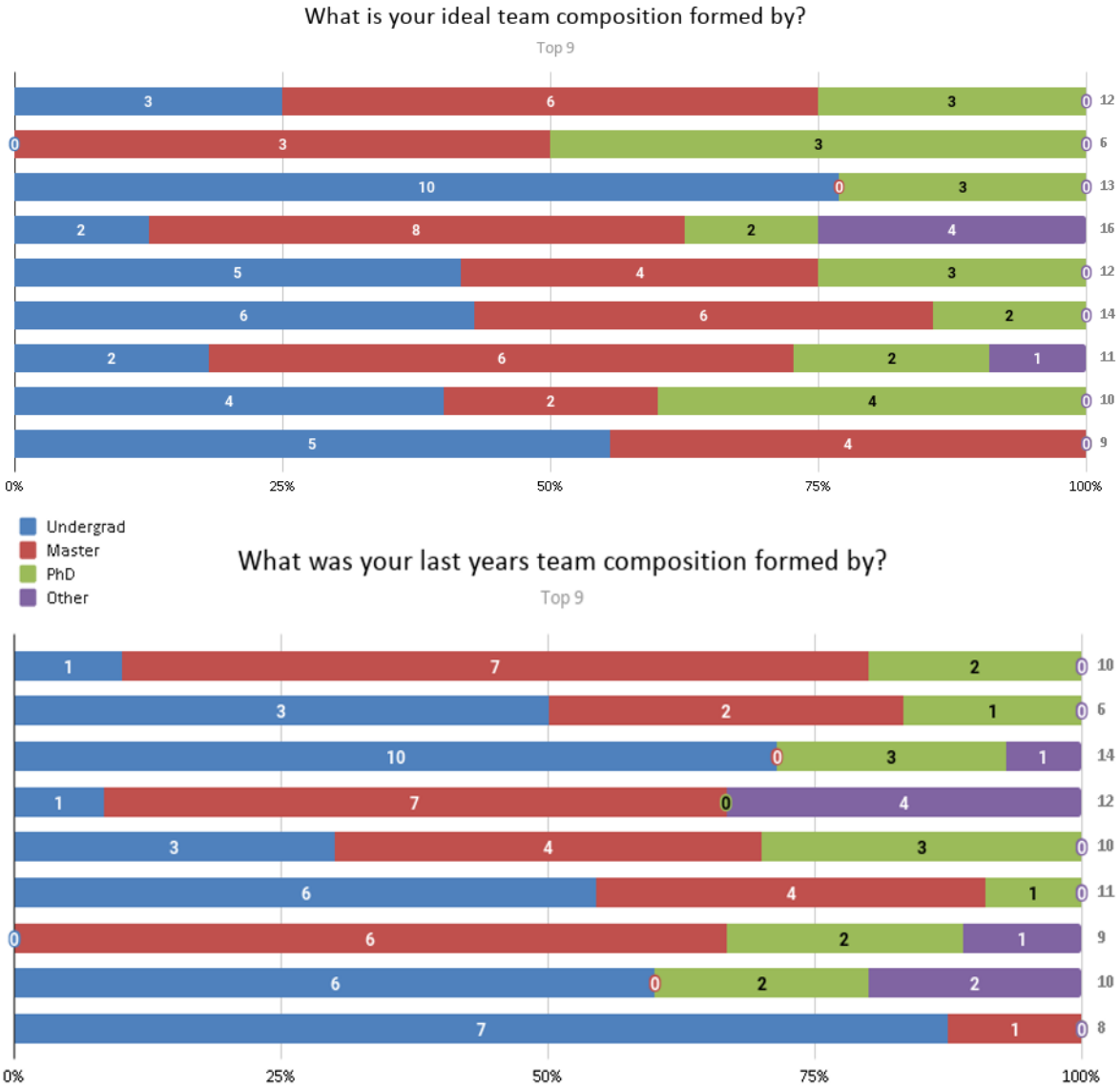


Figure 8.53: Comparison of team compositions for the top 9 teams: ideal team composition and last years team composition. The same row pertains on both plots pertains to the same survey response.

8.3.4.2 General Questions

When asking how many person-hours are estimated as needed to successfully participate in a RoboCup competition the answer of the top 9 teams have been summarized in Figure 8.54.

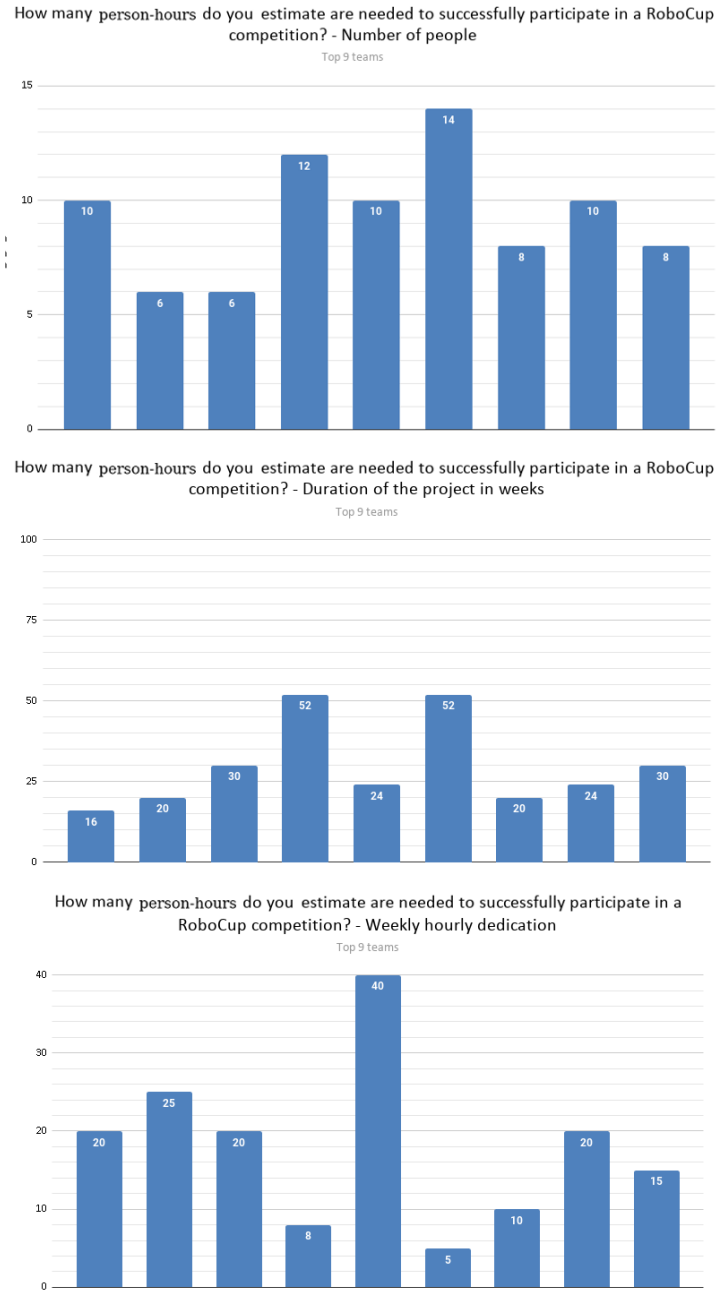


Figure 8.54: Top 9 teams answers to *"How many person-hours do you estimate are needed to successfully participate in a RoboCup competition?"* with the parts of the answers: number of people, duration of the project in weeks, and weekly hour dedication.

Different approaches can be observed. From full-time for a relatively short duration of weeks to a low amount of hours a week for a long time with a big team. This may be a topic dependent on the profile of the team.

If we make an approximation in the same spirit as done with all the available answers, we get a team of 9 people, working part-time (20h/week) for 9 months. Using all the previous data, we got 8 people, working part-time for 7 months. This suggests some additional work is needed.

When checking if these teams had a practice environment like their competition, all of them reported having an environment as close as possible to the real competition environment.

In Figure 8.55 the summary of answers to *"When recruiting new team members which skills do you find more important for the success of the team?"* can be found. While the general results are similar, there is a peak on proactivity and a bigger drop of importance towards competition-specific experience and robotics knowledge compared to the results of this question with all data.

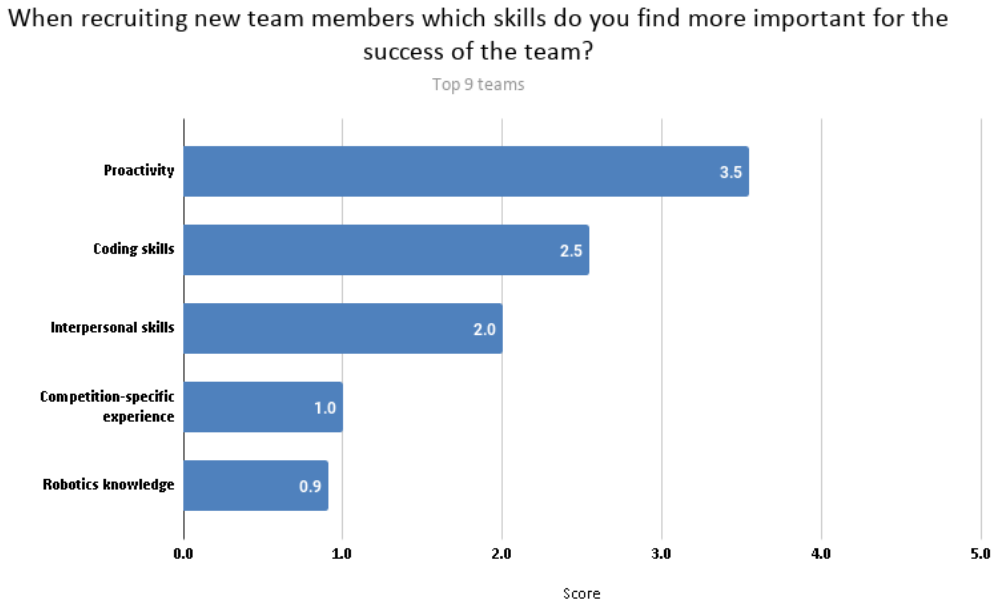


Figure 8.55: Answers by the top 9 teams to *"When recruiting new team members which skills do you find more important for the success of the team?"* are similar the previous results.

When analyzing *"How important is to have an always fully working version of the software of your robot at any time?"* the conclusion is that it's between very important (two answers) and extremely important (7 answers).

Following with how important automated software and hardware checks are, there isn't a consensus on any of them. All answers are over slightly important, so it could be

interpreted as just "good to have".

Finally, for this section, the teams responses on how important having a simulation for their robots is had a wide range of answers indicating this is very team dependent.

8.3.4.3 Recruiting

Responses for methods on how to recruit new team members, informative meetings was the most repeated method. Others varied in usage.

When looking at the incentives provided to participate in the team, all top teams had Future job prospects as an incentive. Most (6 of 9) also stated that having coursework done by participating in the team was also an incentive.

To teach the basics needed to participate in the project, all coincide in the use of Personal work and Supervised personal work. This may relate well with the wish of having proactive team members.

Where these teams do not agree, as responses were very varied, is on how long it takes for a new team member to make a contribution to the code base of the team. Very dependent on the student is the only repeated answer.

8.3.4.4 Meetings

Most of these teams (8) report meeting once a week for an average of 60 to 90 minutes. One team reports fortnightly. But most teams (7) report that they modify the frequency and duration of meetings as the competition gets closer. Seven teams aim to have everyone that can make it attend the meetings. Two specify that not everyone is expected to attend all meetings.

8.3.4.5 Practices from Software Development Methodologies

Exploring the Software Development Methodologies reported by the teams we have seven teams reporting using some kind of Agile methodology. Within those, three report Scrum and two report Kanban. The other two report using a mix of what they feel works for them from Agile methodologies. Only one team reports following a methodology strictly, and they report to use Kanban strictly.

Two teams report no methodology or no explanation given.

When asked about using a backlog for the project (long term goals) and a backlog for current tasks, all top participants in the survey report using them both. Six teams report

storing these backlogs in GitHub or GitLab issues, three report using Trello. Four also report using a physical means to store these, reporting whiteboards and notebooks.

Most teams report not using the concept of sprints, only two answered that they do and with 3 or 4 week cycles.

When asked if they prefer a project manager or a project steering figure in the team to lead, responses are mixed. Four prefer the project manager and five the project steering figure. But when asked if they have a project manager, seven report having one.

Six out of nine do not hold planning meetings.

However, almost everyone agrees that retrospective meetings are very important, and they all hold them.

8.3.4.6 Coding Practices

When taking into account only the subset of 9 top performing teams, the question that asked to order by importance a set of concepts got different results. In Figure 8.56 we observe that code simplicity stays as the most important concept. But then, continuous integration becomes the second most important. The following items (naming conventions, commenting code and code portability) switch their positions, but overall stay similar in importance.

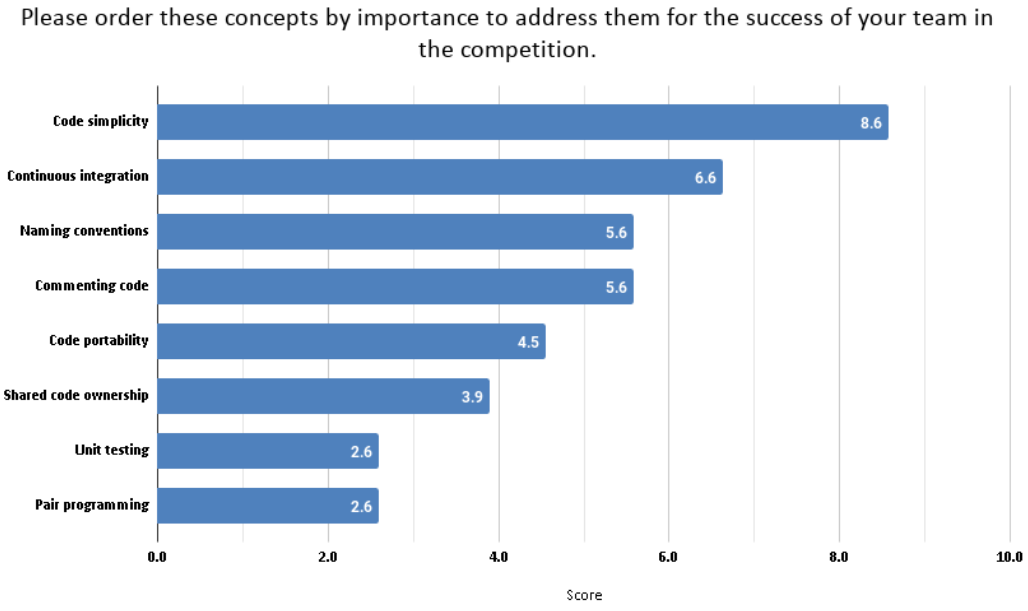


Figure 8.56: Top 9 teams order on the importance of these concepts related to coding practices.

The question about how important is the concept of dependency hell had different responses depending on the league of the team. For RoboCup@Home teams it was considered between moderately and extremely important. Meanwhile for RoboCup Logistics League the answer was not at all important.

For the types of testing that the teams practice, the summary of responses can be found in Figure 8.57. The six most responded types of testing stayed the same but with a slight variation in their order. Integration testing stays as the most important type of testing.

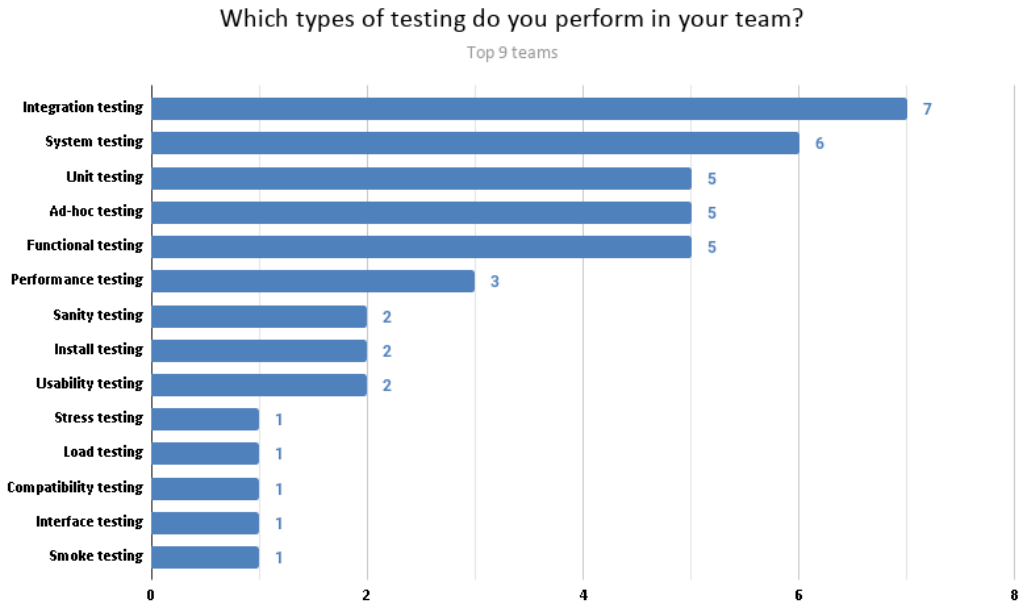


Figure 8.57: Types of testing used by the top 9 teams.

When asking specifically for the importance of end-to-end testing (as in, practicing full RoboCup competition tests) the consensus is that it's extremely important.

As for the practice of pair programming, eight teams responded that they use it and the three most repeated reasons to use it were: to pair members on a competition test, to guide new members and to fix hard problems.

This section also asked about how important the choice of programming language is. There is no consensus here as five teams report Very important, three just moderately important and one slightly important. The fact that all teams use ROS, C++ and almost all use Python may make this question not very meaningful as there seems to be less choice anyways.

8.3.4.7 Outcomes: Research and Competition

In this section we find repeated comments about publishing research based on RoboCup is done, but it's hard.

There were three other responses that were common to all Top 9 respondents (besides podium positions in the competitions and research outputs): team members are easily employed by big companies (like Google, Amazon, Microsoft, etc); many team members embark and finish PhDs; and a few team members create startups related to robotics.

8.3.4.8 Experiences on Specific Situations

A common issue between the teams was to have a part of their system that only one person had knowledge of or was an expert about. Solutions proposed included to pair the expert with someone to teach. To create a simple code and documentation. But these were reported as only helping. Ultimately someone motivated must make the effort to take over. This last point related to the wish of having proactive team members.

When these teams were presented the question on how to get a team member up to speed with their work, when they seem to be falling behind, the following comments were given and shared between different answers:

- Give them tinier tasks.
- Team them up with someone more experienced.
- Some teams report that education comes first even when contribution to the team is low. Other teams let the students that do not catch up drop out from the team.

When asked what the biggest challenge in RoboCup competitions was, the concept repeated by most answers was integration of software.

8.4 Conclusions

The conclusions of this chapter are formed by the most significant insights from the survey while taking into account the feedback from the workshop that shaped the questions in the survey.

8.4.1 Significance

To quantify the significance of the insights of the survey the profile of the participants is analyzed. 28 answers, 26 of those having completed every question in the survey, may not be interpreted as a number that can represent the whole large community in RoboCup. However, these participants of the survey have an average of 6.75 years of experience, with up to 21 years of experience and an accumulated 189 years of experience in RoboCup major leagues participation. Such an amount of experience makes their opinions relevant.

Most of the participants come from the RoboCup@Home leagues, which does skew some answers related to the technologies used, but there are participants also from different Soccer leagues, Rescue leagues, Logistics and Work. This provides a wider range of answers to better represent the community.

Furthermore, these participants represent teams that have succeeded in their respective competitions. The participants have reported 25 victories, 16 second place, 10 third place and 6 fourth place in their competitions, in total. They also reported success in other competitions.

As an extension of that last point, a specific analysis on the top 9 performing teams by their reported success in their leagues was also done in subsection 8.3.4 with the aim of extracting further insights from highly performing teams.

8.4.2 Proposed Guidelines

From the answers of the survey we create a set of guidelines with the aim to help new teams that wish to participate in these RoboCup competitions. Alternatively, already existing teams may use this information to find new approaches and hopefully improve their participation.

Note that in the following chapter 9 guidelines from the action research cycles and the guidelines that follow will be discussed together.

As the answers to some questions have shown previously, definitive answers are hard (if not impossible) to find. Every team has a different background, exists in a different context, and may have different goals. These goals may arise from individual idiosyncrasies or the team, the laboratory or the university having different goals. Taking this into account the following guidelines are to be taken as a piece of advice.

The following guidelines are written with a new team in mind. There is a constant evolution in a team, so approaching all of them at the same time may not be possible,

neither may they be appropriate. Good judgement and trial and error are recommended to find the best fit for each case.

- Find team members or advisors with experience in the RoboCup competition you aim to participate in. Contacting successful teams and finding someone willing to collaborate may be a good approach.
- The amount of team members and its composition depends on factors like the availability of these and the goals of the team. But, to provide a guideline, the average team had 8 team members with as little as 5 and as many as 20. Having a distribution of students from PhD, Master and Undergrad, with sometimes some Staff member is common. The average team composition had an almost even distribution of student types, with a possible Staff member. On a continuing team, knowledge tends to be passed on from PhD to Master to Undergrad, in that order. Recruiting team members as early as from Undergrad allows the possibility of having long lasting team members which benefits the team by their experience and knowledge transfer.
- Be prepared to spend a considerable amount of time preparing for a RoboCup competition. The results from this survey show that a team of 8 members in average dedicate 20h/week (part-time) each for 7 months. Furthermore, the top 9 performing teams average 9 team members, part-time for 9 months.
- When recruiting team members the most valued reported skill is to find people that are proactive. Their coding skills and interpersonal skills are also considered very valuable. Many respondents also related motivation to proactivity. You may want to keep that in mind when creating activities to gather interest in joining the team.
- To recruit new team members the two most used approaches are via informative meetings and via courses. Coursework may introduce skills and technologies that are necessary for the competition and, from there, students may find interest in joining a RoboCup team. When mentioning what incentives does participating in RoboCup have, the most reported ones were improved future job prospects and the ability to publish work done for RoboCup. Future job prospects improve by learning useful knowledge and skills and well known companies appreciating the fact a person has been involved in RoboCup. Futhermore, you may include coursework projects as part of the participation in the team.

- For new team members to learn the basics needed, personal work and supervised personal work are reported to be necessary. This may connect with the opinion of preferring proactive people. Some teams report having courses in their university that prepare them for this, even though they mainly report to teach ROS as they consider it the only necessary requisite.

- Most teams report to meet once weekly for an average of 75 minutes. They also report that, based on the current context, and, especially as the competition approaches, the meeting frequency and duration is altered towards more frequent and longer meetings. Most teams report to aim to have all team members in every meeting if they are available. These meetings are reported to include testing time, code review or programming support, or, have these activities happen after the meeting.

- If you happen to want to take reference of a standard Software Development Methodology (SDM) for your development process, the top performing teams report to take inspiration in the Agile family of SDMs, but not follow them strictly.

- It is a very common practice to keep a backlog of *long term tasks* and another backlog of *currently being worked on tasks*. These tend to be stored in online services like GitHub/GitLab issues or Trello while also keeping a physical one (whiteboard or notebook).

- Holding retrospective meetings is considered very important.

- When thinking about what coding practices are more important, the preferences and goals of the team must be taken into account first. After that, this survey shows that code simplicity is very important. Afterwards, and in the order set by the answer of the top 9 teams, continuous integration, naming conventions and commenting code are considered important too. Add to that good documentation and it's clear that writing code to be read later on is important.

- When you decide what kind of testing to perform, the most important one is end-to-end testing, understood as running full RoboCup competition tests. Performing integration testing is considered the following most important kind of testing. Afterwards, consider doing system testing, unit testing, ad-hoc testing, functional testing and performance testing for your codebase.

- Consider using pair programming in your team. It may be useful to help new team members hop on board, to improve the work on specific RoboCup competition tests and to fix hard problems.

- Create an architecture that takes into account the difficulties of software integration. This is considered the biggest challenge by the top performing teams.

- You may also want to take into account that creating software robust to real life situations (with a lot of variability and uncertainty), and team management, are considered other big challenges in RoboCup competitions.

- Participants report that funding is the main limiting factor when deciding how many participants are able to be sent to the competition. Aim to find funding early on and think over criteria to determine which team members will be preferred over others in the case the situation arises.

- Aim to have a fully working version of your software at all times. You may want to embrace automated approaches for this such as Continuous Integration and Automated Testing.

- Research if there is a simulation available for your robot or competition and if it fits your way of working. The survey showed that using a simulation helps because robots are a limited resource and it may minimize their breakage.

- From a technological point of view, this survey reports most teams to be using the ROS middleware to some extent. All top 9 teams used it too. This may relate to the reported usage of the C++ and Python programming languages to develop their codebase.

- Create an environment that fuels your motivations. Student experience, fun and research outcomes have been mentioned as main motivations for other teams.

Furthermore, most teams reported finding themselves in the following situations and acting on them in the following ways to improve the situation. Many approaches can be taken at the same time, they are non exclusive, practice common sense.

- If you find yourself in the situation where a team member is not doing their part of the work or they lack skills or knowledge to perform it, consider:

  - First, talk about it. Face to face first, understand the situation. Depending on your team, maybe talk about it openly with the team with a constructive mindset. Aim to appeal to the team members interests.
  - Provide simpler tasks to this team member.
  - Team them up with a more experienced team member.
  - If it comes to it, provide non-critical tasks. Work that is good to have.

- You want to avoid the situation where one single team member is the only one to have the expertise of an important part of your system. However, if you do find yourself in this situation, consider:

  - Retrospectively, try to encourage and promote simple code, sharing knowledge (presentations, workshops, tutorials) and documenting code ("documentation day" events are practiced by some teams) during the development year.
  - Pairing this team member with some other team member to learn is proposed.
  - Find someone motivated with taking over and improving the system. This has been reported to be effective by some teams.

CHAPTER 9

CONCLUSIONS AND FUTURE WORK

In this chapter, we will unify the insights from chapters 5, 6 and 7 about UTS Unleashed!'s approach to RoboCup@Home Social Standard Platform League (SSPL) and the insights from the experts' feedback from chapter 8 into guidelines. Afterward, we will discuss future work.

9.1 Conclusions

At this point, it becomes clear that the workshop and survey with experts were an essential evidence-gathering tool to validate the insights from the previously presented action research cycles.

The nature of success is beyond achieving a podium position in the competition. Every team presumably aims to win. However, while the team works towards this goal, other factors come into play. For example, the team members' experience, which usually consists of university students of different degrees, is an important factor during the development of the competition. Most important, the competing students learn both soft and hard skills, which will potentially become building blocks in their future life. Furthermore, as reported in chapter 8, academic outcomes are also important in this setting.

The previous chapters' insights resulted in separate guidelines. Here we unify them into a single set of guidelines structured in two sections: guidelines about team management and guidelines about technical approaches.

These guidelines are to be read as friendly advice to improve a team and take inspiration from. They represent the close experience of one team, UTS Unleashed!, and the communal experience of a significant set of experts in the RoboCup domain.

These guidelines become answers to the research questions from chapter 1 in the following manner:

- *How does a new RoboCup team successfully develop an effective software system for the RoboCup@Home Social Standard Platform League?*

  Before these guidelines were published, a new RoboCup team would be on their own to successfully develop a software system of this kind. They could rely on asking other teams, experts, or doing their own research and experiments.

  Now, a new RoboCup team can consider the following guidelines and iterate over applying the ones found relevant, increasing their chances of success in multiple aspects, including an effective software system for RoboCup@Home SSPL, and saving time.

- *How does the approach of a single team relate to other teams?*

  The approach of a single team is tailored to its context. In this work, several teams' approach has been gathered and analyzed, alongside a close look into the author's team, showcasing a variety of approaches with similarities and differences in between them. A set of guidelines decomposes these approaches into useful elements to analyze and apply to improve a team or relate a team's approach to their context or other teams.

- *What are common insights to improve the development process and its outcomes?*

  The common insights to improve the development process and its outcomes are not universal. To overcome this fact, the insights from this work have been merged into the following guidelines allowing the reader to use them as a starting point. From there, the reader is encouraged to iterate over them to discover what is optimal in their context.

In regards to the research objectives from chapter 1, these have been met in the following manner:

- *To identify the available practices from software development methodologies to improve robotics competitions' software development processes.*

It is important to review what practices are available to not start from scratch and to learn from them. By summarizing them here, readers can understand what influenced this work.

The available methodologies and their practices were discussed in chapter 2, and different elements of these were introduced during the three years of competition as found in chapters 5, 6 and 7. Moreover, in chapter 8 the practices from other teams were also identified.

- *To design, evaluate, and iterate the software development methodology for the three years of competition for our RoboCup@Home SSPL team.*

  Action Research was chosen as the research methodology in order to design, evaluate and iterate over the team's software development methodology. This was done over the three years of participation in RoboCup@Home SSPL as can be seen in chapters 5, 6 and 7.

- *To gather insights from experts about the software development process for RoboCup.*

  The experience and knowledge from experts allowed to learn from them and to compare their approaches in between them and our own. In order to do so, insights from experts were gathered as seen in chapter 8.

- *To analyze, discuss, and compare the software development methodology followed by UTS Unleashed! to the approach from other experts.*

  To create guidelines that apply to multiple teams it is necessary to analyze insights from numerous teams. Other experts' software development methodologies were compared to UTS Unleashed!'s in chapter 8, and is further discussed in the following guidelines.

- *To create a practical set of guidelines that improve the software development experience and outcomes for robotics competitions.*

  It is unlikely to find a solution that would fit every person and every team, e.g., to create the perfect software development methodology. However, a set of guidelines that highlight elements to analyze and experiment with can provide a common tool for almost every reader. This practical set of guidelines to improve the software development experience for robotics competitions, and its outcomes, is described in the following sections.

The guidelines follow divided in sections as previously mentioned.

9.1.1 Team Management

In chapters 5, 6 and 7 are the sections: team management processes; and team software development processes. These sections are united here, as the line that separated them becomes less clear when working on these guidelines.

9.1.1.1 Team Recruitment

Guidelines regarding team recruitment stem, for the most part, from the expert's feedback, despite UTS Unleashed! having showcased engagement in some of the advice.

The most valued reported skill when recruiting team members is finding proactive people. Their coding skills and interpersonal skills are also considered very valuable. Many survey respondents also related motivation to proactivity. Therefore, these facts may be taken into account when creating activities to awaken an interest in joining the team.

The two most reported approaches to recruiting new team members are via informative meetings and via courses. Coursework may introduce skills and technologies necessary for the competition and, from there, set the basis of interest for students to join a RoboCup team. The incentives to participate in RoboCup reported by the survey participants were: improved future job prospects, and the ability to publish the work done for RoboCup. Moreover, future job prospects were reported to be improved by learning useful knowledge and skills, and well-known companies appreciating the fact that a person was involved in RoboCup. Furthermore, it may be beneficial to include coursework projects as part of the participation in the team. In that way, team members could directly benefit from involvement during their studies.

9.1.1.2 Expert Team Members

Remarkably, both the workshop and survey pointed out the importance of having at least some team members with experience in the RoboCup competition they were taking part in, particularly the team leader. The experience from the UTS Unleashed! team is in agreement, since the lead developer had previously participated in RoboCup@Home events.

Based on these results, new teams are advised to find team members or advisors with experience in the RoboCup competition they aim to participate in. Alternatively, if no one can directly be recruited, the right approach may be contacting successful teams and establishing collaborations.

9.1.1.3 New Team Members Basic Skills

Personal work and supervised personal work are reported to be necessary for new team members to learn the needed basics. This may be related to the proactive people preference. Indeed, some teams report having courses in their university that prepare them for this. Nevertheless, they mainly report teaching Robotics Operating System (ROS), considering it the only necessary requisite.

9.1.1.4 Team Size and Composition

The number of team members and their composition depended on factors such as the availability and the goals of the team.

Based on the survey's data, the average team consisted of 8 members, while the range spread from as few as five and as many as 20 members. The team's academic level distribution ranged almost evenly, including undergraduate, masters, and PhD. Moreover, staff members were also common. In teams that continuously participate in their competitions, knowledge was reported to be passed on between team members, in PhD to masters to undergrad order. Finally, the recruitment of team members as early as undergrad allows for long-lasting team members, which benefits the team with their experience and knowledge transfer.

9.1.1.5 Standard Software Development Methodologies

Van Der Zant, Schomaker, Wiering, and Brink [79] explore the usage of Extreme Programming and Gerndt, Schiering, and Lüssem [80] showcase Scrum in RoboCup settings, both approaches showing promising results with their practice of popular standard Software Development Methodologies (SDMs) stemming from the Agile manifesto [57].

The survey revealed that the top-performing teams take inspiration in the Agile family of SDMs, but do not follow them strictly.

The lead developer of UTS Unleashed! investigated in 2018 which methodology, or partial practices, would be a good fit for the team. Some practices were not possible to be applied, such as daily stand-up meetings, as the team could not meet daily at the same time. On the other hand, cycles of three weeks of sprints from the Scrum methodology were tested with some team members, but they were unfruitful. Thus, we cannot provide a definitive conclusion on which SDM to use. Every team may need to evaluate their needs and preferences with regard to the application of these techniques. However, it could be interpreted that the team practiced a lightweight version of sprint cycles, as

tasks were created as required, then prioritized taking into account time estimates. Team meetings would then provide updates to the team as a whole on the state of these tasks. Casual meetings between team members would happen continuously during working days providing closer communication, testing and integration of components. Finally, team meetings would also have demonstrations and a retrospective element, as the team was open to talking about the processes that were used at any time. In conclusion, the process was less rigidly structured than Scrum's sprints, but the Agile philosophy behind it existed.

Finally, it would be recommended that the reader analyze how the Agile manifesto may apply to their team and software development structure, and experiment with their own implementations of this philosophy to improve their processes and team building. Furthermore, using UTS Unleashed!'s experience, the twelve principles behind the agile manifesto can be reinterpreted as shown in Table 9.1 and Table 9.2.

Agile principle	Application to the team
Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.	The team itself acts as a customer and frequent end-to-end testing is practiced.
Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	As new insights and systems appear, the team embraces them as soon as possible.
Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.	Demonstrate the state of the software frequently, for example, via ORTs or demonstrations in meetings.
Business people and developers must work together daily throughout the project.	Every team member acts with the role of requesting features and developing them.
Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.	Encourage and trust motivated team members, support them while giving them freedom.
The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.	Encourage teamwork, propose a day to work in the lab, create events that engage the team to work together.

Table 9.1: Interpretation of the Agile Manifesto by UTS Unleashed!'s lead developer, part one.

Agile principle	Application to the team
Working software is the primary measure of progress.	The robot's software must always work. Ensure it by any means, e.g., snapshots of the robot's system or tagged versions.
Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.	Allow team members to work at their own pace, however, the last weeks of development before the competition are known to carry an additional workload, prepare for it.
Continuous attention to technical excellence and good design enhances agility.	Embrace coding best practices as much as possible while keeping the team comfortable in their own way of working.
Simplicity -the art of maximizing the amount of work not done- is essential.	Keep your systems and code as simple as possible.
The best architectures, requirements, and designs emerge from self-organizing teams.	Teamwork and experience provides the best results.
At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.	Reflect often on how the team is working and aim to improve the processes, tools, and ideas continuously.

Table 9.2: Interpretation of the Agile Manifesto by UTS Unleashed!'s lead developer, part two.

9.1.1.6 Meetings

Most teams report meeting once weekly for an average of 75 minutes. They also say that, based on the current context, and, especially as the competition approaches, the meeting frequency and duration is altered towards more frequent and more prolonged meetings. Most teams report to aim to have all team members in every meeting if they are available. These meetings are reported to include testing time, code review, or programming support, or having these activities happen after the meeting.

UTS Unleashed! did not follow this structure as it followed a fortnightly meeting structure for the first months of the project every year, but by the end, it would also meet weekly. These meetings tended to be longer, but they also included testing time.

The survey brought up the fact that holding retrospective meetings is considered very important. UTS Unleashed! held them after the participation in the competition and, from them, created retrospective documents to aid future work.

9.1.1.7 Keeping Track of Work

Keep a backlog of long term tasks and another backlog of currently being worked on tasks. Teams tend to store them in online services like GitHub/GitLab issues or Trello, while also keeping a physical one (whiteboard or notebook).

UTS Unleashed! used Trello during the first months of the project every year, and used a physical backlog on a whiteboard as the competition drew closer. UTS Unleashed! also used a whiteboard on the competition days.

9.1.1.8 Teamwork

Larson and LaFasto [113] discuss a set of 8 characteristics of highly effective teams:

- A clear, elevating goal
- A results-driven structure
- Competent team members
- Unified commitment
- A collaborative climate
- Standards of excellence
- External support and recognition
- Principled leadership

A relevant quote from that work, “From this list is evident that effective teamwork has a strong relationship with motivation” matches responses found in the survey in chapter 8 and the experience of the lead developer in the UTS Unleashed! team. This makes the concepts of motivation and proactivity stand as candidates for further research.

Furthermore, Asproni [114] explains how, in the context of software development teams, the Agile SDMs showcase these characteristics that make effective teamwork possible.

When thinking retrospectively over the three-year journey of UTS Unleashed!, these characteristics can be interpreted to be present at different points in time. Setting aside the clear and common goal of winning in a competition that can be considered shared by all teams, UTS Unleashed! showed in the action research cycles multiple occurrences of a collaborative climate and unified commitment when the team had some kind of crisis. Furthermore, responses from the survey from other expert teams manifested some of these characteristics explicitly with varying degrees of implication. For example, some teams explicitly reported investment in testing, coding standards, documentation, and other techniques that map to excellence standards.

A higher-level look can be taken over the first-hand experience of UTS Unleashed! where the processes and practices were continually adapting to the changing context that the team was in, and in the results from the survey where no agreement on a single way of approaching the software development process was found.

Finally, we argue that these characteristics become a set of concepts to use as guidelines to improve teamwork.

9.1.1.9 Workload

All teams must be ready to spend a considerable amount of time preparing for a RoboCup competition. This survey's results show that a team of eight members, on average, dedicate 20h/week (part-time) each for seven months. Furthermore, the top nine performing teams averaged nine team members, part-time for nine months.

9.1.2 Technical Approaches

The technical approaches are presented here. They do not engage in particular implementations of robot systems, but aim to provide advice on the software development process itself.

9.1.2.1 Testing

The survey's data revealed that the most critical kind of testing is end-to-end, understood as running full RoboCup competition tests. This fits with UTS Unleashed!'s experience where ORTs had an integral role in the progress of the project.

In subsubsection 6.4.1.2 how the UTS Unleashed! ORTs were held was explained in detail. These became one of the most valuable forms of advice, as they had an integral role in the progress of the project. Holding ORTs, as detailed in the retrospective documents, was reported as most influential in pushing progress in the maturity of the team's systems, and in the scoring for the competition. Additionally, it mentally prepared team members for the competition conditions.

The event itself was very intense, and could become very stressful for some team members. These were considered to be realistic training for the real competition. Care must be taken to warn team members of this phenomena, and managers must be ready to address any kind of issues that may arise from this on the team members. If any team member becomes distressed, it is essential to help this team member and be empathetic with them, as it is a natural reaction. Taking time to calm them down and talk about the issue to prevent it in the future is recommended. However, every person may benefit from a potentially different and personalized approach.

During these events, other external factors arise or are sometimes artificially encouraged. Examples follow:

- Team members not being available, so others must be ready to act in their place.
- Network outages, both local network or internet access.
- Artificial noise added to the environment.
- Artificial illumination changes.
- Variety of operators affecting different factors. For example, for Automatic Speech Recognition (ASR) naïve users and people with different accents and cultural backgrounds may provide new challenges. For perception, people dressed in the same way, people wearing items different from the ones usually tested in the lab (hats, sunglasses, scarfs, jackets) or objects with interesting shapes or colors.
- Unique arena elements. Furniture with different materials (reflective materials tend to be hard for robots), carpets, chairs with different kinds of leg configurations, mirrors, etc.

It is to note that strict and full ORTs could not be run until late in the project. Meanwhile, adjustments were made to benefit as much as possible from the events. For example, partial scores would be created, bypassing of not yet implemented systems

would be allowed, instead of a single run of each test, multiple runs would be allowed as time permitted, etc.

The survey results showed the order in which the other types of testing were ranked. Integration testing won over the following items by a noticeable difference. UTS Unleashed! coincides as they also suffered from integration issues. Examples of integration problems were: unmatching interfaces between systems, different interface assumptions between systems, like expecting different coordinate formats, or partially implemented features.

System testing is considered the following next most important one, although it is of similar importance to integration testing and probably only one step behind end-to-end testing. UTS Unleashed! found system testing necessary, especially on the robustness of booting up the robot reliably with the team's software.

Unit testing follows in importance. Some teams in the survey reported making heavy use of unit testing. UTS Unleashed! found lesser adoption of unit testing, and it was mostly encouraged on a just a few packages that other team members used to build upon, or in packages that suffered regressions[1] often.

The next type of testing was ad-hoc testing. Somehow this kind of testing is inherent in any kind of development, as while writing code, it is expected that that code is tested incrementally. It was the most used kind of testing in UTS Unleashed! as team members would do it without even realizing it.

Functional testing is the next type of testing by the ranking in the survey. UTS Unleashed! did not engage in this kind of testing.

Finally, performance testing was the last major type of testing from the ranking. Performance testing is especially crucial on robot platforms with low computational power. UTS Unleashed! engaged in this kind of testing on systems that needed to run continuously, like autonomous navigation, or when noticeable delays due to computation times created issues. For example, people perception tasks taking too long to be able to finish the test in time.

9.1.2.2 Coding Practices

Students from degrees in computer science or information technology may be familiar with the wide variety of coding practices considered the current standard in software engineering. This is not the case for other fields. Both in the workshop and the survey, it was reported that there were students who were completely unfamiliar with standard

[1] Software regression can be understood as a bug that breaks a previously working function.

coding practices. A team entirely composed of mechanical engineers provided themselves as an example in the workshop discussed in chapter 8.

Following standard coding practices is believed to help improve the quality of software and facilitate its development and maintenance [115]. But they come at a cost. Time and effort must be put towards learning and following them.

From the answers of the top nine teams in the survey in chapter 8, a ranked set of coding practices was developed, which can be used as a guideline on what to promote in a team.

Code simplicity was the most valued coding practice. UTS Unleashed! experienced multiple times during the project how code simplicity made a huge difference in avoiding bugs and embracing others' code. As an example, when a simple interface to use the robot's tablet was created, the team members embraced it and improved their development experience by being able to debug on the robot's tablet. The same interface improved the user experience for the operators of the robot and provided UTS Unleashed! with the best Human Robot Interface award in 2017. Code simplicity was encouraged often, and team members were reminded to comment the code when there was a legitimate reason for complex code.

Continuous integration, naming conventions, and commenting code were ranked very closely together as the following most important practices. These three practices share a fact popularly quoted from Robert C. Martin: "Indeed, the ratio of time spent reading versus writing is well over 10 to 1. We are constantly reading old code as part of the effort to write new code. ...[Therefore,] making it easy to read makes it easier to write." [116].

UTS Unleashed! did not embrace continuous integration over the whole codebase. Only selected packages had it set up. They were maintained by team members with a solid background in software engineering, with interest in unit testing and automatically building these packages. The more experienced team members would help set it up in repositories for other team members that they believed would benefit from it. Naming conventions were followed and felt like an improvement in the last year of preparation, when the team moved away from creative names for packages to concise and straightforward names. Commenting code was encouraged from the start of the project in parallel with keeping the code simple.

9.1.2.3 State of the Art Software

The RoboCup competitions help push state of the art in many fields. To stay competitive, a team must be able to reference and use novel advances in a wide range of areas. This is also a consequence of working in the robotics field where the integration of many technologies fuels innovation.

There is a rising trend in publishing reference implementations about scientific publications. This is commonly seen in different areas of robotics. For example, convolutional neural networks are widely used in works like OpenPose [111], which is at the same time commonly used in RoboCup@Home to guide human detections and interactions. Reference implementations allow researchers and engineers to quickly test other's concepts for their own use case.

In UTS Unleashed! providing access to the ROS framework was deemed by its lead developer as a priority. ROS provides access to a vast number of reference implementations in fields like autonomous navigation, perception, and manipulation. Furthermore, the Theano [99] library was also made available to allow deep neural network approaches to be used. To build and run these tools, considerable effort was needed, but it paid off when looking at the big picture as they were the building blocks for the team's systems.

A large number of responses from the survey, all top nine performing teams included, reported using ROS. This middleware presents many features that make it attractive for some RoboCup competitions.

This is to say that in the time this dissertation is written, ROS is the de facto standard for using and writing robotics software. And new and existing teams should look at what is available and the platform's evolution. But for the future, it is well worth exploring what frameworks and libraries are widely used in the field and analyzing if they fit the team's vision. Some teams develop their own frameworks while also taking inspiration from existing ones.

9.2 Future Work

The direction of the research performed on the UTS Unleashed! team is unique to the context of this specific team. Having a researcher join another, or even multiple teams during their development process, would allow accounting for different cultural backgrounds and team profiles. By investigating how others work, the various nuances of this process can be better understood and improved.

From the point of view of just UTS Unleashed!, if they were to participate again in RoboCup@Home SSPL, further research regarding proactivity and motivation would be of interest. Working on defining the factors and practices that were the most influential in building teamwork, is closely related. Also, the recruitment and teaching approaches by the team could be improved. Every year, new team members reported difficulties in getting up to speed and able to contribute to the team.

For technical approaches, the last retrospective document showed in the "What did not work well?" section minor problems compared to the items found in previous years. Having a robust codebase allowed significant risks to be taken, as a backup would exist. This would seem to be a feature to take advantage of, especially if a team member wanted to take responsibility for a completely new approach in any of the systems.

Improving research outcomes would be the most commonly referenced concept in the survey. A closer look into the approaches of teams that report a more substantial amount of publications would be beneficial, probably in the shape of private interviews. They allow a directed conversation that can potentially provide a more in-depth understanding than a survey to uncover complexity evidence.

In general, a survey officially distributed by and to the RoboCup community, would probably provide higher engagement, which could allow for more fine-grained results. For example, results separated by the different RoboCup leagues that identify approaches that may be more suited to the specific goals of, for example, the @Soccer league rather than the @Home league. A further step could be taken to gain insights meaningful to every league, by allowing the community beforehand to collect questions of interest regarding the development process.

APPENDIX A

ACRONYMS

API	Application Programming Interface
AR	Action Research
ASR	Automatic Speech Recognition
CI	Continuous Integration
CPU	Central Processing Unit
CSV	Comma Separated Values
DHCP	Dynamic Host Configuration Protocol
DSPL	Domestic Standard Platform League
EEGPSR	Enhanced Extended General Purpose Service Robot (GPSR)
GPSR	General Purpose Service Robot
GPU	Graphical Processing Unit
GT	Grounded Theory
HRI	Human Robot Interaction
JSON	JavaScript Object Notation
NLP	Natural Language Processing
OPL	Open Platform League
ORT	Operational Readiness Test
RAD	Rapid Application Development
RGBD	Red Green Blue + Depth
RGB	Red Green Blue
ROS	Robotics Operating System
SDM	Software Development Methodology
SLAM	Simultaneous Localization And Mapping
SMACH	State MACHine library
SPR	Speech and Person Recognition
SQL	Structured Query Language
SSL	Sound Source Localization
SSPL	Social Standard Platform League
STT	Speech To Text
TTS	Text To Speech
UTS	University of Technology Sydney
YAML	Yet Another Markup Language

APPENDIX B

TEAM RETROSPECTIVES

This appendix contains the original retrospective documents created after every edition of the RoboCup@Home SSPL competition. The team members wrote individually in a collaborative document their opinions on how the development year and the competition itself went, including the positive facts, negative facts and what to improve for the next edition. These documents were completed during the month following the competition participation and they may present conflicting opinions from different team members, which have been anonymized.

B.1 Retrospective 2017

B.1.1 What worked well?

- Having several dry-runs, practice runs, and Operational Readiness Tests.
  - Focusing on the points in the first stage tests.
- Team Collaboration at venue.
  - Quote: "We worked pretty well during the competition. Though we had lots of hard coding for the tests, I think we enjoyed the spirit of hackathon. We learned and shared with each other."
- GitLab as collaborative coding platform.
- On-site technology.
  - Plenty of Australian power ports.
  - Renting two robots (rather than just one).
  - We used the local GitLab to synchronize source code whenever internet access was unavailable.
  - We used the backup router to provide DHCP during an emergency. There was no inadvertent WiFi activation.
- Planner (a planning interface with tablet feedback developed for the GPSR test)

  - Onscreen feedback was very helpful for debugging.
  - Adding additional tasks (or new skills) to a program that used the GPSR planner was fast and reliable.
  - Expressing a test step as a discrete task made it easy to test that step only.
- QiMate (a shim layer to ease usage of the Qi APIs).
- Tablet Interface (especially when WiFi was unavailable)
  - Starting and stopping tests from the screen worked great.
  - Feedback about robot inputs and plans.
  - Buttons to skip around within test.
- Human clothing analysis (although we did not get to prove it)
- Social presence and impact on English-speaking audience.
  - Quote: “ANONYMOUS was fantastic at giving presentations.”

B.1.2 What did not work well?

- Long hours at venue: Team members were "on duty" even when not working. By the finals, everyone was too exhausted to be effective at simple tasks.
- Hotel did not have a gym or pool.
- Writing presentations at the last minute.
- Not having a plan for the second stage tests or finals.
  - Quote: “We didn’t pay enough attention to things that are fundamental for all tests, such as localisation, navigation and mapping, which could help us get more points.”
- Too much focus on mechanical tasks over social functions. That is not the ideal use of pepper.
- Starting new projects.
  - Dependencies were not coordinated. Git Submodules did not work well.
  - No project template or best practices.
  - Basic sequencing of actions took repeated effort.
- Developing without the robot.
- Complicated software frameworks.
  - Hard to make small changes to cocktail-party or help-me-carry without fully understanding a large number of single-use abstractions. Hard to test parts without re-running the entire scene.
  - Metaprogramming in SPR and GPSR made it hard to detect type errors at compile time.
  - Quote: “Our code quality is not good from the beginning and we didn’t do any code review to rectify this issue.”
- Feedback for partial speech recognition.
  - In noisy arena conditions, speech recognition would fail very often, and we could not troubleshoot or fall back to a partial understanding.

- Fully offline operation.
  - Speech recognition would sometimes make calls to cloud services, causing timeouts or failures.
  - Many startup scripts expected a Dynamic Host Configuration Protocol (DHCP) address from an external router.
- Hard to tell what is currently running on the robot.
- Hard to stop a task without killing its entire process.
- Reliability of robot hardware.
- Robot Mobility.
  - Navigation.
    * Localization.
    * Mapping.
    * Landmark detection.
- Object Detection and Recognition (we did not even try).
- Autonomous life was all-or-nothing. Would like partial or situational capability.
- Confusion regarding privacy of the team internal projects around the competition.

B.1.3 What should we do next?

The retrospective document had a third question about the next steps:

- Coordinate packages and dependencies
  - We should be able to install a library on the system and let any program import it without additional work.
    * When a package depends on some service, it should support the user in troubleshooting that service.
    * CI should test libraries automatically and deploy with single button.
    * A robot skill developer should be able to easily publish their skill as a package with a standard start and stop interface.
- Standardize development environment
  - Mock interfaces to missing hardware
    * VM or container with base OS, able to update libraries automatically - File sync between VMs and hosts (need to compile kernel extension)
    * Document how to start a new project
    * Run development training sessions
- Maintain components that work well
  - Planner: reliably stop tasks and skip ahead or repeat, use standard formats and dynamic planning theory
    * Launcher: edit menu options without needing to restart
    * Topfile patterns: would like to generalize this
- Extract value from components that did not work well

- Rethink our approach to robot mobility
  - Quote: “I’d like to see more ‘reactive’ robots – robots that are always moving rather than getting stuck trying to navigate around imagined obstacles.”
- Focus on Repeatability and Analytics
  - Use CI to ensure we have a record of everything that gets installed
  - Store ROS data so that we can test new code in old situations
  - Record user interaction events and timing
- Use external processing (with fallback if WiFi fails)
- Work to the purpose of the robot (social not mechanical)
- Make general project plan for entire year
  - Prioritize robot skill development early
  - Start something for every robocup test before competition begins
  - More operational readiness tests
    - Don’t need to be offsite.
    - Do need to simulate complete network failure.
    - Need to include forced absences.
  - Prepare presentation material in advance. Use in-lab publicity events as rehearsal for open challenge.
- Bring Australian presents to give away. (Another team gifted local sweets at the end of the competition and it was great).

B.2 Retrospective 2018

B.2.1 What worked well?

- Everything ran
- Package management and CI is good
- Magic speech
- Pepper skills
- Time saved by passing robot inspection quickly
- Package management worked well
- Continuous Integration helped
- Sprints and feature delegation
- Final week visualisation of test progress / status
- Real world testing
- Working together to test with the robot for Tour Guide outside the lab
- Source control and package management (and DNS?)
- Everyone shared responsibility (and the coolest T-Shirt Banana)

- Having access to multiple powerful computers (Alien laptops)
- Packaging and CI
- Seeing so many in the lab so often at all sorts of times
- Speech recognition worked very well
- Simple HRI interface with simple APIs
- More people in the team and more divided tasks that were easier to manage
- Enthusiastic and talented interns that were well used
- Pairing on work
- Building up from a proof of concept and iterating
- The team worked together pretty well
- Question and answer system
- The tablet animation is good
- In the last few weeks the team was quite efficient, but we can't leave everything to the last minute
- Working directly with different team members
- Team cohesion and collaborative decision making
- ORTs
  - Operation tests
  - Well structured ORTs
  - Frequent ORTs
- The hospitality and sportsmanship
- Tim Tams and Koalas as gifts at end of competition
- Offline capability
  - Programs that were WIFI network independent (entirely onboard)
  - Offline-first strategy
- Trello in early weeks / months
- Rigorous testing paid off
- Deployment: Base Gentoo Pepperfix OS + pip installing + simple extracting a .tar.gz
- Start with simple and modular approaches for problems that "do the job" even not very good, and evolve into refined versions

B.2.2 What did not work well?

- HRI:
  - Need better HRI to guide user in tests
  - HRI Style Guide violations
  - Focus on HRI - establish early-on instead of post-hoc
- Navigation:
  - Navigation gets lost too easily
  - Navigation
  - Navigation didn't work well enough
- Strategy and Decision Making:
  - Reduce duplication by improving feedback of: what's needed; what's working
  - We need a global to do list in priority order
  - Not enough experimentation with different ideas
  - Finals was too last minute
  - No overall team vision - disconnected projects and not prioritising well
  - Going ahead with too-personal choices in disagreement with the team vision and objective
  - Skills repo and state machine did not feel like the right abstraction
  - The Planner didn't work as expected
  - Planning language was not used for tests
- Communication:
  - Communication of people needing help occurred too late or not at all
  - GPSR problems - lack of teamwork; lack of communication; difficult resolution
  - Lack of communication and "transparency" about developed code within team
- Software development process:
  - Need a process to work through team issues
  - Lack of visibility on tests / issues until final week
  - Management of team numbers and expertise
  - Need to meet to discuss skill / test progress more regularly
  - Too much rigidity on software design and unit test preventing fast coding
  - Last minute developments
  - Multiple Docker images
  - Sometimes felt unsure how I could help / be involved
- Testing:
  - Lack of iterative design for tests / testing
  - Tests could only be run by the person in charge, limiting testing
  - Hard to test features independently of each other

  - Test environment eg lighting, noise affected performance
  - Need more testing in different environments

- Training:
  - Lack of shared knowledge on tools and available stuff
  - Skill re-useability
  - Centralised shared code strategy (skills) too difficult?
  - Lack of ROS knowledge to improve development loop on tasks involving sensory data and 2d-3d information

- Others that don't seem to fit an above category:
  - Marketing roadmap
  - Dynamic skill states
  - Network: channel selection; router limiting speed; Jetson pack if needed; stand-by robots
  - Loss of Sam's time
  - Mixed package approach ROS/Python, manual work involved, error prone

- Things that puzzled me
  - Lack of analytics (eg robot tells the same joke twice etc)
  - How to avoid discrimination or bullying
  - Preparation for travel and competition
  - How to be confident that a feature is reliable

B.2.3 What should we do next?

- Navigation stack:
  - Localization:
    * "Solve" localization - get it running fast and flawlessly
    * Add visual odometry
    * Automate parameter tuning
    * Create a method to know how lost we are, and if we are, use HRI to relocalize
  - Mapping:
    * Take advantage of 3D information to map
    * Automate parameter tuning
    * Add virtual obstacles
  - Navigation:
    * Migrate to use move_base_flex
    * Optimize (or make our own) controller
    * Make our own navigation state machine
    * Make our own local planner
    * Make our own global planner (waypoint based)
    * Make our own recovery behaviours

  - Miscellaneous:
    * Have a emergency stop, holding the head maybe, also the hatch of the Robot base
    * Recover motorbike mode
    * On tablet debugging, map, plans, sensors, costmaps... with HRI buttons popping when things go wrong (I'm lost, can you click where I am in the map image?, I'm stuck, see what's I'm seeing, is it a hallucination so I should ignore it?)

- Object recognition stack:
  - Implement efficient scanning, hybrid with navigation probably
  - Implement a "bounding boxes of interest" approach (to ease exchange of recognition packages)
  - Calibrate cameras
  - Check if stereo is possible for shorter distances if there is any overlapping in pepper cameras
  - Test object recognition approaches
  - Use the depth camera as much as possible to ease the task
- Manipulation:
  - Just general work on this direction needs to be done if we want to grasp something, push stuff...
- Speech recognition:
  - Test CMU Sphinx
  - Reduce the time it takes for speech recognition to begin listening
  - Stream compressed audio by WiFi (using UDP for smaller packets?) to our cloud server, and have that server forward it on to Google and find an optimal parse, etc. This lets us control the audio-upload more carefully.
- Speech generation:
  - Precompute speech synthesis and then just play back the files
- Following stack:
  - Better use of head & body pose to keep track of person
  - Create a cheap depth based tracker and rgb based tracker to aid tracking
  - Add safety (tied to better controller in navigation)
- Human perception:
  - Add our own face detection + recognition + tracking (naoqi one is very resource consuming and has side effects)
  - Streamline gender detection
  - Add our own age detector
  - Add our own "human description" detector (glasses? Clothes? Hat?)
- Tablet interface:
  - Discuss an API supporting more flexible showing of stuff from Python (e.g. show a red circle at x.y moving at speed x.y.theta of size, reducing its size at a rate of 10%... update item id 35784 text to 'bla', update item is 46864 to jpeg image IMG). Also discuss general API keeping powerfulness of the interface but also offering simple way of using it.

  - Streamline buttons to start tasks... or maybe just have an “admin” interface with diagnostics (Swype right, get all the admin stuff, Swype left have the nice HRI interface)

- General:
  - Learn the easiest way of optimizing slow Python code (cython? Boost Python?)
  - Move to python 3
  - Wrap all "hard" ROS stuff in a nicer way of using it (dealing with images, dynamic parameters, transform poses, debug 3D stuff on rviz)
  - Creating simple APIs for core functionality with no dependencies on Naoqi/ROS, and then having integration layers

B.3 Retrospective 2019

B.3.1 What worked well?

- Teamwork in the competition was amazing
- Having lunch on the ORT days helped bring people together
- Regular ORTs with realistic point scoring
- The lab set up to be "more difficult" than the real arena (e.g., shiny chairs, difficult tables)
- OpenPose usage for the tests, great results
- magic_ros external computing
- Re-using tablet interface from last year
- Pair programming to work on tests
- Pair programming to share knowledge of specific functionalities
- Last-minute hacking gave amazing results
- RTAB-Map mapping & localization (excellent in Restaurant test)
- Great robustness in tests, especially Stage I
- Operational Readiness Tests as backbone to evaluate and advance our development
- Speech recognition performed amazingly including open grammars
- Booking a hotel nearby seemed to be good
- Having a working space ready for coding (i.e., the magic lab) after the arena closes, to finish incomplete work
- We had the main packages at what seemed to be the right level of granularity: navigation, listen, speak, tablet
- Using face vectors to remember people was really effective and surprisingly reliable
- YOLO seemed to work well, especially with the external compute device (though, in future I’d like to see even more alternatives explored)

B.3.2 What did not work well?

- HRI:
  - Every test had its own code for engaging with the user
  - Every test had different behaviours when people approached the robot
- Navigation:
  - Navigate with the long and narrow black table around (it was invisible to Pepper sensors)
  - This was one of the most crucial aspects of the system yet we left it too late. We should have devoted lots of our efforts to navigation until it was "solved" before working on the rest of the system
  - Navigate to fixed coordinates inconsistent
  - RTAB-Map reduces the navigation speed
  - Although navigation is a core module, late changes extremely affected challenges' logic
  - Lack of (graphical) tools to check hard-coded map coordinates
- Perception:
  - Training the full object recognition dataset was very narrow in time, too stressful
  - Image processing for the training dataset is extremely crucial but it had been left until days close to the competition
  - Didn't have tools to evaluate network models
  - Object recognition was too sensitive to background/context
  - We didn't try anything other than YOLO for object recognition
- Strategy and decision making:
  - A period lacking leadership almost froze development
  - We didn't resolve the different working styles: some people preferred larger chunks of work that they focus on, while others preferred shorter jobs with faster feedback; but we didn't find a way to help both kinds of people to collaborate with each other well
  - New members lacked knowledge transferral about how RoboCup works from previous years
- Communication:
  - Sharing in-depth knowledge of specific systems was not achieved (Navigation & Building SW for Pepper at least)
  - Not much effort to define interfaces/boundaries before projects began: the architecture evolved from a bottom-up process without much top-down vision for how the system will work
  - ORT days were a good way to collaborate and communicate, but it was also very hard to make progress because there was too much noise from all the communication and collaboration
  - Some official releases from core modules were not informed to all team members
  - Language barrier with some teammates slowed down communication
- Software development process:
  - Some team members needed more guidance to advance their work for various reasons, wasted potential when they weren't given that guidance!
  - A lot of time spent on subsystems that didn't have a direct line to scoring points

  - Didn't have streamlined tools to synchronise robot external and internal environments (deployment of code meanwhile developing)
  - Lack of documentation and/ or guiding on implementation of tests from previous years made them hard to re-use
  - Stage-2 tests started too late affecting their robustness

- Testing:
  - Very few unit tests. Quite some time could have been saved by having them in some modules
  - Nothing set up to make unit testing easy

- Training:
  - Not everyone was trained & used parts of what we built for re-use
  - We wanted to build up skills in different areas, but because we lost time everyone fell back into the same areas they were comfortable with
  - Lack of enough documentation/guidance on pre-existing systems for some team members

- Architecture:
  - ROS added a lot of complexity and increased the barriers to entry, setting us back probably 4 months:
    * I don't think it we got much return (in terms of time or productivity) on this added complexity in any area except for navigation
    * Perhaps we could have got the same benefits by just porting RTAB-Map to use Qi framework
    * ROS isn't really cloud-friendly, so it was probably detrimental to magic_listen (I never did end up setting up the transcribe node to run in the cloud)
    * CI processes and apt install didn't work well for ROS
  - User awareness/engagement was a mess: every test handled making eye-contact or engaging with the operator in a different way (need to create magic_awareness)
  - Image related libraries located in different locations made their usage harder than necessary

- Others that don't seem to fit an above category:
  - Mixed deployment system of Python pip packages and ROS packages was sometimes confusing to know where things came from
  - Many Stage II tests involved elements of research combined with elements of implementation. The research aspects were not challenged/tested early enough or regularly enough (e.g., perhaps this didn't suit the ORT approach and might have benefited more from research discussions until an implementation strategy could be decided)
  - Setting up the external computing device at the competition venue was high-stress and risky (even though we had practiced it many times, and we didn't make a mistake, it would have been really nice to have had a robust system that we know just works - perhaps using liveness or health check messages that cause systems to start/restart if needed)
  - I (anonymised) believe that we should never leave early (or go to bed) the day before a test, if the test is not ready. We should not have gone home early the night before the finals until we had tested a first version of the working finals test.
  - Testing restaurant in a nearby cafe during robocup before the real test caused a political issue

- Some of the packages couldn't be installed with magic install even when the document suggested to use it.
- There were interesting things happening in relation to sleep: the decision to stay up late was a trade-off between wanting to show commitment to the team, versus needing to get personal sleep, versus being sufficiently well rested for upcoming challenges. For new team members unfamiliar with the fatigue, they did not manage their energy well... getting too fatigued too early to be "helpful" in the early days, to the detriment of their performance in their later test.

B.3.3 What should we do next?

This was the last year of participation so this section was not completed. It would have been interesting, but unfortunately (even thought the question existed in the document) there is no information available coming from all the team members.

APPENDIX C

DATA ANALYSIS EXAMPLE

This appendix contains an example of how the data analysis was performed in chapter 8. The same technique was used in other parts of this thesis like the analysis of the available data from the workshop and the data from the git and Trello platforms. This example showcases the usage of Grounded Theory.

C.1 Question Q6.2-6.3

Question Q6.2 stated *"Please order these concepts by importance to address them for the success of your team in the competition."* with the options: *"Naming conventions, Commenting code, Code simplicity, Code portability, Unit testing, Continuous integration, Pair programming, Shared code ownership"*. The following question Q6.3 was a text field for participants to optionally provide further explanations.

C.1.1 Question's Background

First we need to understand where this question came from. During the three years of preparation and participation in RoboCup@Home SSPL the lead developer had the intention of creating code with high quality standards while at the same time attending to the needs and preferences of the team members in regards to their development styles. This situation showed to be challenging because code quality concepts needed to be learned and that implied effort and time, just like the tasks the team members were working on already. For example, when requesting to implement unit tests to a team member for a piece of software, the lead developer was engaging in a trade off between a team member implementing more functionalities or testing a system (manually) further. At the same time, this team member would need to decide if to comment pieces of code or write documentation. *The question initially asked to score as "Not important", "Slightly important", "Moderately important", "Very important"* or *"Extremely important"* every one of the options. A trial of this format revealed that it did not provide useful information as respondents simply marked almost every option as "Extremely important". Moreover, in this format the trade off between concepts was not represented.

Furthermore, in the workshop presented in chapter 8, participants manifested they also found trade offs between concepts that, initially, may not seem comparable. After discussing this topic with key team members and iterating over how to present the question and what concepts to include, it took the form seen in the analysis.

C.1.2 Raw Data

An excerpt from the raw data, blurred to ensure anonymity, from the survey for Q6.2-Q6.3 is shown in the following landscape page.

Q6.2_1	Q6.2_2	Q6.2_3	Q6.2_4	Q6.2_5	Q6.2_6	Q6.2_7	Q6.2_8	Q6.3
Please order these concepts by importance to address them for the success of your team in the competition. - Naming conventions	Please order these concepts by importance to address them for the success of your team in the competition. - Commenting code	Please order these concepts by importance to address them for the success of your team in the competition. - Code simplicity	Please order these concepts by importance to address them for the success of your team in the competition. - Code portability	Please order these concepts by importance to address them for the success of your team in the competition. - Unit testing	Please order these concepts by importance to address them for the success of your team in the competition. - Continuous integration	Please order these concepts by importance to address them for the success of your team in the competition. - Pair programming	Please order these concepts by importance to address them for the success of your team in the competition. - Shared code ownership	Please add here any comment about your answer in the previous question about coding practices importance.
{"ImportId":"QID117_1"}	{"ImportId":"QID117_2"}	{"ImportId":"QID117_3"}	{"ImportId":"QID117_4"}	{"ImportId":"QID117_5"}	{"ImportId":"QID117_6"}	{"ImportId":"QID117_7"}	{"ImportId":"QID117_8"}	{"ImportId":"QID118_TEXT"}
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
								[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
[illegible]	[illegible]		[illegible]	[illegible]	[illegible]			[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	

C.1.3 Analyzing Q6.2 Raw Data

The following landscape page shows how the raw data was used to find insights by sorting from most important to least important concepts. Different weightings can be observed with the aim of not missing possible interpretations.

Naming conventions	Commenting code	Code simplicity	Code portability	Unit testing	Continuous integration	Pair programming	Shared code ownership
Amount of #1:	Amount of #1:	Amount of #1:	Amount of #1:	Amount of #1:	Amount of #1:	Amount of #1:	Amount of #1:
2	3	10	3	1	2	1	4
Amount of last:	Amount of last:	Amount of last:	Amount of last:	Amount of last:	Amount of last:	Amount of last:	Amount of last:
1	1	0	1	3	2	6	11
Score (lower better):	Score (lower better):	Score (lower better):	Score (lower better):	Score (lower better):	Score (lower better):	Score (lower better):	Score (lower better):
108	97	60	104	143	117	155	131
Avg score:	Avg score:	Avg score:	Avg score:	Avg score:	Avg score:	Avg score:	Avg score:
4.153846154	3.730769231	2.4	4	5.5	4.5	6.2	5.24
Median score:	Median score:	Median score:	Median score:	Median score:	Median score:	Median score:	Median score:
5	3	2	4	6	4	7	5

		CODE SIMPLICITY MOST IMPORTANT					
	COMMENTING CODE VERY IMPORTANT						
			CODE PORTABILITY VERY IMP		CI VERY IMPORTANT		
NAMING CONVENTIONS							SHARED CODE OWNERSHIP CONFLICTIVE [1]
				UNIT TESTING			
						PAIR PROGRAMMING	
		[1]					
		Marked as most important by 4 and least important by 11					
		May have to do with team type					

C.1.4 Analyzing Q6.3 Raw Data

The following page shows how the raw answers, blurred to ensure anonymity, to Q6.3 were encoded into codes and then unified into categories. These finally created a summary. Note that more intermediate codes existed as this part of the analysis happened. For example, *"documenting code"* and *"adding comments to the code"* are answers from the participants that became codes. After reviewing all the answers, and taking into account the context of the question where it already asked to sort by importance commenting code, these codes were unified in a more general code or category as *"good and complete documentation"*.

Q6.3 Please add here any comment about your answer in the previous question about coding practices importance.

[illegible]	Human factors are more important than specific technical practices.
[illegible]	Good and complete documentation
[illegible]	Not disciplined with any coding standards other than some unit tests
[illegible]	Clear interfaces and well understood architectures
[illegible]	Good and complete documentation
[illegible]	Code review to increase quality of code
[illegible]	Clear interfaces and well understood architectures
[illegible]	
[illegible]	Criticizing the survey

Summary 9 comments:

- Human factors are more important than specific technical practices.
- 2 responses added that good and complete documentation is important.
- 2 responses added that clear interfaces and well understood architectures are important.
- One response added that they have not been disciplined with any coding standard other than some unit tests.
- One response added that code reviews to increase the quality of the code is important.
- One response criticized the question itself considering the concepts incommensurable. Which is a fair critic, but the question arose from looking for a way to compare the trade-offs between these concepts.

C.1.5 Final Interpretation

The outcome from analyzing both parts of the questions is found in subsubsection 8.3.3.6. The trade offs in between the concepts and theories of why these exist are proposed. Moreover, the data was plotted in a user-friendly manner.

BIBLIOGRAPHY

[1] J. L. Gustetic, J. Crusan, S. Rader, and S. Ortega, "Outcome-driven open innovation at nasa," *Space Policy*, vol. 34, pp. 11–17, 2015.

[2] M. Buehler, K. Iagnemma, and S. Singh, *The 2005 DARPA grand challenge: the great robot race*, vol. 36.
Springer, 2007.

[3] M. Buehler, K. Iagnemma, and S. Singh, *The DARPA urban challenge: autonomous vehicles in city traffic*, vol. 56.
Springer, 2009.

[4] E. Krotkov, D. Hackett, L. Jackel, M. Perschbacher, J. Pippine, J. Strauss, G. Pratt, and C. Orlowski, "The darpa robotics challenge finals: Results and perspectives," *Journal of Field Robotics*, vol. 34, no. 2, pp. 229–240, 2017.

[5] S. Thrun, "Why we compete in darpa's urban challenge autonomous robot race," *Communications of the ACM*, vol. 50, no. 10, pp. 29–31, 2007.

[6] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa, "Robocup: The robot world cup initiative," in *Proceedings of the first international conference on Autonomous agents*, pp. 340–347, 1997.

[7] H. Kitano and S. Tadokoro, "Robocup rescue: A grand challenge for multiagent and intelligent systems," *AI magazine*, vol. 22, no. 1, pp. 39–39, 2001.

[8] T. van der Zant, T. Wisspeintner, and P. Lima, *Robocup@ home: Creating and benchmarking tomorrows service robot applications*.
INTECH Open Access Publisher, 2007.

[9] G. K. Kraetzschmar, N. Hochgeschwender, W. Nowak, F. Hegger, S. Schneider, R. Dwiputra, J. Berghofer, and R. Bischoff, "Robocup@ work: competing for the factory of the future," in *Robot Soccer World Cup*, pp. 171–182, Springer, 2014.

[10] A. Ferrein and G. Steinbauer, "20 years of robocup," *KI-Künstliche Intelligenz*, vol. 30, no. 3-4, pp. 225–232, 2016.

[11] U. Visser and H.-D. Burkhard, "Robocup: 10 years of achievements and future challenges," *AI magazine*, vol. 28, no. 2, pp. 115–115, 2007.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[13] E. Osawa, H. Kitano, M. Asada, Y. Kuniyoshi, and I. Noda, "Robocup: The robot world cup initiative," in *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-1996), Kyoto, Japan*, pp. 9–13, 1996.

[14] T. Balch and H. Yanco, "Ten years of the aaai mobile robot competition and exhibition," *AI Magazine*, vol. 23, no. 1, pp. 13–13, 2002.

[15] R. Organization, "Robocup objective web page." https://www.robocup.org/objective/, 2020.

[16] P. M. Pagnucco, "Robocup 2019, business events sydney." https://businesseventssydney.com.au/media/latest-news/australia-faces-robot-invasion-in-2019, 2020.

[17] L. Iocchi, D. Holz, J. Ruiz-del Solar, K. Sugiura, and T. Van Der Zant, "Robocup@ home: Analysis and results of evolving competitions for domestic and service robots," *Artificial Intelligence*, vol. 229, pp. 258–281, 2015.

[18] D. Holz, L. Iocchi, and T. Van Der Zant, "Benchmarking intelligent service robots through scientific competitions: The robocup@ home approach," in *2013 AAAI Spring Symposium Series*, 2013.

[19] M. Matamoros, C. Rascon, J. Hart, D. Holz, and L. van Beek, "Robocup@home 2018: Rules and regulations." http://www.robocupathome.org/rules/2018_rulebook.pdf, 2018.

[20] A. K. Pandey and R. Gelin, "A mass-produced sociable humanoid robot: pepper: the first machine of its kind," *IEEE Robotics & Automation Magazine*, vol. 25, no. 3, pp. 40–48, 2018.

[21] S. Robotics, "Pepper robot presentation." https://www.softbankrobotics.com/emea/en/pepper, (accessed: 15.01.2020).

[22] M.-A. Williams, "Uts unleashed! website." https://utsunleashed.webnode.com, (accessed: 15.01.2020).

[23] M.-A. Williams, "The magic lab website." https://www.themagiclab.org, (accessed: 15.01.2020).

[24] P. Checkland and S. Holwell, *Action Research*, pp. 3–17.
Boston, MA: Springer US, 2007.

[25] M. Riel, "Action research cycles." http://cadres.pepperdine.edu/ccar/define.html, (accessed: 15.01.2020).

[26] M. Staron, "Action research as research methodology in software engineering," in *Action Research in Software Engineering*, pp. 15–36, Springer, 2020.

[27] P. S. M. Dos Santos and G. H. Travassos, "Action research can swing the balance in experimental software engineering," in *Advances in computers*, vol. 83, pp. 205–276, Elsevier, 2011.

[28] K. Charmaz, J. A. Smith, R. Harre, and L. van Langenhove, "Rethinking methods in psychology," *Grounded Theory. London, UK: Sage*, 1995.

[29] K. Charmaz and L. L. Belgrave, "Grounded theory," *The Blackwell encyclopedia of sociology*, 2007.

[30] B. G. Glaser, A. L. Strauss, and E. Strutzel, "The discovery of grounded theory; strategies for qualitative research," *Nursing research*, vol. 17, no. 4, p. 364, 1968.

[31] Y. Chun Tie, M. Birks, and K. Francis, "Grounded theory research: A design framework for novice researchers," *SAGE open medicine*, vol. 7, p. 2050312118822927, 2019.

[32] A. Bryant and K. Charmaz, *The Sage handbook of grounded theory*.
Sage, 2007.

[33] K. Charmaz, *Constructing grounded theory: A practical guide through qualitative analysis*.
Sage, 2006.

[34] K. Charmaz and A. Bryant, "Grounded theory and credibility," *Qualitative research*, vol. 3, pp. 291–309, 2011.

[35] M. Birks and J. Mills, *Grounded theory: A practical guide*.
Sage, 2015.

[36] J. Corbin and A. Strauss, *Basics of qualitative research: Techniques and procedures for developing grounded theory*.
Sage, 2014.

[37] J. Mills and M. Birks, *Qualitative methodology: A practical guide*.
Sage, 2014.

[38] B. Glaser, "Theoretical sensitivity: Advances in the methodology of grounded theory," 1978.

[39] M. Birks, Y. Chapman, and K. Francis, "Memoing in qualitative research: Probing data and processes," *Journal of research in nursing*, vol. 13, no. 1, pp. 68–75, 2008.

[40] B. Dick, "Ar and grounded theory," in *Pap Prep Res Symp Aust New Zeal ALARPM/SCIAR Conf [Internet]*, pp. 4–5, 2003.

[41] L. Lingard, M. Albert, and W. Levinson, "Grounded theory, mixed methods, and action research," *Bmj*, vol. 337, p. a567, 2008.

[42] M. A. Abdel-Fattah, "Grounded theory and action research as pillars for interpretive information systems research: A comparative study," *Egyptian Informatics Journal*, vol. 16, no. 3, pp. 309–327, 2015.

[43] C. Larman and V. R. Basili, "Iterative and incremental developments. a brief history," *Computer*, vol. 36, no. 6, pp. 47–56, 2003.

[44] J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, P. Naur, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, *et al.*, "Report on the algorithmic language algol 60," *Numerische Mathematik*, vol. 2, no. 1, pp. 106–136, 1960.

[45] W. W. Royce, "Managing the development of large software systems: concepts and techniques," in *Proceedings IEEE WESCON 26 (August)*, pp. 1–9, 1970.

[46] H. D. Mills, "Top down programming in large systems," *Debugging techniques in large systems*, pp. 41–55, 1971.

[47] S. Hekmatpour, "Experience with evolutionary prototyping in a large software project," *ACM SIGSOFT Software Engineering Notes*, vol. 12, no. 1, pp. 38–41, 1987.

[48] B. W. Boehm, "A spiral model of software development and enhancement," *Computer*, vol. 21, no. 5, pp. 61–72, 1988.

[49] P. Rook, "Controlling software projects," *Software Engineering Journal*, vol. 1, no. 1, pp. 7–16, 1986.

[50] J. Kerr and R. Hunter, *Inside Rad: How to Build Fully Functional Computer Systems in 90 Days or Less*.
McGraw-Hill, 1993.

[51] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, *et al.*, "Manifesto for agile software development," 2001.

[52] J. Stapleton, "Dsdm: Dynamic systems development method," in *Proceedings Technology of Object-Oriented Languages and Systems. TOOLS 29 (Cat. No. PR00275)*, pp. 406–406, IEEE, 1999.

[53] K. Schwaber, "Scrum development process," in *Business object design and implementation*, pp. 117–134, Springer, 1997.

[54] C. Larman, *Agile and iterative development: a manager's guide*.
Addison-Wesley Professional, 2004.

[55] K. Beck, "Embracing change with extreme programming," *Computer*, vol. 32, no. 10, pp. 70–77, 1999.

[56] P. Coad and S. Palmer, "Feature-driven development," *Java Modeling in Color with UML*, pp. 182–203, 1999.

[57] M. Fowler, J. Highsmith, *et al.*, "The agile manifesto," *Software Development*, vol. 9, no. 8, pp. 28–35, 2001.

[58] C. Edeki, "Agile unified process," *International Journal of Computer Science*, vol. 1, no. 3, pp. 13–17, 2013.

[59] S. W. Ambler and M. Lines, *Disciplined agile delivery: A practitioner's guide to agile software delivery in the enterprise*.
IBM press, 2012.

[60] C. Larman and B. Vodde, *Large-scale scrum: More with LeSS*.
Addison-Wesley Professional, 2016.

[61] R. Brenner and S. Wunder, "Scaled agile framework: Presentation and real world example," in *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 1–2, IEEE, 2015.

[62] M. Poppendieck and T. Poppendieck, *Lean software development: an agile toolkit*.
Addison-Wesley, 2003.

[63] D. J. Anderson, *Kanban: successful evolutionary change for your technology business*.
Blue Hole Press, 2010.

[64] M. Fowler and M. Foemmel, "Continuous integration," 2006.

[65] J. Humble and D. Farley, *Continuous delivery: reliable software releases through build, test, and deployment automation*.
Pearson Education, 2010.

[66] D. Astels, *Test driven development: A practical guide*.
Prentice Hall Professional Technical Reference, 2003.

[67] A. Cockburn, *Crystal clear: A human-powered methodology for small teams: A human-powered methodology for small teams*.
Pearson Education, 2004.

[68] S. A. Qurashi and M. Qureshi, "Scrum of scrums solution for large size teams using scrum methodology," *arXiv preprint arXiv:1408.6142*, 2014.

[69] J. Sutherland, "Scrum@ scale guide," *The definitive guide to Scrum@ Scale: scaling that works. Scrum Inc*, 2019.

[70] K. Schwaber and M. Beedle, *Agile software development with Scrum*, vol. 1. Prentice Hall Upper Saddle River, 2002.

[71] D. Wells, "Extremeprogramming.org," 2000.

[72] K. Beck, *Extreme programming explained: embrace change*. Addison-Wesley professional, 2000.

[73] A. Alliance, "Definition of extreme programming by the agile alliance." https://bit.ly/3bkHnRN, (accessed: 27.05.2020).

[74] F. Oliveira, A. Goldman, and V. Santos, "Managing technical debt in software projects using scrum: An action research," in *2015 Agile Conference*, pp. 50–59, IEEE, 2015.

[75] T. Dingsøyr, G. K. Hanssen, T. Dybå, G. Anker, and J. O. Nygaard, "Developing software with scrum in a small cross-organizational project," in *European Conference on Software Process Improvement*, pp. 5–15, Springer, 2006.

[76] P. Abrahamsson, "Extreme programming: First results from a controlled case study," in *Proceedings 29th Euromicro Conference*, p. 259, IEEE, 2003.

[77] A. Sandberg, L. Pareto, and T. Arts, "Agile collaborative research: Action principles for industry-academia collaboration," *IEEE software*, vol. 28, no. 4, pp. 74–83, 2011.

[78] T. Dybå and T. Dingsøyr, "Empirical studies of agile software development: A systematic review," *Information and software technology*, vol. 50, no. 9-10, pp. 833–859, 2008.

[79] T. Van der Zant and P. G. Plöger, "Lightweight management–taming the robocup development process," in *Robot Soccer World Cup*, pp. 577–584, Springer, 2005.

[80] R. Gerndt, I. Schiering, and J. Lüssem, "Elements of scrum in a students robotics project: a case study," *Journal of Automation Mobile Robotics and Intelligent Systems*, vol. 8, 2014.

[81] N. Tomatis, R. Philippsen, B. Jensen, K. O. Arras, G. Terrien, R. Piguet, and R. Siegwart, "Building a fully autonomous tour guide robot: Where academic research meets industry," in *Proceedings of the 33rd International Symposium on Robotics (ISR)*, pp. 109–134, ISR, 2002.

[82] R. Gerndt, M. Paetzel, J. Baltes, and O. Ly, "Bridging the gap-on a humanoid robotics rookie league," in *Robot World Cup*, pp. 193–204, Springer, 2018.

[83] A. Paraschos, N. I. Spanoudakis, and M. G. Lagoudakis, "Model-driven behavior specification for robotic teams.," in *AAMAS*, pp. 171–178, 2012.

[84] E. Eaton, "Teaching integrated ai through interdisciplinary project-driven courses," *AI Magazine*, vol. 38, no. 2, pp. 13–21, 2017.

[85] U. K. Bindl and S. K. Parker, "Proactive work behavior: Forward-thinking and change-oriented action in organizations.," in *APA handbook of industrial and organizational psychology, Vol 2: Selecting and developing members for the organization.*, pp. 567–598, American Psychological Association, 2011.

[86] S. Overflow, "Stack overflow 2018 developers survey." https://insights.stackoverflow.com/survey/2018#development-practices, (accessed: 15.01.2020).

[87] M. Loukides, *What is DevOps?* " O'Reilly Media, Inc.", 2012.

[88] S. García, D. Strüber, D. Brugali, T. Berger, and P. Pelliccione, "Robotics software engineering: A perspective from the service robotics domain," *arXiv preprint arXiv:2006.10608*, 2020.

[89] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, Japan, 2009.

[90] T. Niemueller, A. Ferrein, D. Beck, and G. Lakemeyer, "Design principles of the component-based robot software framework fawkes," in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, pp. 300–311, Springer, 2010.

[91] H. Bruyninckx, "Orocos: design and implementation of a robot control software framework," in *Proceedings of IEEE International Conference on Robotics and Automation*, Citeseer, 2002.

[92] S. Robotics, "Naoqi framework." http://doc.aldebaran.com/1-14/dev/naoqi/index.html, (accessed: 15.01.2020).

[93] Google, "Blockly." https://developers.google.com/blockly, (accessed: 15.01.2020).

[94] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, *et al.*, "Scratch: programming for all," *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, 2009.

[95] L. van Beek, D. Holz, M. Matamoros, C. Rascon, , and S. Wachsmuth, "Robocup@home 2017: Rules and regulations." http://www.robocupathome.org/rules/2017_rulebook.pdf, 2017.

[96] Linuxize.com, "rsync command explanation." https://linuxize.com/post/how-to-use-rsync-for-local-and-remote-data-transfer-and-synchronization, (accessed: 15.01.2020).

[97] Linuxize.com, "scp command explanation." https://linuxize.com/post/how-to-use-scp-command-to-securely-transfer-files, (accessed: 15.01.2020).

[98] gitscm.com, "git submodules explanation." https://git-scm.com/book/en/v2/Git-Tools-Submodules, (accessed: 15.01.2020).

[99] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley, and Y. Bengio, "Theano: new features and speed improvements," *arXiv preprint arXiv:1211.5590*, 2012.

[100] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pp. 265–283, 2016.

[101] V. A. Ziparo, L. Iocchi, and D. Nardi, "Petri net plans," in *Proceedings of Fourth International Workshop on Modelling of Objects, Components, and Agents (MOCA)*, pp. 267–290, 2006.

[102] M. Cashmore, M. Fox, D. Long, D. Magazzeni, B. Ridder, A. Carrera, N. Palomeras, N. Hurtos, and M. Carreras, "Rosplan: Planning in the robot operating system," in *Twenty-Fifth International Conference on Automated Planning and Scheduling*, 2015.

[103] D. E. King, "Dlib-ml: A machine learning toolkit," *The Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009.

[104] M. Keller, F. Hoffmann, C. Hass, T. Bertram, and A. Seewald, "Planning of optimal collision avoidance trajectories with timed elastic bands," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 9822–9827, 2014.

[105] S. Pütz, J. S. Simón, and J. Hertzberg, "Move base flex," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3416–3421, IEEE, 2018.

[106] M. Matamoros, C. Rascon, S. Wachsmuth, A. W. Moriarty, J. Kummert, J. Hart, S. Pfeiffer, M. van der Brugh, and M. St-Pierre, "Robocup@home 2019: Rules and regulations." http://www.robocupathome.org/rules/2019_rulebook.pdf, 2019.

[107] D. Huggins-Daines, M. Kumar, A. Chan, A. W. Black, M. Ravishankar, and A. I. Rudnicky, "Pocketsphinx: A free, real-time continuous speech recognition system for hand-held devices," in *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, vol. 1, pp. I–I, IEEE, 2006.

[108] S. Pfeiffer, D. Ebrahimian, S. Herse, T. N. Le, S. Leong, B. Lu, K. Powell, S. A. Raza, T. Sang, I. Sawant, *et al.*, "Uts unleashed! robocup@home sspl champions 2019," in *Robot World Cup*, pp. 603–615, Springer, 2019.

[109] J. Ruiz-del Solar, "Visual slam-based localization and navigation for service robots: The pepper case," *RoboCup 2018: Robot World Cup XXII*, vol. 11374, p. 32, 2019.

[110] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7263–7271, 2017.

[111] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh, "Openpose: realtime multi-person 2d pose estimation using part affinity fields," *arXiv preprint arXiv:1812.08008*, 2018.

[112] M. Labbé and F. Michaud, "Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," *Journal of Field Robotics*, vol. 36, no. 2, pp. 416–446, 2019.

[113] C. E. Larson, C. Larson, and F. M. LaFasto, *Teamwork: What must go right/what can go wrong*, vol. 10.
Sage, 1989.

[114] G. Asproni, "Motivation, teamwork, and agile development," *Agile Times*, vol. 4, no. 1, pp. 8–15, 2004.

[115] S. McConnell, *Code complete*.
Pearson Education, 2004.

[116] R. C. Martin, *Clean code: a handbook of agile software craftsmanship*.
Pearson Education, 2009.

List of Figures

FIGURE **Page**

List of Tables