Getting Started with ROS2 Development: A Case Study of Software Development Challenges

Paulius Daubaris, Simo Linkola, Anna Kantosalo **and** Niko Mäkitalo
Department of Computer Science
University of Helsinki
Helsinki, Finland
first.last@helsinki.fi

***Abstract*—ROS2 has started to gain attention from the industry as it fosters robot software development. Companies seek to use ROS2 in real products, thus increasing the need for ROS2-related skills. We investigate what kind of challenges junior software developers – i.e. computer science majors on one of their last B.Sc study modules – encounter when learning robotics using ROS2. We conduct a case study with a group of five students with a project goal to develop a robotic application utilizing an agile software development process. We inquire about the challenges after the project using semi-structured interviews. By analyzing the interviews we identified 87 development challenge instances that can be roughly divided into challenges caused by insufficient or misinterpreted documentation and challenges encountered during the development process or usage of ROS2 and its packages. Hence, based on the results, we suggest that the ROS2 community should invest in an integrated resource to help with ROS2 development and patch up the currently fragmented documentation of individual packages in order for it to be adopted more easily as a technology by junior software developers.**

***Index Terms*—robot operating system, ROS, ROS2, software engineering**

I. Introduction

Developing robot software from scratch is a demanding task. Real industry-level applications are complex and expensive to develop, yet robots are increasingly applied to various real-world scenarios [9, 7, 12]. To alleviate the inherent complexity of the domain, Robot Operating System (ROS) and ROS2 [e.g. 11] emerged as technologies facilitating robot software development by providing an architecture, core technologies (e.g., messaging), as well as an open community for sharing software components with others. Therefore, ROS2 has started to get attention from the industry, and at the moment many are seeking to use it in real products. However, finding ROS2 developers may be challenging. Moreover, considering the nature of the robotics domain, junior developers are expected to encounter a steep learning curve that can make the learning process difficult. Hence, in this paper, our motivation is to study the challenges which developers might face when getting started with ROS2.

Robotics is a multidisciplinary field encompassing electrical engineering, mechatronics, electronics, mathematics and software engineering [10]. Learning to effectively work in such a multidisciplinary field poses various challenges for people with different skills and background knowledge. That is, to build robotic systems the developers need to cooperate with specialists from different fields or to maintain knowledge of multiple domains [13]. Fortunately, ROS and its successor ROS2 – a widely used technology to develop robot software [9, 3, 2] – attempt to reduce the identified knowledge gap. Nonetheless, entering a new domain can be challenging, especially for junior software developers.

In this paper, we provide a case study of a group of (junior) software developers learning robotics using ROS2 in a seven-week software development project. In other words, a group of computer science B.Sc students with varying backgrounds form a developer team to design and implement an autonomous mapping robot on the top of Turtlebot3 Burger platform per the client's requirements.

Our goal is to shed light on the most prominent challenges the computer science majors had when learning the robotics domain in a problem-solving style where they were required to build a full application satisfying specific requirements with no ready-made, complete solutions in hand – a typical situation in nearly any software development project. We focus on the technical development challenges dealing with ROS2 and its packages, which are caused by various reasons such as insufficient or missing documentation. We do not consider pedagogical issues, such as how to best introduce and teach robotics to computer science students.

We base our findings on interviews of a single student group (N=5) working on a single project. Each student kept a development diary during the course to reflect on the development process and the semi-structured interviews were held after the final software was delivered. The interviews were processed by transcribing them and the transcripts were encoded in multiple iterations to support thematic analysis. The coded transcripts were then analyzed and reoccurring topics identified.

The paper is structured as follows. In Section II, we introduce the motivation for this paper, related work and context of our case study, the robotics project. In Section III, we explain how the research was conducted and what methods we used to gather the necessary data. We present the results and insights in Section IV. After considering the results, we introduce threats to validity in Section V, following which, we discuss our findings in Section VI and conclude the paper in Section VII.

II. Background and Motivation

This section covers the motivation, related work and the robotics project the students set out to design and implement.

A. Motivation

The open source robotics middleware ROS has become a widely adopted technology in the robotics domain. It has created the opportunity for its users to rapidly prototype systems with pre-built packages conforming to specific functionalities [6]. Its successor, ROS2, has surfaced as a new emerging technology and a solution to its predecessor's shortcomings (e.g., single point of failure, security) [11].

Beyond minor changes and improvements, the main changes adopted by ROS2 include the new communication standard, the Data Distribution Service (DDS), which relieves systems from the single point of failure (i.e. the removal of the master node) inherent in the initial ROS version. The DDS also seeks to improve the security by providing built-in protection mechanisms and reliability due to its ability to operate in environments where network quality might be problematic [11].

Having two versions of the same technology, however, can have negative effects in the long term. For example, a vast array of knowledge has accumulated for ROS, some of which might not be relevant to ROS2. Expiring knowledge might not be an issue on its own, however, considering the differences and the prevalence of ROS, it might make it difficult for developers of varying expertise to find solutions to the similar challenges encountered with ROS2. Moreover, while the official source code of both technologies might be actively maintained, the same cannot be said for the package ecosystem [3]. Packages can be created by anyone without any maintenance obligations. Therefore, if a package has been assembled for, for example, the initial version of ROS, there are no guarantees that it will be updated for the succeeding version. Therefore, if the needed functionality is unavailable, it can lead to burdensome development, where the solution might need to be rewritten entirely or adjusted to fit the new requirements of ROS2.

Given these circumstances, we see the need to investigate learning experiences with ROS2, especially regarding junior developers. We consider that due to the rising need for robotic developers, we need to investigate the challenges especially junior developers face with adopting ROS2.

B. Related Work

ROS development challenges have been studied before. A similar study has been conducted where researchers with no background in ROS attempted to learn the technology using a five day paid course – "*ROS Basics in 5 Days (Python)*" by The Construction Sim[1] and recorded their experiences [1]. The authors identified issues ranging from documentation and development (e.g., ambiguous error messages) to how ROS abstractions prevent understanding the domain. Our work differs from the study by Canelas [1] in the duration of the learning, nature of the project, as well as the target technology. Although the authors do no explicitly mention the duration of their learning study, the name of the course implies that it is supposed to run for less than a week, while our project course ran for seven weeks. Regarding the nature of the project, the referenced study used multiple robots within a simulated environment. Our study, on the other hand, provided students with actual hardware to work with, thus, potentially expanding the surface of bottlenecks and challenges that could be encountered during the development. With consideration of the technology, we focused on ROS2, while the authors of the former study were investigating learning challenges related to its predecessor.

Estefo et al. [3], focused on a particular aspect of the ROS ecosystem – packages. They conducted interviews, focus-group interviews and online surveys to investigate the matter. The study included participants of varying expertise in ROS and examined experiences related to the package reuse. The researchers found that ROS package ecosystem has considerable shortcomings (e.g., lacking documentation, abandoned packages), which can significantly impact the usefulness and the perceived strength of the technology and provided recommendations on how the identified issues could be mitigated. With regard to the less experienced developers, it was noted that juniors were troubled by third-party packages and, specifically, their reuse due to various issues (e.g., a package being outdated or inability to understand how it works due to documentation).

Fischer-Nielsen et al. [4] investigated the case dependency bugs (e.g., a resource such as a data file being unavailable when needed) in the ROS ecosystem, reported their prevalence and highlighted challenges related to them. Although investigating challenges relevant to junior developers was not the core interest of the study, researchers indicated that dependency bugs indeed do pose challenges to the junior developers.

C. Robotics Project

In this section we provide an overview of the goals of the seven-week robotics project (10 ECTS credits) and the final software delivered to the clients (the writers of the paper). While the exact software requirements or the final software are not the focus of the paper, they help contextualizing our findings on the development challenges.

1) Description and Requirements: The goal of the project was to develop a ROS2-based application for Turtlebot3 Burger robots[2]. Turtlebot3 Burger is a low-cost hobby robot with a differential drive (two wheels that can be controlled separately and a balancing, freely rotating ball). The main components of the robot include: Raspberry Pi 3 Model B for the OS and software; OpenCR (Open-source Control Module for ROS) for the wheel motors and other possible sensors and actuators; and a 2D lidar. Additionally, for computer vision purposes, we included Raspberry Pi Camera Module V2 to each robot. Figure 1 shows a single-robot setup.

The high level functional requirements for the robotics application were the following:

[1]https://app.theconstructsim.com

[2]https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

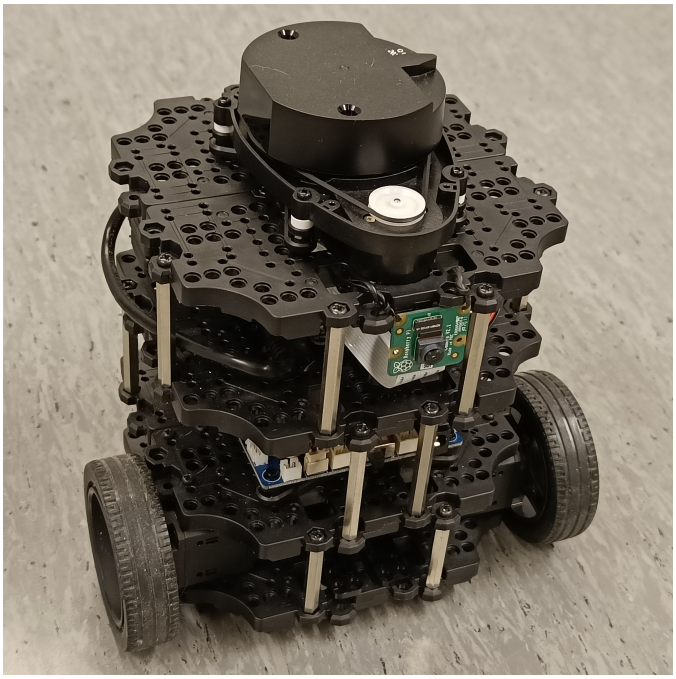


Fig. 1: A single Turtlebot3 Burger with an additional camera.

R1 The robot must autonomously explore its surroundings and form an internal map of the environment using the 2D lidar and a Simultaneous Localisation and Mapping (SLAM) implementation.

R2 The robot must be able to detect QR codes using its camera while exploring, and keep a database of seen QR codes and their locations with respect to the robot's internal map (see R1). The database must be kept up-to-date even if QR code locations move in the environment.

R3 The robot must stop its exploration and navigate next to a specific QR code (see R2) when instructed to do so by an external API call.

These three high level requirements were then broken down to more specific requirements and technical implementation details during the project. The use of existing resources, e.g. for SLAM, was highly recommended. Moreover, the robotics domain imposed some restrictions or guidelines for the development process. For example, even though the application was only required to work on the physical robots, we incentivised the students to use the Gazebo[3] simulation environment with an existing virtual implementation of the Turtlebot3 Burger for quicker development cycles.

The students were instructed to separate the main functionalities (navigation and SLAM, camera, and database) into their own ROS2 nodes, however, the exact architecture was left for the students to design and implement. Both ROS2 and Turtlebot3 were unfamiliar to the students. However, 4 out of 5 students had previous experience either directly with Raspberry Pis or with similar single-board computers.

[3]https://gazebosim.org/

2) Progress and Final Results: The students met with the client almost weekly during the project. In the conversational meetings, the client provided requirements for the software and their priorities, and the student team, with the help of their supervisor, then internally decided how to proceed with those requirements. In the successive meetings, the students showcased their progress and the client approved it, requested changes to the existing solutions, or adjusted the requirements.

The final result of the robotics project was a piece of software which worked both on the Gazebo simulator and the physical Turtlebot3. The software satisfied the three main requirements (R1–3) to an appropriate degree – although there were some known minor bugs left when the application was run on a physical robot – and followed our suggestion for the high-level architecture that separated SLAM, camera functionality and the database to their own nodes. The exact inner working of the nodes and their combined functionality was designed by the students during the project as we provided increasingly more specific requirement adjustments. While the final software ended up using only a few pre-built third-party ROS2 packages (*Nav2*[4], *Cartographer*[5]) and integrated some general Python packages such as *opencv-python*[6], the students explored many more options during the project.

III. Research Method

A. Interview Setup

To gather information about ROS2 development challenges from multiple angles after the project, we opted the semi-structured interviews to include selected themes. We first defined the general categories and then formed questions for each category, both of which are shown in Table I. We interviewed each student developer (referred to as Student 1–5 from now on) separately. We did not include the group supervisor because he did not participate in the hands-on development of the project. In case the student's answer was unclear initially, we asked for clarification or presented our interpretation of the answer and asked whether it was correct. Each interview was held in English and lasted roughly 20-30 minutes. The interviews were recorded and manually transcribed.

B. Data Analysis

The interview analysis was based on an open coding process [8] using an emergent categorization based on the topics discussed in the transcripts. The initial codes were acquired while reading the interviews and later refined by discussing and reviewing the existing set with the other authors of the paper in recurring meetings, repeating the process in multiple iterations. After a few iterations of coding, we decided to add categories and model a hierarchy that would enable us to distinguish common patterns among the responses. Mainly, we focused on identifying the challenges students had while developing the project using ROS2, which the codes aim to

[4]https://github.com/ros-planning/navigation2

[5]https://github.com/ros2/cartographer/tree/ros2/

[6]https://pypi.org/project/opencv-python/

TABLE I: A set of predefined interview questions

Category	Question
Background	Do you have any experience working on open source projects?
	Do you have any prior experience working with robot software?
	What was your role during the development of the project?
	Walk us through what was your robot development experience using ROS 2?
ROS2 as a concept	Could you explain ROS2 in your own words?
	Was it hard to understand the idea of ROS2 (e.g., what it is meant for)
ROS2 development	What concepts of ROS2 were hard/easy to grasp?
	What development bottlenecks have you encountered?
	What ROS2 feature did stand out the most during the project development?
ROS2 packages	Describe what was your experience while using ROS2 third-party packages?
	Did you encounter any difficulties when using a particular package?
	Were the packages sufficiently documented?
	Were the packages hard to integrate into the project?
	Were the packages easy to configure?
Documentation	How would you evaluate the documentation of ROS2?
	Was the information provided in the documentation sufficient for you to confidently start developing the project?
	Based on your experience during the project, what information do you think was missing from the documentation?
	Do you think the information provided in the documentation is easy to grasp?
Project and other	What was difficult to implement considering all the facilities that ROS2 offers (e.g., DDS for communication)?
	After the project course, would you feel comfortable using ROS2 for other robotic applications?
	Do you have any idea of how else you would implement the project?

reflect. In the end, we settled upon a total of 14 codes in three levels of hierarchy. Next, we elaborate on the findings.

IV. Results and Analysis

The results presented in this section describe the development challenge instances extracted from the interview transcripts. Our codes are organized into a three-level hierarchy, where the two interchangeable top levels identify whether the challenge instance was related to *ROS2 core* or *ROS2 packages* and if it was a *development & usage* or *documentation* issue. The third level denotes the concrete challenge. Figure 2 shows the groupings and the identified challenge instance counts.

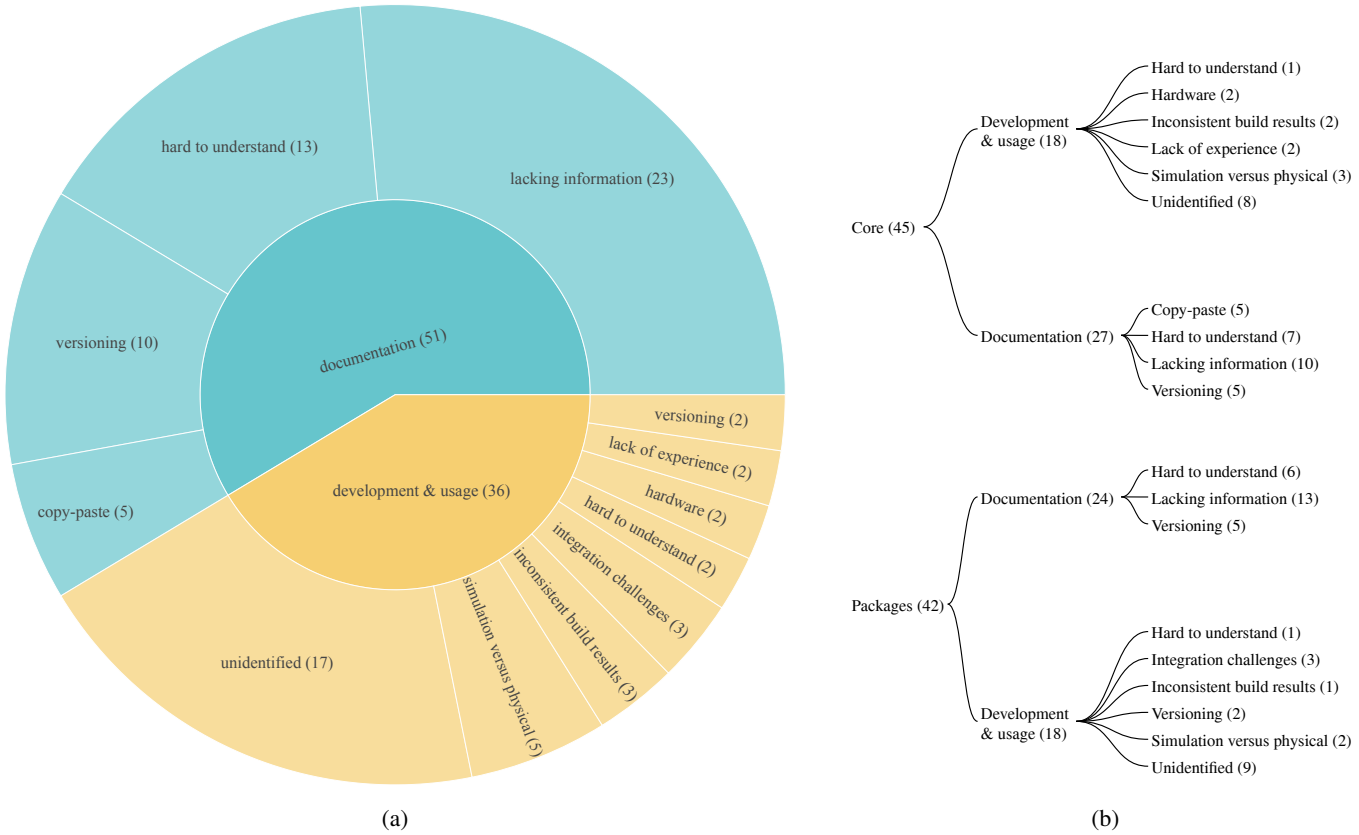
ROS2 core relates only to the core features of ROS2 and its documentation without the inclusion of external, third-party code libraries and/or packages and their documentation provided by its ecosystem. *ROS2 package* challenges are limited to the shortcomings of the third-party packages and their documentation. *Development & usage* refers to the challenges encountered when directly using the technology and *documentation* when searching for information about it.

The third level codes denoting a specific challenge are the following. *Lacking information* identifies insufficient information provided in the documentation or other resource (e.g., online forums). Issues related to the ROS2 versioning scheme have been applied to the *versioning* code. The *hard to understand* code expresses confusion and lack of intuitiveness when using ROS2 and/or applying its packages. A less frequent code, such as *simulation vs. physical*, expresses issues related to the inconsistent behavior of the system when applied in simulation and in a physical setting, whereas *inconsistent build results* identify non-deterministic outputs when building the system. The *hardware* code encompasses issues encountered when applying hardware to the system and *integration* indicates difficulties encountered when integrating, for example, a package. The *copy-paste* code embodies cases where students expressed shortcomings of the learning experience due to full examples provided that could be copied without the need to understand them. Regarding the experiences, *lack of experience* code indicates challenges where students felt the issue was related to their lack of experience with the ROS2 ecosystem rather than the ecosystem itself. Lastly, the *unidentified* code represents issues that were given no explanation during the interviews. For example "*We had lots of different errors or problems when working with the Nav2 [package]*" (Student 1).

Next, we consider the most prominent qualitative results of our case study.

A. ROS2 Core

1) Documentation: In general, when asked to assess ROS2 documentation, the majority of students evaluated it as very well-written: "*the documentation on its own is so, so very decent*" (Student 1). Nonetheless, besides praises, students that focused on the development of the project reported a shortage of tutorials on more practical aspects of the technology: "*I didn't find anything [about] tests. So, to create [them], I had to figure out how to do it on my own, and it was pretty hard, and I was unsure if it was the correct way to do it*" (Student 2). Otherwise, one student compared existing ROS2 tutorials provided on the official ROS2 website and described some as being less intuitive than the others: "*I did start looking into the services and the topics and I noticed that those were ...*



(a) (b)

Fig. 2: Groupings of the 87 identified development challenge instances: (a) on the left the main division is between the challenge instances in the development & usage and documentation, and (b) on the right between the challenge instances in ROS2 Core and ROS2 Packages.

much more complicated than the simple publisher, subscriber one" (Student 3). "*Well, the topics and services were easy to grasp, but [for] the last type of communication, I was unsure how to use [it], and we did not use it in the project. Also, in the code, calling the services was pretty unintuitive and was not as easy as just calling some sort of function. It was a bit more complicated and quite annoying*" (Student 2).

Upon asking whether the information provided was sufficient for project development, students argued that a tutorial showing how to connect prior knowledge would have been beneficial: "*my biggest complaint is that there's no bigger project example or any example where there are multiple nodes and how they work together*" (Student 4). "*I had some difficulties on how [to] make something bigger ... I would have liked to see more intermediate tutorials*" (Student 3).

When discussing documentation, a few students identified the learning process to be difficult since the tutorials provided complete examples that could be copied: "*[Y]ou can get the initial stuff, but it was pretty much copying general commands from the basic tutorials ..., you didn't really understand what was going on*" (Student 5). "*The ROS2 tutorials were like 'yeah, just copy-paste this code, and then we will quickly [go] through the code and move on. And also we will just copy and paste these terminal commands and go on'. Well, you did not learn there much*" (Student 2).

Although the documentation has been identified as well-written, based on the feedback received from all five students, we observe that junior software developers still encounter difficulties when trying to apply new knowledge.

> ***Observation 1.*** *ROS2 documentation is highly approachable to new users. However, some relevant documentation is missing, while existing code examples are not thought-provoking. It remains difficult to connect the knowledge acquired from the documentation and apply it to build a larger project encompassing the concepts described in the content without a proper example.*

2) Development in Simulation vs. Physical: Besides documentation issues, we expected students to encounter development bottlenecks throughout the project. When asked to identify them, the students reported inconsistencies between the simulation and the physical environment: "*[E]verything works differently on a physical robot than in the simulation. It's really fast to do things in the simulation, but when we try to run it on a physical robot, we get different results*" (Student 4). "*The implementation was pretty easy [when] working within the simulation, but we constantly ran into trouble trying to get things [to work] on a physical robot. That*

was the biggest single hurdle and that occurred constantly throughout the project" (Student 5). "*Some nodes worked differently in the Gazebo simulator in comparison to the actual robot*" (Student 2).

Although most students did not elaborate on what caused the different outcomes in these two environments, one identified that a part of the system was not working due to the lacking documentation in the Nav2 package: "*In Foxy [version of Nav2], there were no API commands, and it was difficult to understand which launch files to use ... because [they] were not documented ... and accidentally, we were using the wrong launch file for SLAM for the physical robot because it's different [for] physical and simulation. It cost us a lot of hours.*" (Student 4).

Not all the participants encountered this issue. However, it is mainly because Student 3 took on the managerial role, while Student 1 developed nodes and concentrated on the presentation aspects of the project. This leads us to the following observation.

Observation 2. *Developing a system using a simulator is quick and convenient. However, it is probable to encounter issues when deploying the code onto the physical hardware, even when the system appeared to work in the simulation.*

3) ROS versioning: Project participants reported the versioning aspect made development difficult. "*Some of the development bottlenecks were that some of the [packages] were lacking documentation or the tutorials were for ROS or an older or newer ROS2 version or they did not have something that we would have liked them to have*" (Student 2). When evaluating the documentation of ROS2, some students, based on the development experience, hypothesized that it would have been easier to work with the initial ROS version "*My answer would probably be different if we did this with ROS1. I felt like we would have a generally fairly good amount of documentation, and there was already such an amount of information online, people dealing with different stuff and so on, but with ROS2 – no*" (Student 5). One student identified ROS2 as a confusing platform. When asked to elaborate, they explained that the confusion was caused by many different distributions of ROS2: "*I think [it was confusing] mostly because there were so many versions of ROS2. For example, there is the Foxy one [that] we used, but then there [are] also 10 other bigger updates... And also because there is ROS. When you [search] for some problem, you might find the solution, but it probably won't be for the [distribution] you are using at that moment. The solutions ... spread across all the versions of the software. I felt that was a big issue.*" (Student 1).

Besides that, ROS2 versioning made it difficult to use certain packages because it was unclear for which ROS version or distribution it was made for: "*[W]hen we first started to get the camera working, we used ... V4L2 [package], and it was not that good, it did not work. It was also made for maybe some other version of ROS2 or other distribution*" (Student 1).

More than half of the participants reported development hardships caused by the ROS versioning scheme. We observe that such a scheme can often make the solutions to the previously encountered problems obsolete and make the development burdensome.

Observation 3. *Multiple distributions and versions of ROS can negatively impact the development process due to the expiration of the accumulated knowledge caused by the release of new versions of the technology.*

B. ROS2 Packages

1) Documentation: The students described difficulties while using ROS2 packages, many of which were related to their documentation. Since students used only a few packages, many answers identified shortcomings of a particular package. For example, a package such as Nav2 was both praised and criticized. The praise came mainly from students who did not work on the navigation part of the project. We did not ask how they came up with the conclusion, however, we speculate that those participants had only a brief encounter with its documentation and prematurely assumed it was sufficient for their use case. However, those students who did work with Nav2 or other ready-made packages mentioned that the documentation was insufficient: "*Nav2 documentation [is] bad at telling which part does what*". (Student 5). "*Some of the development bottlenecks were that some of the preexisting [packages] were lacking documentation*" (Student 2). "*In the beginning, we mostly used pre-made [packages] that ... somebody has been developing and has been nice enough to publish, but sometimes the documentation was non-existent.*" (Student 3). Even when it was sufficient to begin the development, some packages had relevant information missing: "*Nav2 was well documented, but it did not mention any of the errors that we got*" (Student 2).

A few cases reported by a couple of students mention that the developers of some packages omitted details beneficial to a junior, such as the information on basic usage: "*Sometimes it was very difficult to get [packages] working, I had an experience where one package said, basically to paraphrase ... 'start it like you normally would'. And [I], of course, I don't know how to normally start [it].*" (Student 3). Or the concepts behind the package: "*[An issue during the development was] to understand the role of Nav2 and SLAM and the combination of how they work together. What's the starting point?*" (Student 4). Moreover, when asked whether it was difficult to integrate a package due to lacking documentation: "*I think that was the hard part. Not really knowing how to or through what API to use them*" (Student 1).

Based on the feedback received from all students, ROS2 package documentation that was used in the project appeared to be a significant bottleneck during the development. It assumed a degree of expertise and omitted information relevant to junior software developers.

Observation 4. *Some ROS2 package documentation does not live up to the same standards as ROS2 documentation. Students identified that it is either lacking or non-existent. Moreover, even if a package is well-documented, it may omit details that matter to junior developers.*

2) Usage: Besides documentation issues, ROS2 package usage has caused some frustration as well. Mainly, students encountered packages that did not work properly or did not understand how to use them. A commonly identified pattern was to spend time trying to understand what went wrong and then fallback to re-implementing the package or some of its functionality from scratch if feasible: "*We got [the explorer package] working and it explored the entire area, but then it stopped, and we somehow had to figure out how to make it continue moving, but in the end, we could not do that and had to write our own node*" (Student 2). "*We had to adjust our own program because we did not really understand how it worked ... and in the end, we had to just make our own camera node, we could not get it [to work] properly*" (Student 1). Nonetheless, one student described the issue as not necessarily being the package but the lack of experience: "*My experience was that there were ... a lot of useful packages, but with my knowledge, I could not get them to work the way that I needed them to work.*" (Student 3).

When asked about the difficulties of using a particular package, one student mentioned that it was difficult to understand how to build a package properly: "*To get the video working in order to read the QR codes ... [we had to use a] package called V4L2. ... to get it working, I think there were four different things I had to pull from Git and users dependencies. ... All the instructions I had found ... had slight differences and in the end, I had to look up about four different tutorials and combine them in a way that I finally got them to work on our device ... And also when building and starting the node it threw a lot of error messages and at first, I thought, I must get rid of the error messages for it to work properly, and I spent about two hours on that when I finally realized that, [instead], I will try it with the error messages and everything worked perfectly. So, I think that was [difficult] for me to find out how to build it properly, so it actually works and then ignore non-relevant error messages.*" (Student 3).

Students working with third-party packages experienced the development process to be difficult to such a degree that they had to rewrite specific functionality by themselves to advance the project further.

Observation 5. *Some ROS2 packages are difficult to use due to lacking accessibility, which can significantly delay development advances.*

V. Threats to Validity

The main threat to the case study validity is the small sample size of five students working in the same group.

The students worked on the same project throughout the whole course. It was expected they would share ideas and their progress, discuss different tasks, and explain concepts behind ROS2 or its packages to one another. In a general sense, such collaboration is welcome. However, there is a risk that one student's experience shared with another could have skewed their perceived difficulties.

Considering the course itself, clients gave feedback and various instructions to the students. For example, we pointed what are the most commonly used packages that might be useful for them. Such input could have affected the general development process and, ultimately, the difficulties they encountered.

Another aspect is that the students used only a few packages during the project. Such circumstances prevent us from making definitive conclusions implying that the identified shortcomings apply to all ROS2 packages.

VI. Discussion

ROS2 and its ecosystem ease the development process and enable novice users to understand the underlying concepts of the technology and to rapidly construct robotic systems by leveraging third-party packages. However, several disadvantages persist. The findings presented in this paper indicate that there are multiple pain points regarding learning ROS2 related to its core and the packages.

With regard to the used ROS2 packages, mainly two topics were reoccurring: package documentation and their usage. Students identified the state of package documentation to be less polished than the official ROS2 documentation – often lacking or non-existent and also assuming a degree of expertise, thus omitting details relevant to the newcomers. Such aspects made the packages difficult to understand and use. However, because the participants used only a handful of third-party packages, these claims cannot be applied to the whole package ecosystem. Nonetheless, differently than with ROS2 core documentation, we see overlapping ideas in the related work (see e.g., [3]), where beginner users identified similar issues such as lacking documentation (Observation 4) or failure when attempting to reuse a package (Observation 5); or packages being outdated [5]. Considering the study has been conducted in 2019, it appears that the issue is still relevant and there is room for improvement. Although ROS accessibility issues have been acknowledged in several previous studies [6, 7], we identified that ROS2 core documentation was mostly sufficient (Observation 1), but the package documentation was in many cases scattered or insufficient (Observation 4). The latter observation is also backed up by the work of Canelas et al. [1], where the authors argue that the technology could benefit from improved documentation. In addition, the versioning scheme of ROS2 adds additional complexity to new adopters as finding the appropriate ROS2 distribution to work with may require its own exploration. However, definitive conclusions cannot be made because, to our knowledge, only a handful of studies exist that evaluate the accessibility of ROS2 and, more concretely, its documentation. Therefore, the conducted study should be expanded to verify the reliability of the presented results regarding the current state of ROS2 accessibility to novice users.

VII. Conclusions and Future Work

In this paper, we investigated ROS2-based software development challenges encountered by junior developers that have no experience in robot software development. After a seven-week software development project we conducted semi-structured interviews with each member of the developer team. During the analysis of the interview data, we distinguished issues related to two main categories: *ROS2 core* and *ROS2 packages*. *ROS2 core* relates only to the core features of ROS2 and its documentation without the inclusion of external, third-party code libraries and/or packages and their documentation provided by its ecosystem. *ROS2 package* challenges are limited to the shortcomings of the third-party packages and their documentation.

The results indicate that considering all the amenities ROS2 and its ecosystem provides, junior developers remain to encounter hardships that require them to seek answers elsewhere rather than the official documentation, rewrite code that is poorly documented or is difficult to integrate and seek alternatives and workarounds when no clear solution exists to the encountered issues. We draw the conclusion that while ROS2 makes robot development more approachable to the developers, the varying levels of package documentation and versioning-related problems decrease the accessibility for junior developers. As a result, we suggest (1) improving the documentation by developing an integrated resource to help with ROS2 development and the currently fragmented documentation of individual packages, and (2) standardizing the documentation process and providing best practices that package creators could follow. Nonetheless, definitive claims cannot be made and an expanded study should be conducted to see how well the results generalise.

For future work, we see the possibility to investigate the listed difficulties in more detail and consider how these shortcomings could be alleviated, thus improving the development process, especially for novice users. We also seek to put ROS2 into a broader context by conducting a more elaborate study regarding its technological maturity in the industrial context.

Acknowledgments

This work was supported by Business Finland and Academy of Finland project #328729.

References

[1] Paulo Canelas et al. “An Experience Report on Challenges in Learning the Robot Operating System”. In: *IEEE/ACM 4th International Workshop on Robotics Software Engineering*. 2022, pp. 33–38.

[2] Vincenzo DiLuoffo, William R Michalson, and Berk Sunar. “Robot Operating System 2: The need for a holistic security approach to robotic architectures”. In: *International Journal of Advanced Robotic Systems* 15.3 (2018), p. 1729881418770011.

[3] Pablo Estefo et al. “The Robot Operating System: Package reuse and community dynamics”. In: *Journal of Systems and Software* 151 (2019), pp. 226–242.

[4] Anders Fischer-Nielsen et al. “The Forgotten Case of the Dependency Bugs : On the Example of the Robot Operating System”. In: *IEEE/ACM 42nd International Conference on Software Engineering: Software Engineering in Practice*. 2020, pp. 21–30.

[5] L. Garber. “Robot OS: A New Day for Robot Design”. In: *Computer* 46.12 (2013), pp. 16–20.

[6] Nadia Hammoudeh Garcıa et al. “Bootstrapping MDE Development from ROS Manual Code - Part 1: Metamodeling”. In: *Third IEEE International Conference on Robotic Computing*. 2019, pp. 329–336.

[7] Nadia Hammoudeh Garcıa et al. “Bootstrapping MDE development from ROS manual code: Part 2—Model generation and leveraging models at runtime”. In: *Software and Systems Modeling* 20.6 (2021), pp. 2047–2070.

[8] Shahedul Huq Khandkar. “Open coding”. In: *University of Calgary* 23 (2009), p. 2009.

[9] Seulbae Kim and Taesoo Kim. “RoboFuzz: Fuzzing Robotic Systems over Robot Operating System (ROS) for Finding Correctness Bugs”. In: *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Singapore, Singapore: ACM, 2022, pp. 447–458.

[10] Sophia Kolak et al. “It Takes a Village to Build a Robot: An Empirical Study of The ROS Ecosystem”. In: *IEEE International Conference on Software Maintenance and Evolution*. 2020, pp. 430–440.

[11] Steven Macenski et al. “Robot Operating System 2: Design, architecture, and uses in the wild”. In: *Science Robotics* 7.66 (2022), eabm6074.

[12] Ivano Malavolta et al. “How do you Architect your Robots? State of the Practice and Guidelines for ROS-based Systems”. In: *IEEE/ACM 42nd International Conference on Software Engineering: Software Engineering in Practice*. 2020, pp. 31–40.

[13] Elisa Tosello, Nicola Castaman, and Emanuele Menegatti. “Using robotics to train students for Industry 4.0”. In: *IFAC-PapersOnLine* 52.9 (2019). 12th IFAC Symposium on Advances in Control Education, pp. 153–158.