EasyChair Preprint

№ 7632

Wirom2.0: an Extensible System for Mission Planning of Heterogeneous Multi-Robot Teams

Gunnar Kleiven

March 28, 2022

Wirom2.0: An Extensible System for Mission Planning of Heterogeneous Multi-Robot Teams

Gunnar Fimreite Kleiven

Western Norway University of Applied Sciences, Bergen, Norway
gkl004@uib.no

Abstract

Robotics software systems are becoming increasingly more complex, which makes the development progressively more challenging. Different robots have different requirements, and they often require to cooperate together to achieve a common goal. This thesis presents Wirom2.0, an extensible system for mission planning of heterogeneous Multi-Robot Teams. By using the principles of Model Driven Software Engineering Wirom2.0 simplifies the mission planning of multi-robot teams by creating useful abstractions and facilitate for great extensibility.

Contents

1 Introduction

1.1 Background

Robotics systems is a field in software development which is becoming increasingly more complex, and therefore makes the development progressively more challenging. Historically, robotics development is a field which has struggled with defining strong development methods. The industry and research have been influenced by ”ad hoc” solutions, where the demanding needs has made it difficult for common solutions to emerge. This is slowing down the evolvement of

industry standards, as the developers are more likely to write their own architecture and code [4]. The slow progression is lacking behind the progression of traditional software development in general. However, researchers and engineers in the robotics community has become more aware that it makes sense to spend time developing reusable software.

The robots themselves has seen great development in since their early days. Previously, robots were mostly found in an industrial setting, doing repetitive work with minimal human interaction. Nowadays, robots have become much more versatile and are used in many other fields with countless applications. This creates a lot of new challenges, with roots in several different domains. Software for autonomous is created to function in varying and challenging environments, and this generates new levels of requirements for reliability, fault tolerance and safety. Furthermore, these systems often need to process data asynchronously, as well as having a demanding need for communication solutions in the vast collection of different components. Adding to all of this, the software often needs to be able to change system configurations and goals at run-time, without the possibility to redeploy code.

To manage the complexity of robot software systems, there is a need for engineering of better modeling languages and simulation tools. Achieving code which can be reused on different platforms generates the need for creating abstractions for the robotic behavior. This thesis will focus on creating some of these abstractions in the robotics domain. In Wirom2.0, one of the goals is to simplify a mission planning system, featuring a set of heterogeneous robots.

1.2 Motivation

The main motivation for this thesis is to contribute to some of the challenges in robotics software development by further developing a mission planning system for multi-robot teams. One of the ideas is to enhance the system by automating some of the expendabilities for the program, to make it more sustainable to change and raising its abstraction levels. Another motivation is to look more into how we can use Model Driven Software Engineering approaches, and especially Domain Specific Languages, to automate some of the functionalities of the system.

The resulting prototype in this thesis should be usable for users with different knowledge of both programming in general and the robotics domain. Novice users should be able to intuitively generate missions for the given robots, even without prior programming knowledge at all. Users experienced in the robotics domain should be able to further expand some of the content functionalities for the robots in the system, and the system should facilitate this by providing code generation and extensibility.

1.3 Contributions

The foundations for this thesis was created by Joakim Grutle [6]. In his Master Thesis he developed a mission planning system for a heterogeneous multi-robot setup, using the concepts of Model Driven Software Engineering and low-code development. The result is Wirom, with a web-based interface where the user can generate robot missions, task allocation and task development. One of the core ideas of his thesis is to use abstractions to help the users utilize these functionalities.

This thesis will use his project and code as a starting basis, and will both improve upon some of his functionalities as well as adding new features based on the motivation and research questions explained in these sections. As with most technologie development moves with a rapid pace, which will make some of the currently used technologies and solutions outdated. This means that while the main goal will be to add functionalities, some parts of the previous project is subject to change as well.

1.4 Research Questions and Methodology

To achieve what has been set out in the background and motivation section, the main research question for this master thesis will be:

- **RQ1:** *How can we use Model Driven Software Engineering Principles to simplify and enhance the abstraction levels and the extensibility of a multi-robot mission planning system?*

Furthermore, this thesis will look at the following sub-questions:

- **RQ2:** *How can the users experiment with different task allocation algorithms to increase the efficiency of the missions?*
- **RQ3:** *How can the communication approach enhance the robot communication and coordination?*

The research method followed in this thesis will be a case study. In [9] Runeson and Höst looks at several definitions of case studies in software engineering and how the common consensus is how case studies investigates *"contemporary phenomena in their context"*. With the focus on studying phenomenon in its context, conducting the research as a case study will align well with what this thesis is trying to achieve through the development of Wirom2.0.

The data collection can be either quantitative or qualitative. Quantitative data is measured in numbers, while qualitative data is not numerical data but rather text derived from human interaction and perception. Although the most common form of data in case studies is the use of qualitative data [9], they can be combined together to utilize the advantages from both methods [10]. In the case of this thesis, the data will be collected through developing useful mission examples and use cases of the extensibility of the system.

2 Background

To get a better understanding of both the introduction and motivation, as well as the next chapter about the implementation, it will be useful to get a short overview of some of the theoretical background. In the following sections there will be some brief walk-through of some of the most important concepts.

2.1 Robotics

2.1.1 Taxonomy for Multi-Robot Task Allocation

In [5], Gerkey and Matarić defines a formal taxonomy for task allocation in multi-robot systems, which they refer to as Multi-Robot Task Allocation (MRTA). In MRTA the tasks, robots and their assignments are categorized, each with two categories:

- The *tasks* can either be a **Single-robot (SR) task**, which are assignments where only one robot is required, or it can be a **Multirobot (MR) task**, which are tasks where several robots are required for it to be executed.
- The *robots* can be either **Single-task (ST) robots**, which are robots capable of handling only a single task at a time, or **Multitask (MT) robots**, which means robots capable of executing multiple tasks in parallel.

- Lastly, the *task assignment methods* are separated as **Instantaneous allocation (IA)** and **Time-extended assignment(TA)**. These are tasks that are assigned to be executed right away with no regards to the planning or future tasks, and an assignment of tasks where the planning and optimization of future tasks are taken into considerations, respectively.

These abbreviations can be combined in triples to form further descriptions for task allocation problems. The most common of them is the problem with Single-task robots, single-robot tasks and instantaneous assignment (ST-SR-IA). When changing any of these abbreviations to create different problems, they become inherently increasingly more challenging because of their nature. Although Wirom2.0 does not limit itself to just the ST-SR-IA problem, it will be used as an example because it's the simplest to implement and demonstrate. In this thesis we will not try to solve the ST-SR-IA problem because it has already been solved with several approaches [7]. However, we will facilitate for domain experts to add and experiment with task allocation algorithms, as finding new solutions and implementing them is by itself outside the scope of this thesis.

2.2 Model Driven Software Engineering

2.2.1 Overview

Model Driven Software Engineering (MDSE) is a methodology for developing software using models and transformations. MDSE is meant to increase productivity and efficiency in software development by emphasizing the usage of models to create complex software, with the core being the models as well as the transformations between the models [2]. It also focuses on applying *modeling* on different levels of abstractions an even on different levels of models. A model can itself be defined by another model, called a meta-model. Transformations defines the mapping between the different models and is the second important part of MDSE. Model transformations are defined using transformation languages, often provided by a modeling tool.

2.2.2 Domain Specific Languages

A Domain Specific Language (DSL) is a programming language which is specifically designed to solve problems in a particular domain. They are not necessarily designed to be able to solve any kind of problem, but rather focuses on a particular area of interest. Some known examples of DSLs are SQL and HTML [1]. A DSL is on the contrary side of a General Purpose Language (GPL), which as the name suggests is a programming language meant to solve all kinds of problems, e.g. Java and C++.

2.2.3 Model Driven Engineering in Robotics

Robotics software development is inherently complex. One reason is because of the different software components interacting in highly dynamic and uncertain environments [8]. Another reason is the need for collaboration of experts from different fields, e.g. mechanical and electrical engineering, as well as developers from software engineering. Because of these embedded challenges, some developers and researchers are looking at Model Driven Software Engineering to help solve some of these challenges. In [3], Brugali analyzed the role of MDSE technologies in robotics software engineering. He argues that one of the key characteristics that separates the systems for autonomous robots from other embedded systems is the large variety of functionalities that comes together to make op the robot capabilities, such as navigation and collecting

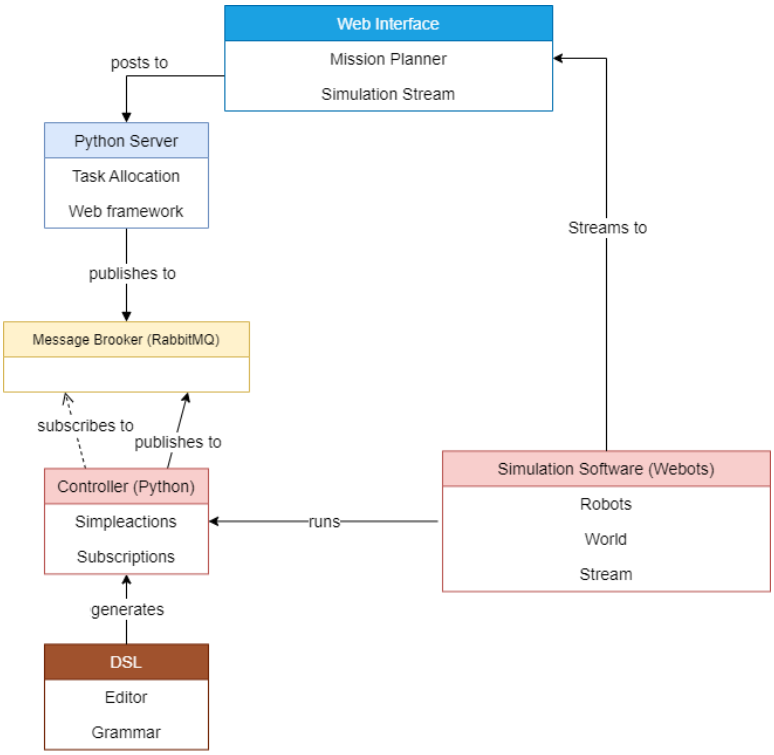

Figure 1: Architecture model of Wirom2.0

data from sensors and other input. He states that this creates a critical development phase when it comes to the software architecture of these systems, and that MDSE approaches can support these architecture designs by automating some of the complex tasks.

3 Wirom2.0 Implementation

The resulting system for trying to approach the mentioned challenges is Wirom2.0: An Extensible System for Mission Planning for heterogeneous multi-robot teams. Wirom2.0 lets users plan and execute robot missions through a high-level interface, through an easy-to-use low code platform. The missions are executed through a simulation software, which is streamed directly to the browser. Furthermore, the system provides for thorough extensibility, allowing users with some knowledge about the robotics domain to add both new and existing robots to the system. This is achieved by using a custom Domain Specific Language, which generates the needed code for adding new robots with simple commands.

3.1 Brief Overview

The overall system architecture is shown in Figure 1 as a top down view. At the top level, we have the web interface. It contains the missions, the tasks that constitutes the missions and the *simpleactions* which in turn defines the tasks. These are explained in section 3.2. Finally,

the web interface includes a streaming instance of the simulation view, which can be controlled from the browser.

The web interface communicates with a Python server using HTTP requests. The web interface sends the missions to the server, and the server publishes the missions to the correct communication channels. This is done using the publish/subscribe model. Task allocation is also handled by the server, by having the web interface sending a request to the server.

Going further down we have the simulation software and the robot controllers. The simulation software used is Webots, which runs the controllers implemented in Python. Each robot in the simulation world runs its own instance of a controller, and they communicate both with each other and the server through a message broker.

Finally, there is a DSL which can be run independently from the system. It can be used to generate the code for adding new robots of the types which already exists in the system. The DSL can be run through the Eclipse Modeling Framework, which provides an editor and an infrastructure to utilize the DSL.

3.2 Simpleactions

One of the most important elements in Wirom2.0 is the simpleactions. A *"simpleaction"* is a single action a robot can execute, e.g., *go forward*, *turn right* and *go to point*. The users adds these together to create missions. The robot controllers, explained in section 3.5, needs to implement each simpleaction for that robot type. When the user adds a robot of an existing type, this is done automatically. If the users add a brand new robot to the system, he/she needs to also add the implementation of as many simpleactions they want the robot to have the capabilities of doing. Furthermore, each simpleaction has a *cost* which plays a key role in the task allocation algorithm.

3.3 Frontend: Web Interface

The project was developed as a web application to make it simple to use and setup for the users. It is implemented using React, a popular JavaScript framework for building user interfaces. The web interface includes all the *simpleactions* which the user can put together to define the tasks. Several tasks can be but together to constitute a mission. A task is a list of actions for a robot to execute sequentially. Allocating these tasks can either be done manually by the user, or automatically by an implemented task allocation algorithm on the server. A screenshot of the top of the web interface is shown in Figure 2. Below is a window with a stream of the simulation, which looks like the screenshot of the demonstration in figure 3.

Lastly, the web interface includes a "Streaming Viewer", which is an instance of the robot simulation streamed from the simulation software to the browser. By streaming the simulation to the interface we get the advantage of having "everything in one place", and it creates a good opportunity to later deploy the system on a hosting service for the backend and the simulation software.

3.4 Backend: Python Server

The server is the main communication module between the web interface and the running simulation. It is implemented in Python and uses the web framework Flask to handle incoming HTTP requests from the web interface. These are sent as JSON objects, and the server forwards these messages by publishing them to the proper "topics". It also includes a task allocation

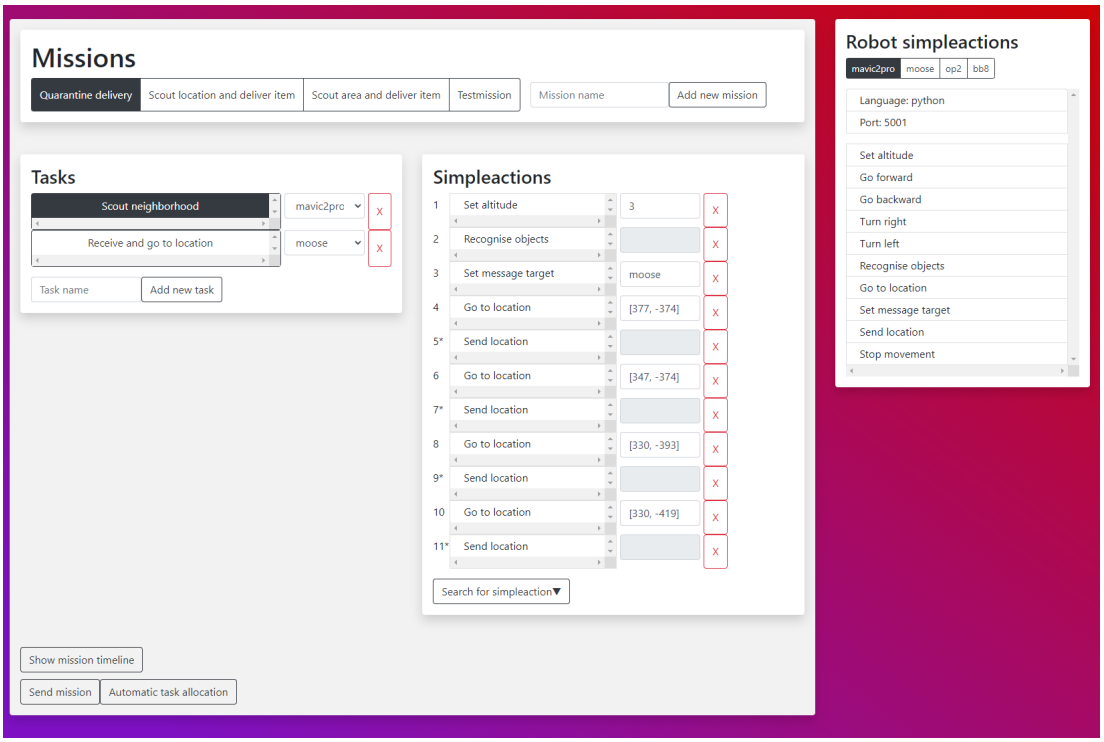

Figure 2: Screenshot from the Web Interface

manager, which can be called by the users to automatically allocate the tasks to the different robots.

RabbitMQ, an open source message broker, is used to implement the publish/subscribe model for the communication between the robots. This is a common pattern in robotics, and allows for a hybrid architecture where the communication is both centralized (by mainly going through the server) and also decentralized (the robots can publish messages to each other). Publish/Subscribe pattern works by having the robots choose which *topic* they want to subscribe to, which in Wirom2.0's case is their own robot-name queue, and listening for updates. A publisher publishes a message to this channel, and all the subscribers receive this message. A publisher can also subscribe to topics.

Lastly, when the server is started it runs a script which checks if there has been generated any robots from the DSL (explained in section 3.6). This script will find newly added robots and add their data to the necessary configuration files, such as the web interface lists and the simulation software world file.

3.5 Backend: Simulation

Using simulation software to simulate the robotics behavior has many advantages over using real robots. These includes both the cost savings and the opportunity to test functionalities which would otherwise be too challenging to do. In this thesis we are using Webots, an open-source robot simulator. Webots is well suited for research and educational purposes, with a large set of supported robots as well as a mode for streaming the simulation to the browser. A

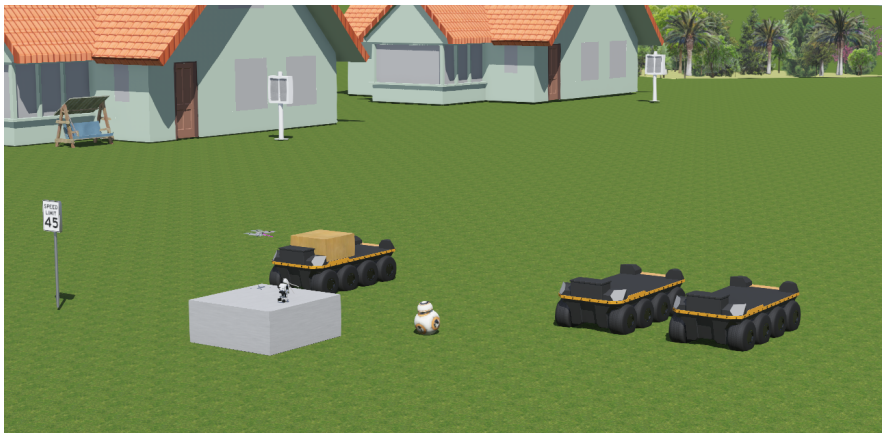

Figure 3: Screenshot of the Simulation in Webots, with some extra added robots

screenshot of the simulation can be seen in Figure 3

The individual robots in Webots are controlled by a *controller*, which is written in one of the supported languages. In our case, we are using Python. The controllers are an essential part of this thesis, as it is here the different kinds of robots in the heterogeneous teams have their own individual implementations. The capabilities of the robots, namely how many *simpleactions* they have implemented, play a large role in the potentials for the mission planning. One of the goals in this thesis is to facilitate for "domain experts" to be able to expand the repository of robots and their functionalities. It is therefore important with the expansion of both new and existing robots, where the controllers are the key part of Webots.

3.6 Robotgenerator - A Domain Specific Language and its implementation

One of the features of MDSE which is utilized in this project is the usage of DSLs for code generation. By using the DSL, we can create abstractions for the creation of adding new (existing) robots, by automating the process and removing some of the complexity for the users. The DSL created in this thesis, currently named "Robotgenerator", comes with simple commands with a familiar syntax. Without much prior knowledge about programming, the users should be able to easily utilize its features. Robotgenerator is created using Xtext, an Eclipse framework which covers all the aspects of the language infrastructure.

In figure 4 we can see a screenshot of the editor which is generated by the Eclipse framework. The commands seen are defined by the *grammar* of the language, which specifies the syntax. We can see that the editor includes functionalities common in an IDE (Integrated development environment), such as auto completion and error detection. When the file is saved, the DSL generates a controller file and a JSON file with the needed data. The command shown should be quite familiar to common programming syntax, with a "function call", and the arguments: ENUM parameter type, robot name as a string, X position value and Y position value. A semicolon marks the end of the line.

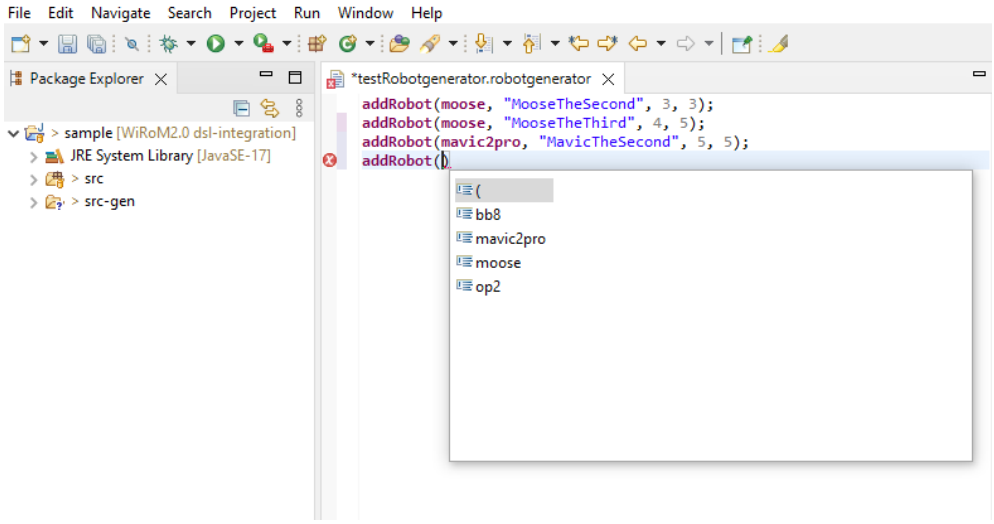

Figure 4: Usage of the DSL in the editor

3.7 Method

The software development method used in this work is an agile approach, leaning most towards Kanban. To organize the tasks, a Github project board has been used. Work-tasks are labeled with categories and their importance, which makes it easier to prioritize the work. There has also been frequent communication between the student and supervisor, with scheduled meetings once a week.

3.8 Further planned work

The plan going forward is to add another task allocation algorithm and create the possibility for the users to add their own, to try to answer the second research question in section 1.4. I will also focus on refactoring and making the current solutions better in general. As a huge part of this thesis is the focus on extensibility, it is important to lay a good foundation for future work on the project.

4 Results and Conclusions

The thesis has yet to finish its results. Data collection in this thesis will mainly be to create different mission scenarios, as well as user testing of the system. Since some of the focus is to have users with different "levels" of knowledge using the system, it will be useful to gather qualitative data from the user testing of the system. Ideally we will have users with different experience levels to test the individual features, both non-developers and students in software engineering.

Through the work in this thesis, we are further developing a mission planning system for multi-robot teams. The system have been updated to the newer versions, and there has been added a lot of improved functionalities. The end result is new and improved prototype for heterogeneous multi-robot teams, with simple usability and useful abstractions. It is extensible

by facilitating for adding any robot in the Webots ecosystem, scalable with its communication model. Wirom2.0 is hopefully an example of how MDSE principles and abstractions can help in the robotics developments of this category.

References

[1] Lorenzo Bettini. Implementing domain-specific languages with xtext and xtend : learn how to implement a dsl with xtext and xtend using easy-to-understand examples and best practices, 2016.

[2] Marco Brambilla, Jordi Cabot, Manuel Wimmer, and Luciano Baresi. *Model-Driven Software Engineering in Practice: Second Edition*. Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers, San Rafael, 2017.

[3] Davide Brugali. Model-driven software engineering in robotics: Models are designed to use the relevant things, thereby reducing the complexity and cost in the field of robotics. *Robotics & Automation Magazine, IEEE*, 22:155–166, 09 2015.

[4] Davide Brugali, Arvin Agah, Bruce MacDonald, Issa A. D. Nesnas, and William D. Smart. *Trends in Robot Software Domain Engineering*, pages 3–8. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[5] Brian P Gerkey and Maja J Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International journal of robotics research*, 23(9):939–954, 2004.

[6] Joakim Moss Grutle. Wirom: a high-level mission planning system for heterogeneous multi-robot simulations. Master's thesis, University of Bergen, 2020.

[7] G. Ayorkor Korsah, Anthony Stentz, and M. Bernardine Dias. A comprehensive taxonomy for multi-robot task allocation. *The International journal of robotics research*, 32(12):1495–1512, 2013.

[8] Arunkumar Ramaswamy, Bruno Monsuez, and Adriana Tapus. Model-driven software development approaches in robotics research. In *Proceedings of the 6th International Workshop on Modeling in Software Engineering*, MiSE 2014, page 43–48, New York, NY, USA, 2014. Association for Computing Machinery.

[9] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering : an international journal*, 14(2):131–164, 2008.

[10] C.B. Seaman. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering*, 25(4):557–572, 1999.