UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

SOCIO-TECHNICAL ANALYSIS OF THE ROBOT OPERATING SYSTEM ECOSYSTEM FOR FOSTERING PARTICIPATION IN KNOWLEDGE SHARING ACTIVITIES

TESIS PARA OPTAR AL GRADO DE
DOCTOR EN CIENCIAS, MENCIÓN COMPUTACIÓN

PABLO IGNACIO ESTEFÓ CARRASCO

PROFESORES GUÍA:
JOCELYN SIMMONDS WAGEMANN
ROMAIN ROBBES

PROFESOR CO-GUÍA:
JOHAN FABRY

MIEMBROS DE LA COMISIÓN:
TOM MENS
SERGIO OCHOA DELORENZI
JUAN SANDOVAL ALCOCER

Este trabajo ha sido parcialmente financiado por ANID/CONICYT a través de la beca CONICYT PCHA/Doctorado Nacional 2015/21150762

SANTIAGO DE CHILE
2023

Resumen

ANÁLISIS SOCIO-TÉCNICO DEL ECOSISTEMA DEL ROBOT OPERATING SYSTEM PARA EL FOMENTO DE LA PARTICIPACIÓN EN ACTIVIDADES DE INTERCAMBIO DE CONOCIMIENTO

El desarrollo de software para robots presenta desafíos particulares: amplia variedad de disciplinas involucradas (ingeniería de software, ciencia de datos, mecatrónica, etc.) y uso de hardware heterogéneo (diferentes actuadores y sensores). La comunidad de robótica ha llegado al uso de Middleware para robótica como una solución al manejo de la complejidad del programar robots. Estas tecnologías consisten en una capa de abstracción entre el sistema operativo y las aplicaciones. Existen varios middlewares pero el más popular hoy es el Robot Operating System (ROS). Aun cuando existe una participativa comunidad detrás del éxito de ROS, existen muy pocos estudios respecto a ROS como un ecosistema de software.

En esta tesis presentamos un estudio en profundidad del ecosistema de ROS y propusimos un método para identificar miembros de la comunidad calificados para compartir conocimiento. Primero hicimos un estudio inicial sobre un artefacto de software propio de ROS: los ROS launch files. Estos son archivos de configuración usados para hacer el despliegue de procesos de un programa robótico. Encontramos suficiente evidencia para afirmar que un gran porcentaje de los paquetes de ROS presenta duplicación de código en sus launch files.

Posteriormente, indagamos aún más en la forma en que los usuarios interactúan con el middleware y en la comunidad en sí. Para esto hicimos un estudio de campo que combinaba aspectos cualitativos y cuantitativos. Los resultados nos permitieron destilar los cinco principales cuellos de botella en las dinámicas de colaboración del ecosistema ROS. Entre ellas destacan la *falta de tiempo* y la existencia de *paquetes abandonados*. Junto con esto, proponemos cinco recomendaciones para sobrellevar estos cuellos de botella, entre ellos destaca: *Recomendar Oportunidades de Contribución.*

Finalmente, nos enfocamos en esta última recomendación y presentamos un sistema de recomendación de usuarios calificados para responder preguntas en la plataforma de preguntas y respuestas (Q&A) ROS Answers. La idea principal fue distribuír la carga de trabajo recomendando usuarios con poca participación. A través de tres preguntas se probó la eficacia del método: saber si recomienda a los usuarios que contestaron correctamente la pregunta, saber si recomienda usuarios calificados y saber si ayuda a distribuir mejor la carga de trabajo.

Los resultados del trabajo realizado nos permiten concluir que el estudio de artefactos de software sí *permite detectar problemas en el ecosistema*, que los cuellos de botella en las contribuciones se pueden *atribuir a problemas socio-técnicos*, y que un sistema de recomendación basado en tags puede ser *insuficiente para recomendar usuarios calificados que redistribuya la carga de trabajo* en plataformas Q&A de robótica.

Abstract

Software development for robots presents particular challenges: a wide variety of involved disciplines (software engineering, data science, mechatronics, etc.) and the use of heterogeneous hardware (different actuators and sensors). The robotics community has arrived at the use of Robotic Middleware as a solution for handling this complexity. This technology consists of an abstraction layer between the operating system and the applications. There are many robotics middleware but the most popular is the Robot Operating System: ROS. Even though there is a participatory community behind ROS' success, there are very few studies about ROS as a software ecosystem.

In this dissertation, we present an in-depth study of the ROS ecosystem and we propose a method for identifying members of the community qualified to share knowledge. We first performed an initial study about a software artifact of ROS: the ROS launch files. They are configuration files used for the deployment of processes of a robotic program. We found enough evidence to affirm that an important percentage of ROS packages has a noticeable amount of code duplication in their launch files.

Later, we delved further into the way users interact with the middleware and in the community itself. We did a field study with a qualitative (interviews) and a quantitative (surveys) approach. With the result, we distilled the five principal bottlenecks in the collaboration dynamics of the ROS ecosystem. Among them are the *lack of time* and the existence of *abandoned packages*. Moreover, we propose five recommendations to overcome these bottlenecks, one of which is: *Recommendation of Collaboration Opportunities*.

Finally, we focused on this recommendation and developed a recommender system of qualified users to answer questions posted in the Q&A platform ROS Answers. The main idea was to better distribute the workload recommending users that are less participatory. We tested its effectiveness through three questions: to know if it actually recommends the users that answer correctly the question, to know if it recommends qualified users, and to know if it helps with a better distribution of workload.

The outcomes of the work carried out allow us to conclude that the study of software artifacts *do help detecting issues in the software ecosystem*, contribution bottlenecks can be *attributed to socio-technical issues* and that a recommender system based on tags can be *insufficient to recommend qualified users to redistribute the workload* in Q&A platforms of robotics.

Para Bruno.

Acknowledgments

En primer lugar quiero y debo agradecer a aquellas/os que están siempre conmigo. Mi madre quien me ha motivado y apoyado en este largo y, por qué no decirlo, dificilísimo camino que ha sido el doctorado. Sin sus palabras, acciones, consejos y cariño no estaría escribiendo esta última sección de la tesis. También agradecer a mi hermano Tomás con quien siempre comparto interesantes discusiones que me refrescan intelectualmente y ponen a prueba mis supuestos más arraigados. Eres para mi una inspiración y te deseo lo mejor ahora que ya comenzaste tu camino en la academia. Gracias por tu cariño. A la Giovi, mi compañera, por su atenta, paciente y constante compañía en este proceso: Te Amo. A la Jaquito con quien paso la mayor parte del tiempo y quien con su ternura logra bajarme el estrés con esas pausas de dos minutos de regaloneo a mitad del día. No puedo dejar de mencionar a la Tía Marité, Javiera, Camilo y Dani, que siempre estuvieron atentos a mis avances, pusieron oído a mis dificultades y alentaron mis progresos en la tesis.

Quisiera agradecer a mi primer profesor guía Johan Fabry: por apostar por mi en desarrollar el doctorado, por su constante apoyo, consejos en mi carrera y sus estrictas correcciones. Al profesor Romain Robbes por haberme acogido siendo mi segundo profesor guía: por su profesionalismo, su rigor, sus ideas que sin duda aportaron significativamente en el desarrollo de la tesis. También quisiera expresar mi inmensa gratitud para con la prof. Jocelyn Simmonds: por creer en mí y estar ahí en los momentos más difíciles que tuve durante este proceso (que no fueron pocos). Su apoyo incondicional, su visión, sus aportes en mi trabajo, por siempre mantener una actitud positiva que invitaba a esforzarme un poco más para avanzar y ahora completar esta disertación. Aprovecho también de disculparme con ustedes tres por no haber estado a la altura en algún momento del doctorado: en ningún caso fue intencional. Gracias por seguir confiando en mí, corregir y respaldar este trabajo.

Agradezco también a la comisión de esta tesis: Tom Mens, Juan Pablo Sandoval, Sergio Ochoa y Alexandre Bergel por su trabajo corrigiendo esta tesis para elevar su calidad y valor. A Alex darle las gracias por su invitación al mundo de la academia que comenzó en pregrado. A los profesores Eric Tanter y Gonzalo Navarro por su gran labor como directores de postgrado. A la profesora Cecilia Bastarrica por sus llamados de atención a "despabilar" y su apoyo en el estudio cualitativo.

A todos quienes participaron en los estudios realizados en este trabajo, tanto las entrevistas como la encuesta. Muy agradecido por su tiempo y el valor aportado para haber desarrollado con éxito este estudio que me llevó a publicar un lindo artículo de journal. También agradecer al profesor Serge Stinckwich por recibirme por dos meses en su laboratorio en los suburbios de París, Francia; y a Ilhem Bouderbala por su compañía durante mi estancia allí. Reitero mis agradecimientos al profesor Romain Robbes también por recibirme durante la pasantía en Bolzano, Italia.

A mis compañeros del doctorado por sus aportes a mi trabajo en conversaciones de pasillo. Pero principalmente por los carretes y momentos de distensión: Karen Oróstica, Hernán Sarmiento, Camilo Garrido, Jesús Concha, Elizabeth Labrada, Miguel Romero, Víctor Molina, Mauricio Quezada, Francisco Palma, Yenisei Delgado, Guillermo De La Torre, Maira Marquez, Fabián Rojas. A Miguel Campusano, mi gran compañero del doctorado, por esas conversaciones, esos cigarreins y cervezas compartidas: tantas anécdotas. A la Vane, mi gran amiga de esos años, por escucharme, alentarme, empatizar conmigo y enseñarme. Les debo mucho a cada uno de ustedes, les estoy eternamente agradecido.

El Departamento de Ciencias de la Computación fue por muchos años mi casa, y en ella hubo mucha gente que me apoyó y asistió para facilitar el proceso. Quisiera destacar el trabajo de Sandra y Angélica quienes, además de su impecable trabajo administrativo, siempre tuvieron palabras de aliento para con mi trabajo. Su profesionalismo y rigor en el trabajo es inspirador.

A mis amigos del proyecto Duckietown y actual Corporación Cuac: Matías, Christopher, Gonzalo, Luz y Rodrigo. Aprendí mucho con ustedes y fue muy gratificante sentirse parte de un equipo tan profesional, apasionado, alegre y dicharachero jajaja. Son todos ustedes unos cracks en lo que hacen.

Mis amigos también fueron muy importantes para sobrellevar este turbulento viaje que resultaron ser los “años del doctorado”. Quisiera destacar principalmente la compañía, el constante apoyo y alegría de mi amiga Ale. Soy muy afortunado de tenerte entre mis cercanos y estoy muy agradecido por cómo has sido conmigo. A mis amigos del pregrado también, con ellos he aprendido mucho y me he reído también, su compañía es fundamental. Al Felipe Engelberger por estar ahí cuando lo necesitaba, y junto con Gonzalo Mena ser una fuente de inspiración intelectual. Junto a ellos debo también incluir a Simón Arévalo, Hans Bevensee, Pablo Garrido, Edgardo Hernández y Felipe Smith por tan buenos memes. Gracias también a Chris Labarca por las juntas y los cócteles. A Andrés Monares por su (ácido) humor y las cumbres. Finalmente, a Rodrigo González (aka “El Tata”) por sus chelas conversadas, sus consejos y apoyo.

Como ya he dicho, estos años de doctorado han sido probablemente los más desafiantes de toda mi vida. Siendo la exigencia académica solamente uno de los varios retos que tuve que enfrentar durante este tiempo. Quiero agradecer el trabajo contínuo y la profesional compañia de mi psicóloga Angélica Castañeda, quien debe ser de las personas que más me conoce y quien, a través de su empatía y atenta escucha, ha sabido orientarme y revelar aspectos personales que sin duda me han transformado y me han hecho ser una persona más consciente de sí misma: me enseñó a conducirme en la vicisitudes de la vida. También agradecer el trabajo complementario de los doctores Marco Gonzalez y Daniel Sejas por su profesionalismo.

Agradecer también las horas, días, meses y probablemente años de compañía de la obra musical del prolífico cantautor chileno Cristóbal Briceño. Sus letras, ritmos y melodías calaron profundo en lo que llamo “mi segunda adolescencia” que justamente ocurrió en el transcurso de este doctorado.

Debo también agradecer a ANID (ex CONICYT) por financiar mis estudios a través de la beca CONICYT PCHA/Doctorado Nacional 2015/21150762.

Finalmente, y siguiendo el inusual pero legítimo ejercicio del rapero Snoop Dogg: quisiera agradecerme a mí. Gracias Pablo por tanto trabajo, tanto esfuerzo que incluso en circunstancias muy duras supiste llevar a cabo. Por tu resiliencia y tu perseverancia, por hacer lo mejor que se podía con lo que se tenía a mano. Gracias por estos años de transformación y crecimiento, de sublimar dolores en días de esperanza y satisfacciones postreras como son las que estoy a tan solo semanas de disfrutar.

Table of Content

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Managing robotics complexity with Robotics Middleware

Writing robotics software is hard and complex. It can be modeled as a loop of three steps: first, retrieving data from sensors (*e.g.,* cameras, lidar scanners, laser scanners, etc.), second, processing the data provided by them: to plan actions to be sent to actuators (*e.g.,* robotic arms, wheels, etc.) and third the action of the actuators. Each step and its transitions represent a complexity.

From the hardware side there are software challenges due to the sensor and actuator limitations. A sensor is always imprecise. Sensors are also affected by noise from the environment (for example sunlight or fog in optical sensors). And actuators and their mechanical and electronic components are exposed to wear and tear. Such limitations have to be handled by the software side.

The software side is in charge of orchestrating the movements of the actuators following the robot's goal using the data from sensors. To do that, the software implements probabilistic models which are, by definition, imperfect representations. This has to be taken into account in the operation of such robots [2].

Another aspect of Robotics' complexity is related to the Robotics discipline itself. It comprehends different knowledge and skills disciplines: statistics and data analysis, software engineering, mechanics, and electronics. Any robotics software projects embrace one or more of such disciplines [3]. People behind such projects commonly have a deep background in such areas and have to interact with other experts or experts' artifacts to achieve a functional robot.

Robots have a wide diversity of purposes, operational contexts, requirements, and constraints. They have a specific task to do and a goal to accomplish. A consequence of this is the wide variety of robots: humanoids, aerial robots, ground vehicles, educational, pick-and-place, etc. This usually imposes the use of custom software development which makes it more expensive in time and money.

The robotics community has proposed the use of Robotics Middleware as a solution for managing this complexity. A Robotics Middleware is an abstraction layer between the operating system and the software application. It is meant to manage the heterogeneity of the hardware, support and simplify the design and development of complex distributed robotic applications, and reduce its cost as a consequence[4]. There are many middleware for robotics, among the most popular are: Robot Operating System (ROS)[1], Yet Another Robot Platform (YARP)[2] and OROCOS[3].

However, ROS is the mainstream platform for developing robotic software. Its popularity relies on the huge amount of packages available to be reused, and the official support of more than 160 different robots. Also, a rich and active community that contributes with software and sharing knowledge, and proper channels for knowledge sharing: ROS Answers[4], a custom StackOverflow[5]-like Q&A website.

1.2 Problem Statement

> "It's common that you find a package and it's only compatible with ROS Fuerte and the only step to being compatible with upper versions is to catkinize it. But the student who made it finished his thesis and left the package as it is."
> Roboticist (+4 years of experience using ROS)

Because of its popularity, ROS has been widely used in robotics research. Several packages are usually side-products of a research project. Robotics Scientists are authors and first maintainers of them. However, maintenance of those packages, as it's not a research task, is not a strong priority for them.

We define as an *abandoned package* a software package whose maintainer is either unknown or non-responsive. Because of this, abandoned packages do not evolve along with ROS[6] and contributions from members of the community (*e.g.,* bug fixes, bug reports, feature requests, etc.) cannot be received, evaluated nor integrated.

The emergence of abandoned packages threatens the health of the ROS ecosystem in the following ways:

1. Abandoned packages make the ROS Ecosystem less reliable
2. We cannot predict which packages will be abandoned nor do we know which ones have already been abandoned
3. A missing maintainer cuts off the contribution flow
4. Breaking changes are dangerous because no one reacts to them

[1]ROS.org | Powering the world's robots, `http://www.ros.org`.
[2]YARP: Welcome to YARP, `http://www.yarp.it`.
[3]The Orocos Project – Smarter control in robotics & automation!, `https://orocos.org/`.
[4]Questions - ROS Answers: Open Source Q&A Forum, `http://answers.ros.org/`.
[5]Stack Overflow - Where Developers Learn, Share, & Build Careers, `http://www.stackoverflow.com`
[6]In ROS, such packages are called *orphaned packages*, `http://wiki.ros.org/OrphanedPackage`

The knowledge artifacts in abandoned packages, as in any software package, may be missing, incomplete or unclear. This is where knowledge-sharing platforms help in finding information to overcome installation issues, bugs, and unexpected errors, among other problems related to package reuse. In the context of ROS, robotics-related packages may make use of specific hardware, algorithms, or statistics techniques. As such, finding *the one person* who can have the answer and help users can be difficult as the problem may need specific knowledge or deep expertise in the area.

Q&A web platforms such as Stack Overflow have proven to be a solution in connecting users who need help with package reuse, with other more experienced users who may contribute to answering questions. Several recommendation systems have been implemented to find such users who can contribute to searching for an answer for a given question [5].

The problem of expert finding in Community Question Answering has been studied in depth[6] defining taxonomies of the approaches to find the best candidate to answer a given question. However, little research has been done related to finding not the best but instead qualified users that may contribute to finding an answer to a given question. This, in the spirit of distributing the workload among the community instead of concentrating it on a few very active

1.3 Hypotheses and Goals

Considering the problem statement and the study of the dynamics of the ROS community, we propose the following hypotheses:

■ Hypothesis 1: *By performing code analysis of ROS launch and configuration files, it is possible to find issues in the ROS ecosystem*

There are many ways of profiling and investigating the software ecosystem's health. We believe that by studying a software artifact we can identify issues in the practice that can lead to situations that hinder the health parameters of the ecosystem.

In particular, we propose to investigate ROS launch files that are configuration files used in ROS to start processes in robotics programs. These artifacts are very important to have a smooth deployment due to them describing the integration between components of the software that runs over ROS.

■ Hypothesis 2: *Contribution bottlenecks in ROS can be attributed to socio-technical issues within the ecosystem*

Every ecosystem is based on contributions among the community. They are individuals who spend time and effort in a contribution that fixes, enhances, or enriches a software artifact of knowledge artifact in the ecosystem.

We think that bottlenecks in contributions can be caused by not only technical by also social issues. We propose an investigation regarding the collaboration dynamics in the ROS ecosystem in order to address this hypothesis.

■ Hypothesis 3: *Expert Recommendation Systems based on the use of tags are useful for finding less participatory qualified candidates for answering questions on the Q&A site of a robotics software ecosystem*

The use of tags in Question & Answers web platforms is a common way to characterize questions and users. They give context regarding the topics in which that question is about (besides its title and body) and also what are the areas of expertise of users.

In online communities it is normal that there is a long-tail distribution in the frequency of activity of users: few users concentrate the majority of the activity in the platform [7]. When it comes to contributions and collaboration in the Q&A platform ROS Answers, few users are the most active in answering the given questions. We think that in the context of very specialized topics of the questions, a recommender system could find less participatory users that are qualified to answer a given question. This in the spirit of having a more distributed workload in the community.

1.4 Objectives

In order to be able to verify our hypotheses we propose four objectives.

1.4.1 Study clone activity in ROS Launch files

As a starting and exploratory study and based on personal experiences, we decided to work on the problem of code duplication in ROS artifacts. The study is focused on a particular artifact: ROS Launch files, a configuration XML file used to launch several robotics processes that work together for achieving robot behaviors. These files are very important because the connections between processes and their proper configuration are needed to make a robot behave in a certain way.

Our first sub-objective is to know about the diversity of file types in ROS packages. The second one is to know about the existence and frequency of code duplication instances in ROS launch files. Finally, the third sub-objective is to know the particularities of these duplicated code fragments, to know which tags are actually duplicated, and their frequency.

1.4.2 Characterization of the collaboration dynamics in the ROS community

Once we had a first approach to the developer experience in ROS we wanted to have a better understanding of how community members use ROS and how are the collaboration dynamics in this community.

To do that we proposed a field study that involves a set of interviews with ROS users to know about their sense of discomfort in using ROS. The questionnaire (Annex A) covered a wide range of aspects of their development experience. Once we analyzed the transcripts of the interviews we applied an open survey (Annex B) over the community to confirm or disproves the interviews' findings.

1.4.3 Identify the more important bottlenecks in contributions and provide recommendations to address them

The analysis of both the interviews and the survey gives us hints regarding how ROS users collaborate and contribute to the ROS ecosystem: their experience reusing packages, how they look for help and what are the contribution dynamics.

Our first objective is to distill the most frequent contribution bottlenecks and describe them. The second objective is to provide recommendations from the literature to prevent them or overcome their negative impacts on the ROS ecosystem's health.

1.4.4 Recommend qualified members for knowledge contributions

One important aspect of the community contribution dynamics is the search for help. The ROS software ecosystem relies on platforms of Question & Answers to aid users and developers looking for help in reusing packages, configuring their launch files, and troubleshooting among other needs in building robotics applications. In these communities, sometimes very few users are the most active and are the ones that commonly provide answers. We want to look for a method that helps the community to find qualified users to answer a given question. This, in the purpose of having a better workload distribution.

1.5 Contributions

This dissertation provides four main contributions:

1. **Methodology and Evidence of Code Duplication in ROS Launch files**: As a first study of the ROS ecosystem we performed a code duplication analysis of a specific software artifact in ROS: launch files. These configuration files are responsible of orchestrate the launch of different processes that a robot's behavior is based on. We found that a non-negligible portion of ROS packages has duplication in these files. In this study, we characterize the packages that have code clones and reveal the impact of their presence in their development.
2. **Evidence on Socio-Technical aspects of the ROS Ecosystem**: We studied the ROS ecosystem from a socio-technical approach obtaining interesting insights into the experience of ROS users, collaboration issues, issues on Package reuse, support dynamics, and user needs and expectations.
3. **Recommendations for addressing contribution bottlenecks**: We proposed 5 different recommendations for overcoming the effects of the identified contribution bottlenecks. They are based on the existing literature and address topics such as package abandonment identification, providing an informative package repository, recommending contribution opportunities to qualified community members, limiting breaking changes and motivating and encouraging community contributions.
4. **Novel approach for actively distributing knowledge sharing the workload in the ROS Ecosystem**: We created the Tag Map Based Algorithm, a recommender system that aims to find qualified members in Q&A platforms that have less participation than the most active users. The recommender system was applied to the ROS

Ecosystem and succeeds in recommending qualified users but does not recommend less active users.

1.6 Associated Publications

This work has produced the following publications:

- **Estefó, P.**, Robbes, R., & Fabry, J. (2015, November). *Code duplication in ROS launchfiles.* In 2015 34th International Conference of the Chilean Computer Science Society (SCCC) (pp. 1-6). IEEE.[7]
- **Estefo, P.**, Simmonds, J., Robbes, R., & Fabry, J. (2019). *The robot operating system: Package reuse and community dynamics.* Journal of Systems and Software, 151, 226-242.[8]

1.7 Outline of the thesis

The dissertation is structured as follows:

- *Chapter 2: Robotics Middleware and The Robot Operating System*, introduce the issues in robotics programming and presents the concept of robotics middlewares and shows examples of them.
- *Chapter 3: Code Duplication in ROS Launch Files*, presents a study on the code duplication of these ROS artifacts characterizing them in depth.
- *Chapter 4: Field Study: Interviews and Survey*, in which we report the results of the study about ROS community dynamics and use of ROS.
- *Chapter 5: Contribution Bottlenecks and Recommendations*, as an outcome of the previous chapter it presents a set of bottlenecks in contribution and recommendations to overcome their negative impacts.
- *Chapter 6: Distributing Community Efforts in Knowledge Sharing Platforms: the case of ROS Answers*, shows the experience of building an expert recommendation system for identifying qualified members of the community to answer a given question in Q&A platforms.
- *Chapter 7: Final Remarks and Future Work*, summarizes the contributions, discusses the objectives and the validation of the hypotheses, and projects the future work that can be done from this dissertation.

[7] 8 citations up to March 24th 2023
[8] 57 citations up to March 24th 2023

Chapter 2

Robotics Middleware and The Robot Operating System

In this chapter, we present several Robotics Middleware as they arguably are the most complex software engineering products that are used to face the challenges in Robotics. We will present different middleware and give more detail about those that are most widely used. We will then evaluate the use cases of these from the software engineering perspective.

2.1 Challenge of Robotics

Robotics is the discipline involved in the design, construction, and operation of robots and its applications. Robotic systems are designed and built to pursue a specific goal, which is achieved by the interaction and coordination of one or more robots and external systems with the physical environment. Robotics relies on Electrical Engineering, Electronics Engineering, Mechanical Engineering, and Computer Science [4]. The Computer Science perspective on this field is called Robotics Programming, and is in charge of the software development of the robotic system [8].

On the one hand, from the hardware perspective, a robot or a robotic device is composed of a combination of sensors, actuators, and computers. Sensors are devices that obtain data from the physical world. Actuators are devices that interact physically with the physical world. Computers process data from sensors, control the actions of actuators, and communicate with other computers in external systems.

On the other hand, from a software perspective, robot programming aims to develop the optimal robot behavior that accomplishes the given goal. This robot behavior is determined by the data flow from the sensors that is processed by the computer, generating a reaction on the actuators. This process is typically bounded by a strong real-time response [9], but the severity of this requirement varies from context to context of the application.

Robotic systems perform in different contexts such as industrial or home automation, health care, military, emergencies, or tasks in human inaccessible areas [4]. An example

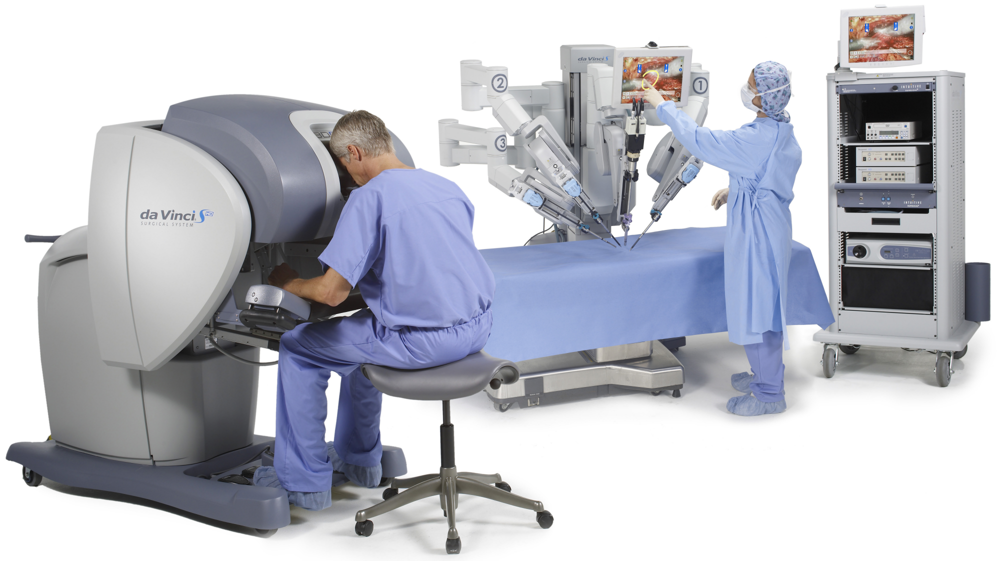


Figure 2.1: Da Vinci Surgical System (*Image Copyright © 2015 Intuitive Surgical, Inc.*)

of the latter application is robotic-assisted surgery, *e.g.,* the Da Vinci System[1] shown in Figure 2.1). This is an application that requires high precision of the different actuators of a robot surgeon and a strict real-time response of the underlying system.

All the previously listed contexts of application impose particular constraints on the design of the robotic systems. For example precision: the precision of the motors of one arm of the Da Vinci System is much higher than the precision of an arm of a robot that moves pallets. There is a large heterogeneity of hardware that causes challenges in three aspects:

1. Choosing the most appropriate hardware is not easy due to the variety of hardware available. For example, there are many types of laser scans (two or three dimensions) and different sensitivity ranges, among other characteristics, each of these useful for different contexts.
2. There is a need for a deep understanding of the developer of the chosen hardware component to be able to use it in an optimal way.
3. Once the developer knows how to use the component, she or he needs to know which algorithms work for this piece of hardware and for the task to perform (*e.g.,* navigation or object recognition).

The final important challenge when programming robot systems is that the robot environment is intrinsically unpredictable [10]. This uncertainty is present in each part of what composes a robot: *sensors*, *actuators*, and *software*. Sensors are limited in what they can perceive: there are physical limitations (*e.g.,* resolution), they are subject to noise and they can get damaged or broken. Actuators are subject to control noise and wear-and-tear, and their parts are susceptible to mechanical damage. Finally, software uncertainty is based on the uncertainty of models of the physical world. These models are, by definition, mistaken

[1]Da Vinci Surgery, `www.davincisurgery.com`

versions of the physical world. Software models are also fed with data that comes from algorithms that give approximate results, either due to real-time constraints or the intrinsic complexity of the problem.

Summing up everything we mentioned before, robot programming is different from computer programming due to the fact that robots are not only computers and this complexity make them more intricate and challenging to program.

2.2 What is a Robotics Middleware?

The Middleware concept is defined by Bakken *et al.* [11] as follows:

> "*A class of software technologies designed to help manage the complexity and heterogeneity inherent in distributed systems. It is defined as a layer of software above the operating system but below the application program that provides a common programming abstraction across a distributed system.*"

A Robot middleware is hence an abstraction layer between the operating system and the software applications, meant to manage the heterogeneity of the hardware, to support and simplify the design and development of complex distributed robotic applications, reducing its cost as a consequence [4].

According to the robotics middleware survey of Mohamed *et al.* [12], the requirements that a middleware for robotics should fulfill are:

1. Simplifying the development process by providing high-level abstractions with simplified interfaces.
2. Supporting communications and interoperability between robot modules.
3. Providing efficient utilization of available resources to ensure real-time interaction between robot components.
4. Providing heterogeneity abstractions for software and hardware components hiding their low-level complexity.
5. Supporting integration with other systems.
6. Offering often-needed robot services such as algorithms and basic robotic libraries.
7. Providing automatic resource discovery (*e.g.,* external devices) and configuration.
8. Supporting embedded components and low-resource-devices.

Reasons for adopting the Middleware design in Robotics are given by Ceriani *et al.* [1]:

Modularization: The middleware also defines the structure of its modules and its interfaces for communication between modules as well as with hardware resources and other external systems. This component paradigm structure eases the maintainability and testability of each component, lowering the complexity of the whole application.

Portability: The middleware enables portability for robotics applications as it is placed over the operating system (OS). Although the limit between the middleware and the OS is arbitrary. Applications built using it will use low-level resources through a strictly defined common programming model.

Reliability and Testability: This layered architecture provides reliability and testability, due to the middleware being tested separately and that libraries and applications developed on top can also be tested in an isolated manner.

Besides the requirements that every robotics middleware satisfies, the design of each middleware defines the design of its modules and the applications that use it. Similar robotic behaviors may have completely different implementations due to the use of a different middleware.

A considerable quantity of robotics middleware has been proposed. We list here all the robotics middleware found in the middleware robotic surveys we are aware of [12, 1, 4, 13]:

1. Robot Operating System (*ROS*) [14] defines itself as a "thin, message-based, peer-to-peer" robotics middleware,
2. *Miro* [15] an object-oriented robotics middleware developed at the University of Ulm,
3. Microsoft Robotics Developer Studio (MRDS) [16] is an environment for the simulation and control of robots in Windows,
4. *Marie* [17] (Mobile and Autonomous Robotics Integration Environment) was developed by the Mobile Robotics and Intelligent Systems Laboratory of the University of Sherbrooke.
5. Coupled Layer Architecture for Robotic Autonomy (*CLARAty*)[18] developed by the NASA for research on extra-planetary exploration
6. *Orca* [19] is a component-based framework designed by the IT R&D program of the Ministry of Knowledge Economy of Korea
7. Yet Another Robot Platform (*YARP*) [20] is defined as a "set of libraries, protocols, and tools to keep modules and devices cleanly decoupled" and it is developed as a collaboration between the University of Geneva and the MIT
8. *ERSP* Software Development Kit [21] provides software for vision, navigation and system development
9. *OpenRTM* [22] is a software platform that implements the RT middleware standard [23] and is developed by the National Institute of Advanced Industrial Science and Technology-Intelligent Systems Research Institute
10. *Webots* [24] allowing programming mobile robots through a package for robot simulation (commercial) *Player* [25], developed at the University of Southern California, defined as a "distributed device repository server" for robots, sensors, and actuators
11. Open Robot Control Software (*OROCOS*) [26] is a real-time toolkit for developing robotics applications with C++.
12. *UPnP middleware* [27] was developed by Korea Institute of Science and Technology to utilize the Universal Plug and Play (UPnP) architecture for robot software integration and ubiquitous robot control.
13. *ASEBA* [28] is an event-based middleware initially developed by the École Polytechnique Fédérale de Lausanne (EPFL), that supports distributed control and efficient

resource utilization of multiprocessor robots.

14. The *PEIS Kernel* [29] is a middleware for ubiquitous robots and it is based on a collaborative research project between the Electronics and Telecommunications Research Institute (ETRI), Korea, and The Centre for Applied Autonomous Sensor Systems, Sweden.
15. The *RSCA* (Robot Software Communication Architecture) [30] is a QoS (Quality of Service)-Aware middleware for networked service robots developed by Seoul National University.
16. The *AWARE* platform [31] is a data-centric middleware for the integration of wireless sensor networks and mobile robots developed by the University of Seville, Spain, and the University of Stuttgart, Germany.
17. *Sensory Data Processing Middleware* [32] provides abstracted services for accessing sensor information to support service mobile robots. It was developed at The University of Tsukuba in Japan.
18. *RoboFrame* [33] is an object-oriented middleware written in C++ developed for lightweight humanoid and wheeled robots.

2.3 Popular Middleware

After a revision of the projects in the previous list, the activity of their contributors, and their most popular uses (number of contributors, the impact of projects that uses them, etc.), we consider that *OROCOS*, *Orca*, *MIRO*, *YARP* and *ROS* are the most popular. We will now present the first four, and the fifth will be developed in detail in Section 2.4.

2.3.1 OROCOS

The OROCOS [26] project started in 2000 as one of the first Open Source (LPGL) general-purpose robot control software projects. It was designed to be extremely modular and flexible and developed to run on any robotic device or computer platform. It was made to be independent of commercial robot manufacturers but open to contributions from third parties of the industry as long as they follow the open license. It provides software for kinematics, dynamics, planning, sensing, and control. Its components can be added or removed from a network and it offers its services through a language natural interface.

OROCOS defines four main C++ libraries as shown in Figure 2.2:

1. **OROCOS Real-Time Toolkit (RTT)** [34] provides an infrastructure and functionalities to build robotics applications in C++. RTT allows components to execute on real-time operating systems and offers real-time scripting, an API for component communication, and XML configuration. An RTT component can be written to control actuators or whole robots, or to capture and plot data flows from a sensor. They can be configured online through set/get values and also offline through XML files.
2. **OROCOS Components Library** provides pre-built RTT components to be used out of the box.
3. **OROCOS Kinematics and Dynamics Library (KDL)** is a library written in C++ which allows calculating kinematic chains: the mathematical model that describes the

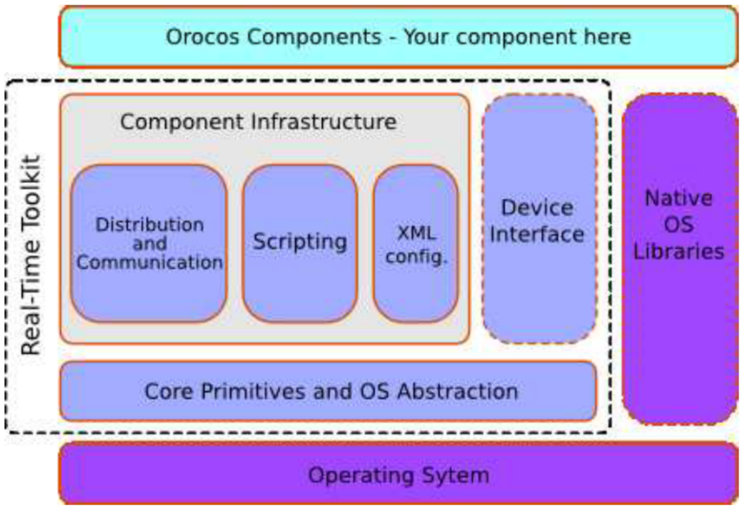


Figure 2.2: OROCOS Real Time Toolkit architecture [1]

mechanical system, in real-time. It provides a framework for modeling and computation of kinematic chains, *e.g.,* biomechanical human modes.

4. **OROCOS Bayesian Filtering Library (BFL)** [35] provides a framework for inference in Dynamic Bayesian Networks, *i.e.,* "recursive information processing and estimation algorithms based on Bayes' rule, such as (Extended) Kalman Filters [36], Particle Filters [37] (or Sequential Monte Carlo methods)" [35]. These algorithms are used in real-time services or used in applications for kinematics or dynamics estimations.

The Autonomous Vehicle, *MadeInGermany* [38] developed by the AutoNOMOS Labs [2], is an example of a complex application of a robotic behavior built on top of the *OROCOS* middleware.

2.3.2 Orca

Orca [19] is an Open Source framework for developing component-based robotic systems. It is a fork of the OROCOS project that took place in 2005 [39]. Its main goal is to encourage software reuse in robotics by defining and developing the building blocks that can be assembled to construct robot systems. This diversity of building blocks should lead to a wide variety of systems, from single vehicles to distributed sensor networks. Orca achieves this goal by defining a set of interfaces, providing a high-level API for simplifying the use of commonly-used libraries and maintaining a repository of components facilitating the reuse of these. Orca also provides a battery of tools for easing component development. However, they are optional when full access to the underlying communication engine and services is required. The main difference with its parent project, OROCOS, is the replacement of its communication engine. Orca replaced CORBA (Common Object Request Broker Architecture) [40] with ICE (Internet Communication Engine) [41] a more modern and easier-to-use framework.

Orca's infrastructure is based on the *IceGrid* and *IceStorm* services and a set of common

[2]AutoNOMOS Labs, `http://autonomos-labs.com/`

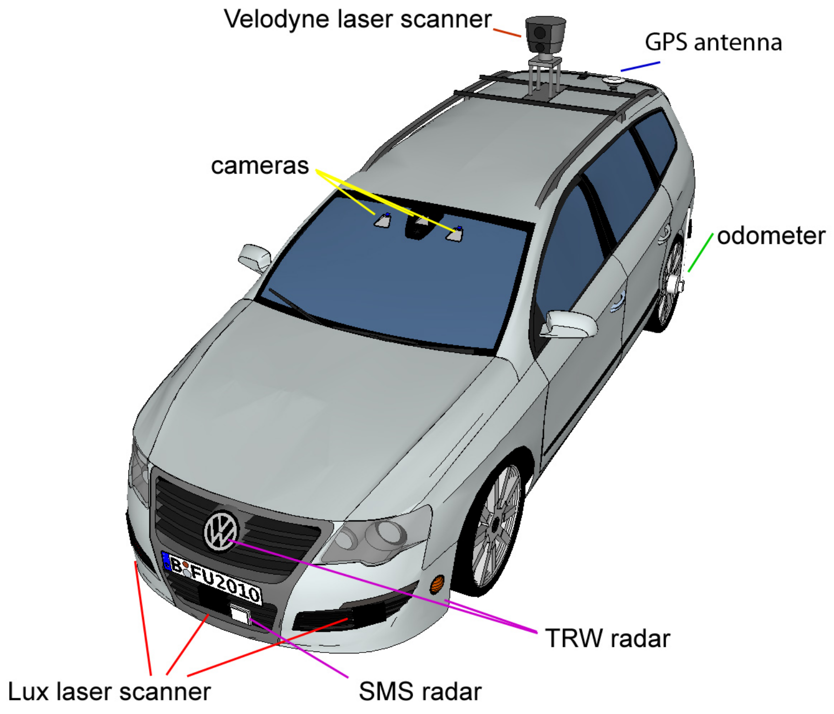


Figure 2.3: The *MadeInGermany* autonomous vehicle (*Image Copyright © 2015 AutoNOMOS Labs)*

components that can be extended or reused. *IceGrid* is a registry that provides a naming service for components to find another one to interact with. *IceStorm* is an event service that decouples message traffic between publisher and subscriber components. An Orca application is then built by assembling components, which know each other through *IceGrid* and communicate through *IceStorm*.

2.3.3 MIRO

MIRO [15] is an object-oriented middleware for robots developed by the University of Ulm. Its main goal is to improve the software development process for mobile robots and enable the interaction between robots and enterprise information systems. It is implemented using the CORBA standard [40]. This standard enables inter-process and cross-platform interoperability for distributed robot control. The structure of MIRO can be divided in three layers: the *device* layer provides object-oriented abstractions for sensors and actuators; the *service* layer provides abstractions for devices via CORBA; and the *class framework* provides common-used services such as self-localization, mapping, and path-planning, amongst others. The layered architecture and the use of an object-oriented approach aim to make it a flexible and extensible middleware for supporting new devices and services.

2.3.4 YARP

YARP (Yet Another Robot Platform) [20] is an open-source robotics middleware designed especially for humanoid robots. Its first release was in 2002 as a joint effort between the University of Geneva and the MIT. It is composed of a set of libraries for decoupled devices and processing communications. YARP supports incremental architecture evolution by providing loose coupling between sensors, actuators, and processors. Communications in YARP follow the Observer Pattern [42] using objects called *Ports*. An object *Port* delivers data to any number of observers. Other *ports* are associated with different processes, which might be distributed across different computers. A *port* allows a user code to receive inputs or send its outputs from and to different processes using the appropriate protocol for the communication context, *e.g.,* TCP, UDP, or multicast. Ports can then be programmatically connected at runtime. Connections are asynchronous and the default behavior is to maintain stable streams of data, instead of ensuring that there is no loss. This is due to the type of applications that need a constant flow of sensor data (*e.g.,* image processing) in the context of several processes consuming this data at different rates. YARP also abstracts the details of the devices from the programs using them by defining common interfaces for device families. The most popular robot running YARP is the iCub robot platform [43]. It is a complex 53 degree-of-freedom humanoid robot, and its development is part of a cooperative project funded by a program of the European Union.

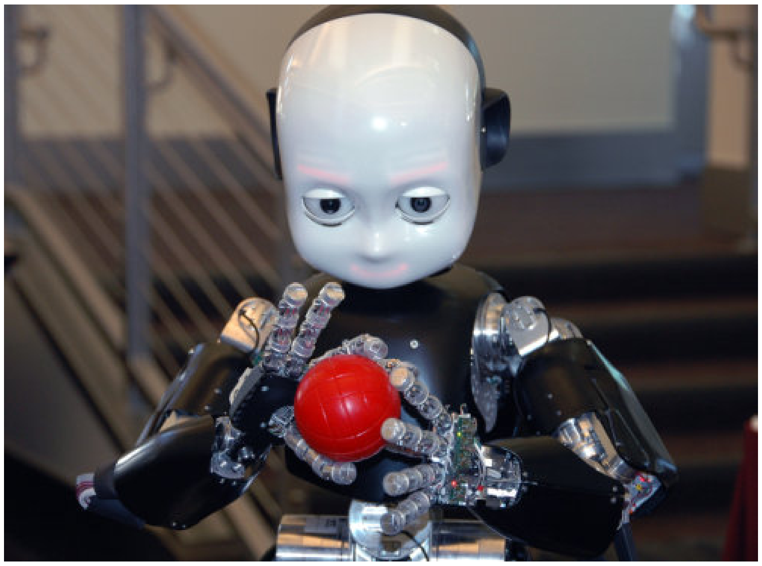


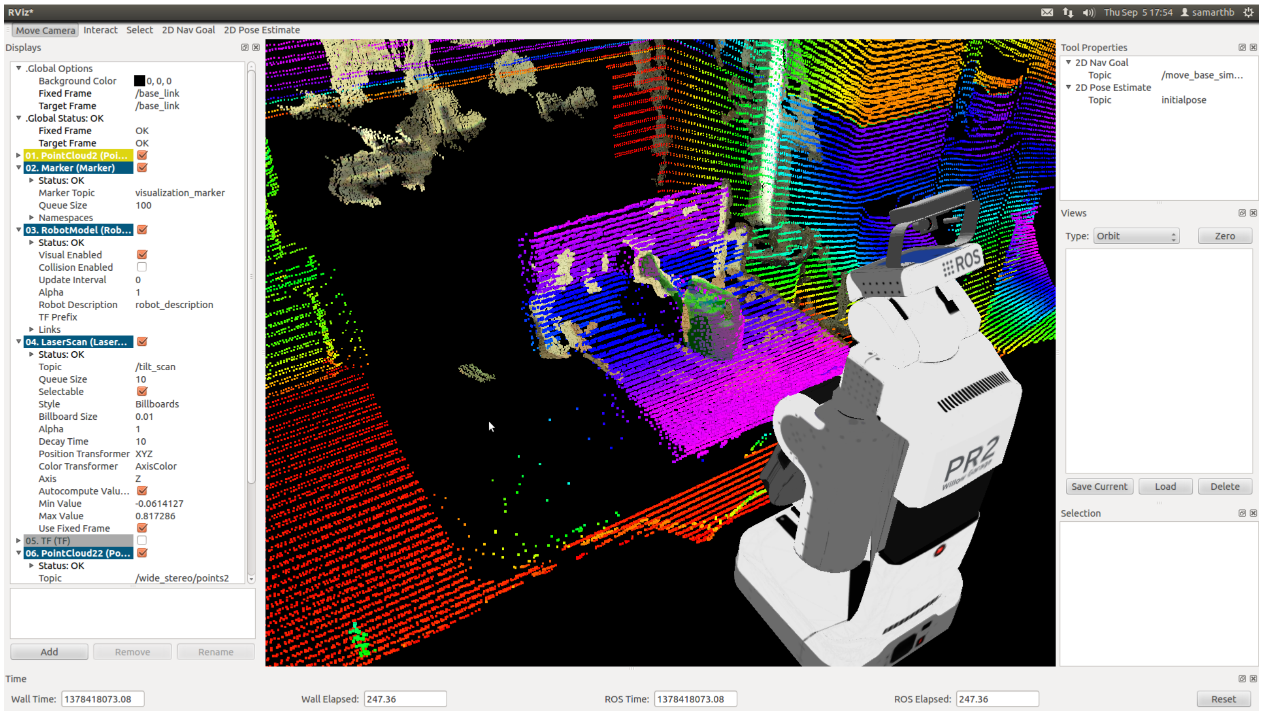
Figure 2.4: iCub robot platform. (*Image Copyright © 2015 David Vernon)*

2.4 Robot Operating System (ROS)

ROS is an Open-Source middleware presented in 2006 by Quigley *et al.* [14] from Stanford University and Clearpath Robotics (formerly Willow Garage) [3]. ROS is more than a middleware as it also provides for hardware abstraction, commonly-used functionalities and package management for easy code reuse.

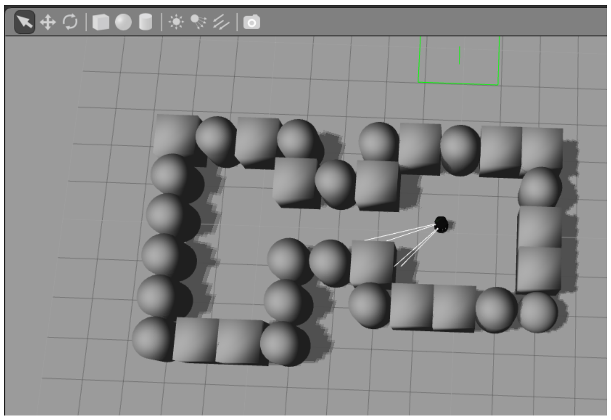
A ROS application is modeled following the Publish-Subscribe architecture. A non-trivial ROS application is thus represented as a computational graph. This graph is composed of the following elements:

[3]Clearpath Robotics, `https://clearpathrobotics.com/`

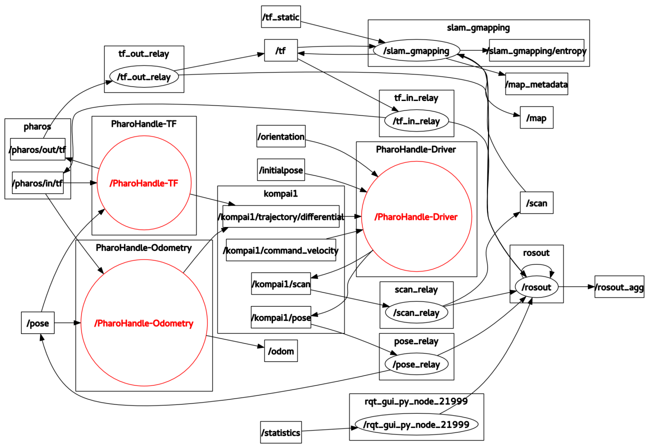


```
▼ .Global Options
    Background Color        ■ 0, 0, 0
    Fixed Frame             /base_link
    Target Frame            /base_link
▼ .Global Status: OK
    Fixed Frame             OK
    Target Frame            OK
▶ 01. PointCloud2 (Poi...   ✓
▼ 02. Marker (Marker)       ✓
  ▶ Status: OK
    Marker Topic            visualization_marker
    Queue Size              100
  ▶ Namespaces
▼ 03. RobotModel (Rob...    ✓
  ▶ Status: OK
    Visual Enabled          ✓
    Collision Enabled       ☐
    Update Interval         0
    Alpha                   1
    Robot Description       robot_description
    TF Prefix
  ▶ Links
▼ 04. LaserScan (Laser...   ✓
  ▶ Status: OK
    Topic                   /tilt_scan
    Queue Size              10
    Selectable              ✓
    Style                   Billboards
    Billboard Size          0.01
    Alpha                   1
    Decay Time              10
    Position Transformer    XYZ
    Color Transformer       AxisColor
    Axis                    Z
    Autocompute Valu...     ✓
    Min Value               -0.0614127
    Max Value               0.817286
    Use Fixed Frame         ✓
▶ 05. TF (TF)               ☐
▼ 06. PointCloud22 (Po...   ✓
  ▶ Status: OK
    Topic                   /wide_stereo/points2

▼ 2D Nav Goal
    Topic                   /move_base_sim...
▼ 2D Pose Estimate
    Topic                   initialpose

Type: Orbit

Wall Time: 1378418073.08    Wall Elapsed: 247.36    ROS Time: 1378418073.08    ROS Elapsed: 247.36
```

(a) Visualizing data from a PR2 Robot with *RViz* *(Screenshot by Samarth Brahmbhatt)*



(b) Simulating a Turtlebot Robot Behaviour with *Gazebo* *(Screenshot taken from ROS Answers)*



(c) Visualizing a Navigation behaviour with *rqt-graph*

Figure 2.5: ROS Tools

Nodes are processes that perform a specific computation (*e.g.,* read a laser, calculate a map, control certain motors, etc.)

Topics are communication channels to which a node can publish on or subscribe to.

Messages are data structures sent from a node through a topic to any node subscribed to it. They are described in a language-neutral interface definition language.

Parameter Server centralizes parameter definitions, so they are accessible from any node in the network.

Master service provides name registration and lookup. Nodes communicate with the Master service to be aware of both active nodes and their URLs as well as currently available topics.

Services are used when synchronous interactions are needed between nodes. This follows the client/server model and each server defines a pair of typed messages for communication: one for the request and the other for the response.

This architecture favors a decoupled operation using names as a means of assembling nodes into a complete system. It allows, for example, any proximity sensor to publish its readings as a scan without knowing the nodes that are subscribed to its publishing topic. The sensor can even be interchanged at run-time and all the subscribed nodes do not perceive major differences, as long as the node of the new sensor keeps publishing on the same topic.

ROS speeds up programming robotic behavior by offering a rich set of libraries with commonly-used robotic algorithms and ready-to-use components (*e.g.,* the Point Cloud Library [44] and OpenCV [45]). In addition, ROS provides tools (see Figure 2.5) for assisting development such as: *RViz*[4] for 3D Visualization, *Gazebo*[5]: a 3D simulator and the *RQt*[6] tools family for debugging.

On their website, the ROS project lists 140 robots that can be programmed using this middleware, classified by its kind:

- **Mobile robots** are automatic machines that are capable of translating themselves. There are 56 mobile robots registered, where the classic example is the *Turtlebot* [46] shown in Figure 2.6. Its first version is composed of a mobile base with a connected gyroscope, a Kinect as a scan sensor, and a laptop running ROS. It is mostly used for educational purposes.
- **Manipulators** are robots that allow users to manipulate materials or objects, thereby avoiding direct contact by a human. The classic example is a robotic arm. There are 20 manipulators registered.
- **Mobile manipulators** are manipulators on a mobile platform. One of the biggest projects of the 25 mobile manipulators registered and an iconic ROS project is the *PR2* [47] (see Figure 2.7 and Figure 2.5a). It is a platform for "researchers and devel-

[4]RViz, `http://wiki.ros.org/rviz`
[5]Gazebo Simulator, `http://gazebosim.org/`
[6]RQt Common Tools, `http://wiki.ros.org/rqt_common_plugins`

Figure 2.6: Turtlebot V2 (*Image Copyright © 2015 Open Source Robotics Foundation, Inc.)*

opers to more quickly advance the state of the art in robotics" [48]. It is composed of a mobile base, two articulated arms, and a head. The mobile base has a laser rangefinder, and the head has a tilting laser rangefinder. Each arm has a gripper and a camera in its forearm. In its head, it has three cameras, a LED texture projector, and a Microsoft Kinect. Each component of the PR2 is controlled by a ROS package.

- **Autonomous cars** are regular cars that are autonomously driven. Three projects are registered: Carla [7], ADAS [8] and RBCar [9].
- **Social Robots** are autonomous robots that interact and communicate with humans or other autonomous physical agents by following social behaviors and rules attached to their role. We found 4 robots under this category, as an example, there is the QTRobot [10] project.
- **Humanoid** is a robot whose shape is built to resemble the human body. There are 9 humanoid robots registered.
- **Unmanned Aerial Vehicle (UAV)**, popularly known as *drone*, is an aircraft without a pilot aboard. There are 10 UAVs registered.
- **Autonomous Underwater Vehicle (AUV)** is a robot that translates itself underwater without the necessity of input from an operator. There 3 UAVs registered.
- **Unmanned Surface Vessel** are vehicles that operate on the surface of the water without a crew. There are 3 USVs registered.
- **Others**: There are 11 robots in this category, including the Lego EV3 and an autonomous skid-steerer, amongst others.

[7]Carla Robot,`https://robots.ros.org/carla/`

[8]ADAS Development Vehicle Kit,`https://robots.ros.org/adas-development-vehicle-kit/`

[9]RBCar,`https://robots.ros.org/rbcar/`

[10]QTRobot,`https://robots.ros.org/qtrobot/`

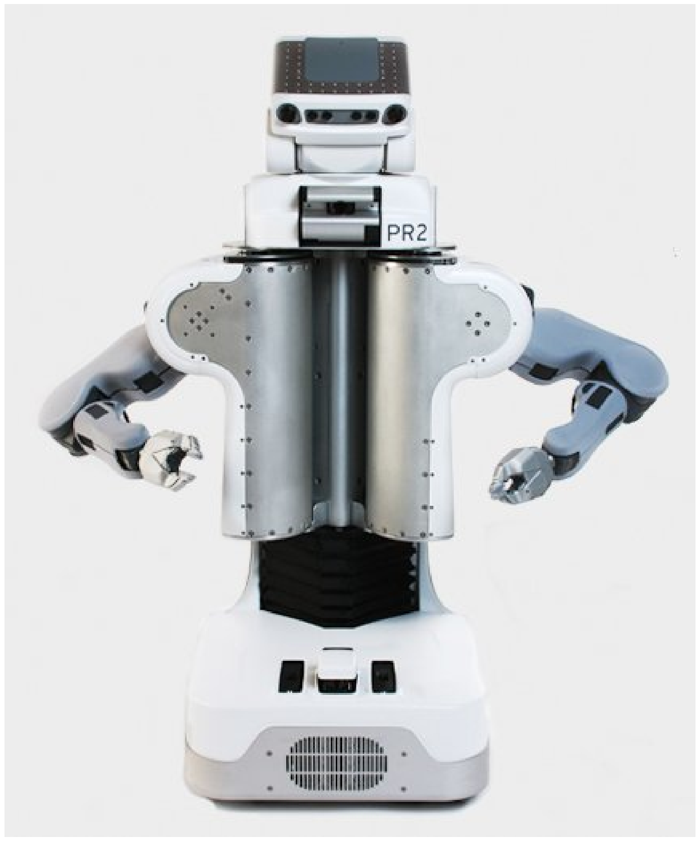


Figure 2.7: PR2: “Personal Robot 2” platform (*Image Copyright © 2015 Willow Garage, Inc.)*

The amount and variety of robots are a strong indication that ROS has standout advantages for robot programming. In fact, a case study related to In-Hand Manipulation was reported by Walck *et al.* [49]. They present the lessons learned in the integration of a software architecture based on ROS within their HANDLE project. With the HANDLE [50], the authors aim to generate knowledge on how humans manipulate objects with their hands in order to replicate it for the development of grasping and other in-hand movements with robotic anthropomorphic hands. Technically, they used ROS on a robotic system composed of a Shadow [11] dexterous robotic hand with 5 fingers (24 degrees of freedom (DoF)) and the Shadow 4 DoF biomorphic arm, coupled altogether with several external sensors for vision and tactile systems. Their experience confirmed several features of this middleware:

- **Multi-robot support**: ROS is able to simultaneously handle several robots that can be linked together in a single robot description.
- **Large Libraries Availability**: ROS users can use a wide variety of packages and libraries to accelerate the development of their robotic behaviors. Walck *et al.* reused perception libraries, a household objects database, the control, and real-time loop, and the pick and place and arm navigation component. The arm navigation component was complemented with the Orocos KDL (previously defined in Section 2.3.1). This is an interesting case of re-usability and integration of ROS with components from other middleware.

[11]Shadow Robot Company,`http://www.shadowrobot.com/`

Additionally, the HANDLE project involved the combination and integration of over a hundred components and packages of the ROS middleware [49, 51]. Although they report having had issues using topics (scalability) and services (node blockage), they conclude to have had success using the re-usability mechanisms in terms of flexibility, reconfigurability, and interoperability.

ROS as a Software Ecosystem:

A widespread definition of Software Ecosystems as stated by Lungu *et al.* [52] is:

> "*A software ecosystem is a collection of software projects which are developed and evolve together in the same environment.*"

Considering the evaluation presented so far, we can make the following observations:

1. ROS applications are composed of interactions of several components grouped in the form of packages
2. ROS packages usually make use of other packages, creating dependencies between them.
3. All these packages are subject to updates of the ROS middleware.
4. ROS packages hence evolve and coexist around ROS.

The aforementioned characteristics of ROS clearly fits Lungu's definition of a Software Ecosystem, and hence we can say that ROS is a Software Ecosystem.

The community in ROS has three main platforms for communication: ROS Answers[12] and ROS Discourse[13] and ROSCon[14].ROS Answers is a Question & Answers platform similar to the well-known StackOverflow platform, it is comprised of more than 103K users and 67K questions (58% of them are marked as "answered")[15]. ROS Discourse is an online forum that hosts discussions about the present state and the evolution of ROS, its core, its packages, and the community. It has more than 8.3K users and 63.2K posts. ROSCon is a conference held in different countries[16] every year, in the year 2022 it had 801 attendees from 38 countries. Unlike the two first communication channels that are virtual, ROSCon is face-to-face.

2.5 Open Issues and Final Remarks on Robotics Middleware

Although Robotics Middleware has actually accelerated robotic projects by re-using shared expertise and encouraging collaboration, several issues remain unaddressed while other new problems have arisen.

Among the new issues for robot programming, there are the *barriers to entry* for utilization of robotics middleware in complex robotic projects. These barriers refer to the time, effort,

[12]ROS Answers, `http://answers.ros.org`
[13]ROS Discourse, `https://discourse.ros.org`
[14]ROSCon, `https://roscon.ros.org`
[15]All the numbers given in this paragraph were observed by October 24th of 2022
[16]Except in the 2020 and 2021 editions in which due to the COVID pandemic the conference was virtual

and knowledge that a new user needs for achieving the desired robotic behavior. There are several barriers to entry, for example, customizing configuration files, choosing the appropriate parameter that optimizes a certain algorithm, creating a virtual robot model geometry, etc. More in detail: non-trivial robotics applications rely on software components from several domains of robotics, which most of the time imply various degrees of customization and optimization. Choosing the right parameters that best fit the task at hand then involves expert human input that exceeds the capacity of one sole person.

This situation represents an emerging challenge for Robot Agnostic Middleware [53], due to the loss of the necessary critical mass of collaborators for healthy development. Such large, complex, and research-borderline meta-projects rely on the sum of the efforts of hundreds or thousands of developers and researchers. Reducing the barriers to entry favors collaborations and enhancements on the available software resources provided by the middleware.

Favoring the accessibility of robotic development will speed up its innovation, in the end.

The success of using middleware for the support of robotic projects is revealed by the emergence of many middleware as we presented in Section 2.2. Developers and researchers then need to compare different middleware and decide which one fits better their projects' needs. Middleware have been traditionally compared from the perspective of their users, looking at features and performance, which is adequate for projects that operate in short time scales [54]. However, robotics middleware typically grows in size, features, and technologies involved, in essence growing together with robotics research. These hence need to be analyzed from the software evolution perspective as well. For instance, upgrades in the framework (or middleware) have no effect for developers in the short term, but they do in the long term [55]. This is due to the time needed to invest in development not directly related to the robotic application but in the use of the bindings to the middleware.

As previously said, the architecture of the middleware deeply impacts the way that any application using it is built. Moreover, Ceriani *et al.* [1] clearly point out in their survey:

> "*Middleware is not only a set of libraries and APIs for specific applications. It represents a way of thinking about the software, which will be designed and developed on a specific middleware, that defines the foundation for a complete application* "

The concept of Robotics Middleware has proven to be a successful model for accelerating the development of complex robotic projects due to its hardware abstractions, communications facilities, and ready-to-use components, libraries, and dedicated tools. This success has opened new challenges for robotics middleware developers, such as the *barriers to entry* problems, which slow down the innovation and improvements in this area. The long-term vision in the development of a robotics middleware is fundamental for its future as a reliable alternative for robotic projects.

Chapter 3

Code Duplication in ROS Launch Files

In this chapter, we deal with the problem of intra-package code duplication in a particular software artifact of the Robot Operating System: the *launch files*. These files are in charge of launching several processes and communication between them to achieve certain robot behavior. This chapter is structured as follows: in Section 3.1 we present the motivation of this study, later we present the technical aspects of ROS, and introduce the Launch Files (Sections 3.3, 3.4). The methodology of this study is presented in Section 3.5. In the following sections 3.6, 3.7, 3.8, 3.9, 3.10 the five research questions are discussed and answered. Finally, we recapitulate and compare some existing work in Section 3.2, and present our conclusions in Section 3.12. This chapter is mainly based on the work of the article *Code duplication in ROS launch files* [56].

3.1 Motivation

Our experience developing robotic applications using ROS has been, in some measure, a bumpy road. We have faced many difficulties and cumbersome situations that make us think about the quality of ROS as software.

At the time of this investigation, there were few studies of the ROS code, even with the big impact that this problem can impact on robotics R&D worldwide A low code quality of ROS can favor error propagation, hamper evolution and increase maintenance effort which outcome in a waste of time and effort among the community affecting the ecosystem's health.

An example of bad code quality is the discovery from a personal experience: Using a PR2 robot [47] (see Figure 2.7) we failed to launch the teleoperation program. We reviewed the different launch files of the package `pr2_teleop_general` [1], responsible for teleoperation of the PR2.

Inspecting the launch files we arrived with the case of two of them were 28 lines long files and only had two lines that were different. After a further review, we discovered that this

[1]`pr2_teleop_general` package - ROS Wiki, `http://wiki.ros.org/pr2_teleop_general`

was not an isolated case. We found several other cases of code duplication in the 7 launch files, with on average almost 13 duplicate lines.

In addition to this particular experience, the report of the BMW company [57] talks about problems managing their launch files for different vehicles involved in automated driving projects. These two experiences motivate a study of code duplication, as it is a popular criterion for assessing software quality.

It is very common that developers copy and then paste a piece of source code for software reuse [58]. The copied snippet can be edited afterward or not. In any case, this portion of the source code is called a *clone* [59]. The practice of using code clones is based on several situations such as lack of a mechanism in the programming language to abstract parts of code [60], a programmer's lack of understanding of the system [61] and time limits [62]. One of the main consequences there is that this practice can be enabling bug propagation [63] or design flaws [64], which increases maintenance costs and impacts negatively on evolution. Detecting code clones is a recommended first step for reducing these potential negative repercussions [65, 58]. And not only in source code artifacts but also in configuration files (such as launch files) that according to García *et al.* [66] they are often reused.

In the following, we present the investigation of the presence of code duplication (code clones type-1) activity in the aforementioned software artifact that is driven by these five research questions:

RQ1 What are the artifacts normally used to build robotics applications with ROS?
RQ2 To what extent do ROS packages have code clones in the ROS launch files?
RQ3 What are the characteristics of the clones in such packages?
RQ4 What is cloned in launch files with duplicated code?
RQ5 What is the impact of code clone presence?

3.2 Related Work

We have not found any other studies on code clone analysis in the ROS middleware nor on robotic applications using ROS. The closest relation is the work of Walck *et al.* [49] that evaluates ROS from another software engineering criteria: re-usability, by presenting an experience of developing robotic projects using ROS. The paper reports a case study of using ROS for implementing an in-hand manipulation robotic behavior. The authors highlight the re-usability capability of ROS due to its flexibility, interoperability, and reconfigurability. Work by Coleman *et al.* [53] studied the barriers to entry for developing robotic applications by reporting a case study of using the MoveIt! framework[2], an open-source tool for mobile manipulation in ROS. They point out that an important barrier to entry is finding the parameters that better fit certain algorithms of hardware components. This is related to our work because parameters are one of the most frequently copied kinds of data in launch files. Richards *et al.* [67] discuss the impact of the middleware on the software quality of robot control systems and report the development. FINROC: a robotic middleware that aims to

[2]MoveIt! framework, `http://moveit.ros.org/`

improve maintainability by providing optional APIs, refactoring code into reusable libraries, and separate framework-independent software.

From the Software Ecosystem's perspective, the work of Kolak *et al.* [3] exposes a complete analysis of the growth of the ROS Ecosystem. They found that although the number of new contributors and end users is increasing, the number of packages available grows at a slower rate, concluding that people tend more to reuse packages than to create new ones. They also found that collaboration is more common in special interest groups than between distinct individuals. Pichler *et al.* [68] studied the quality of ROS packages and their dependencies. They found that high-quality ROS packages tend to have more activity in their GitHub repositories: a high number of Pull Requests and Issues. They also report that high-quality ROS packages tend to depend on packages of the same quality standards.

3.3 ROS Internals

Technically, the ROS consists in its core stack (*e.g.,* `roscore`: the main process that coordinates nodes and topics, and `catkin` the build system), dedicated tools (*e.g.,* RViz for 3D visualization, and `rqt_graph` for active nodes and topics visualization) and third-party ROS packages.

ROS packages are a fundamental entity in the ROS Ecosystem. A ROS package represents a specific robotics feature to be reused by other, often more complex, packages. They are constituted by different and diverse types of files: source code of the nodes, build files, message, and service type definitions, documentation files, robot configuration files, XML files for launching several nodes (called *launch files*, discussed in Section 3.4) and the mandatory `package.xml` definition file.

ROS packages are meant to be developed, primarily, in C++ or Python (the tutorials of ROS are available in those programming languages). However, ROS provides bindings to allow to use of different programming languages, such as Java (ROSJava) [69], Lisp (ROSLisp) [70], and Smalltalk (PhaROS) [71].

3.4 ROS Launch files

A common robotics program developed with ROS is composed of many nodes interconnected by many topics. Instead of launching the application starting node by node manually, ROS provides a particular format to describe the launching of several nodes and its parameters: ROS *launch files.* In them they are described which nodes are being deployed, in which system, and with the parameters for its configuration. A launch configuration file is a file in XML syntax with the `.launch` extension.

We have a snippet at 3.1 which is part of a launch file. In the second line, the launch file sets the launch of `turtlesim_node` node. This is a node defined in the `turtlesim` package and it is given the name `sim`. The third line sets the parameters `publish_frequency` with the type `double` and a value of `10.0`. This parameter can be then accessed by the `turtlesim_-node` and it is also available for any other node that is launched afterward. Finally, an

Listing 3.1: ROS launch file example taken from ROS Tutorials and modified.

```
<launch>
        <node pkg="turtlesim" type="turtlesim_node" name="sim" />
        <param name="publish_frequency" type="double" value="10.0" />
        <include file="$(find other-pkg)/path/turtlebot-spec.xml" />
</launch>
```

external launch file is included in the current launch file. This means that all the nodes, parameters, and even other included files declared in the file `turtlebot-spec.xml` from the `other-pkg` package are included as if they were defined in this code.

3.5 Methodology

In this section, we present the methodology used to answer the research questions.

3.5.1 Software artifacts characterization

To answer the first research question we did a semi-automatic (following the idea of Robles *et al.* [72]). In their study, they propose a set of heuristics for categorizing software artifacts in Open Source projects. They rely on matching the kind of file with the extension of the file. Although it is not mandatory, the convention is to name files using the appropriate extension for each file. For example, a file whose name ends in `.c` is highly likely to be a source code file of a program written in the C language, a file that ends in `.launch` will correspond to a ROS launch file. When a file does not have an extension there are naming conventions used to define the kind of file. For example, `README` or `LICENSE` files are known to be documentation files and the license of the project respectively.

3.5.2 Code Clone definition

In this investigation of the code quality of the ROS ecosystem, we performed a clone analysis on ROS *launch files*. Our focus was on detecting the presence of type-1 clones among launch files belonging to the same package. We excluded type-2 or type-3 because, as *launch files* are configuration files: identifiers do matter. Despite other software artifacts (like source code), and as an example, in *launch files* we may define two different nodes using the same syntax structure (identifier, topic identifiers, arguments, etc.) and that should not be a case of a code clone. We also focused on intra-package code clones due to the finding that motivated the study. It would be very interesting to study inter-package code clones but that is out of the scope of this chapter. From this study, we cannot extrapolate the results to other software artifacts like the source code. Different approaches and tools should be used for that purpose. The experiment was performed completely using the Moose Platform [73], which provides several tools for software analysis such as an XML Parser, a basic text-based Clone Detector, a Clone Visualization Tool, and a Visualization engine for rapid results evaluation.

Other popular tools such as Nicad [3], CloneDR [4] and CCFinder [5] were evaluated, but the advantages of interconnected tools of Moose were more useful than the advanced features of the mentioned clone detection tools.

In all packages, we compared all pairs of launch files belonging to the package. We set a threshold to define what was going to be considered a clone to reduce noise. The sizes of launch files have, as its distribution, an average of 21.4 lines and a median of 13 lines. After a manual empirical revision, we set the threshold for considering a *clon pair*: a pair of launch files are considered as a *clone pair* if they have at least 7 identical lines in common. During the development of the experiment and its findings, this arbitrary number was enough for our purposes. The results actually confirmed that it was a conservative number as with loosened thresholds we would have arrived at the same answers for the research questions.

3.5.3 Metrics for Code Clone Study

For answering RQ2 we make use of the *overlap* [74] metric. This is a measure of similarity in a clone pair. let L_a be the set with lines of a launch file and the operator $|L_a|$ the number of lines in a launch file, then the overlap operator is defined as follows:

$$\text{overlap}(L_a, L_b) = \frac{|L_a \cap L_b|}{|L_a \cup L_b|} \quad \in [0, 1]$$

This means that the more lines two launch files have in common, the greater their overlap measure is which can be interpreted as that these two launch files are more similar.

Next, for answering RQ3 we wanted to understand better how the launch file code fragments are. To do that we studied in depth a subset of packages with 7 or more clone pairs. In order to prioritize the packages to study, we relied upon three metrics:

1. *average overlap*: between clone pairs (defined before).
2. the proportion of launch files in clone pairs versus all the clones defined in that package (including those which do not belong to any clone pair). A high value implies that many launch files are involved in clone duplication cases, and a lower value means that only a few launch files have code clones.
3. *clone cohesion*: it refers to groups of launch files that have many clone relationships between the different files, and it is calculated as the ratio between the number of clones and a number of launch files involved in clone pairs. The higher this ratio is, the higher the chance to find shared clone fragments between launch files.

The chosen packages are listed in the Table 3.2.

[3]Nicad Clone Detector, `http://www.txl.ca/nicaddownload.html`
[4]Clone Doctor, `http://www.semdesigns.com/products/clone/`
[5]CCFinder, `http://www.ccfinder.net/ccfinderx.html`

3.5.4 Data

On the 25th of March 2015, there were 1672 ROS packages registered on the official ROS website (`http://www.ros.org/browse/list.php`). 87% of the packages are hosted on Github [6], 8 packages on Bitbucket [7] and 176 of them do not report any hosting site. For our study, our sample considers all of the 469 GitHub repositories, containing 1560 ROS packages with 47796 files totaling 1.73 GB.

Relative to the presence of launch files in the packages, from all 1560 packages, 1027 (65.83%) defined no launch files. The rest (533) contained 2650 launch files, half of these packages define only one launch file. We also identified 160 packages that are for templating or testing purposes and were ignored. In terms of distribution, the vast majority (80.39%) defines up to 5 launch files, 96 packages (18.01%) define between 6 and 20 launch files. There are two outliers (0.43%). The first one, `cob_bringup`, collects all the scripts, launch files and dependencies to boot the *Care-O-bot*[8] (59 launch files). The second outlier is `jsk_pcl_ros` which provides programs for object recognition, it contains 72 launch files.

	Category	**Number of files**	**%**
Source Code	C++	6238	13.1
	C	5722	12.0
	Python	3524	7.4
	Lisp	1250	2.6
	JavaScript	375	0.8
	Bash	298	0.6
	Java	257	0.5
	Others (eg. ruby, qt)	889	1.8
	Total	**18553**	**38.8**
ROS Related	Launch files	2810	5.9
	Others	2757	5.7
	Package definition	1671	3.5
	Message definition	1237	2.6
	Xacro	668	1.4
	Service definition	574	1.2
	Robot Structure	530	1.1
	Total	**10258**	**21.46**
Documentation		**4538**	**9.5**
Build files		**3677**	**7.7**
3D Modelling		**2926**	**6.1**
Pictures		**2333**	**4.9**
Project metadata		**1073**	**2.2**
Non-categorized		**4438**	**9.28**

Table 3.1: Categorization of files per use

[6]Github, `http://www.github.com`

[7]Bitbucket, `http://www.bitbucket.com`

[8]Care-O-bot robot, `http://www.care-o-bot-4.de/`

3.6 RQ1: What are the artifacts normally used to build robotics applications with ROS?

We applied the methodology of Robles *et al.* [72] for characterizing the software artifacts found in ROS packages. The result of this characterization is depicted in Table 3.1.

As seen in the table, C/C++ and Python are the predominant programming languages in ROS packages. They cover 32.5% of the files. A second main category is ROS-related files (27.6%) which comprises from *launch files* (5.9%), `package.xml` file definitions (3.5%) to message definitions (2.6%). This group is followed by Documentation (9.5%) and 3D Modeling files (6.1%). The latter is required for simulation and robot object perception tasks.

The amount of non-categorized files could not be reduced because of the high variety of extensions (481), the most frequent extensions in this category had no more than 20 files and 462 extensions had 10 files or less.

Table 3.1 exposes a high diversity both in file types and technologies involved in the implementation of robotics software. This implies that the ROS middleware works with a wide variety of tools that manage different types of files for different uses. The ROS Ecosystem presents a high heterogeneity, which represents a challenge for its study and understanding.

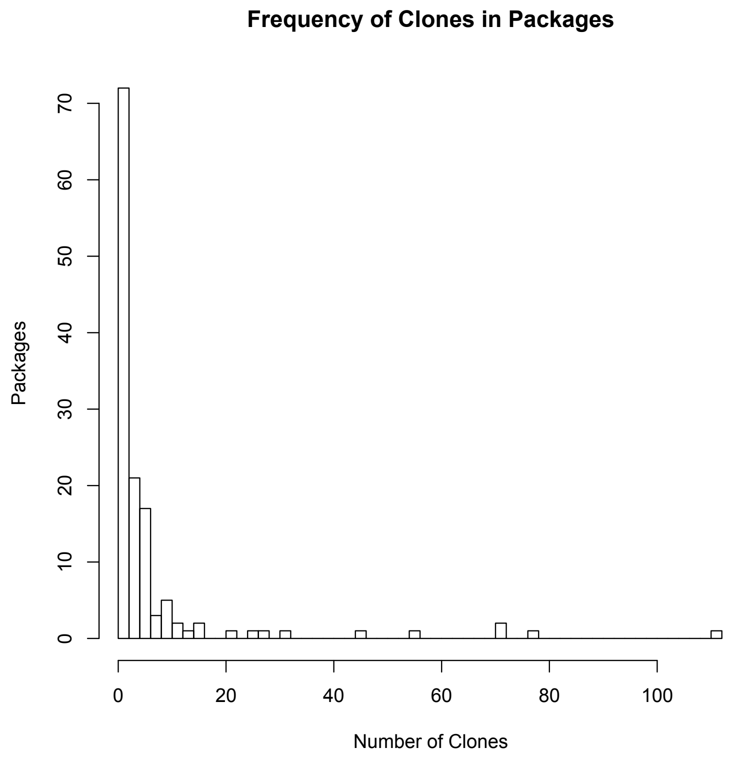


Figure 3.1: Clone distribution among packages

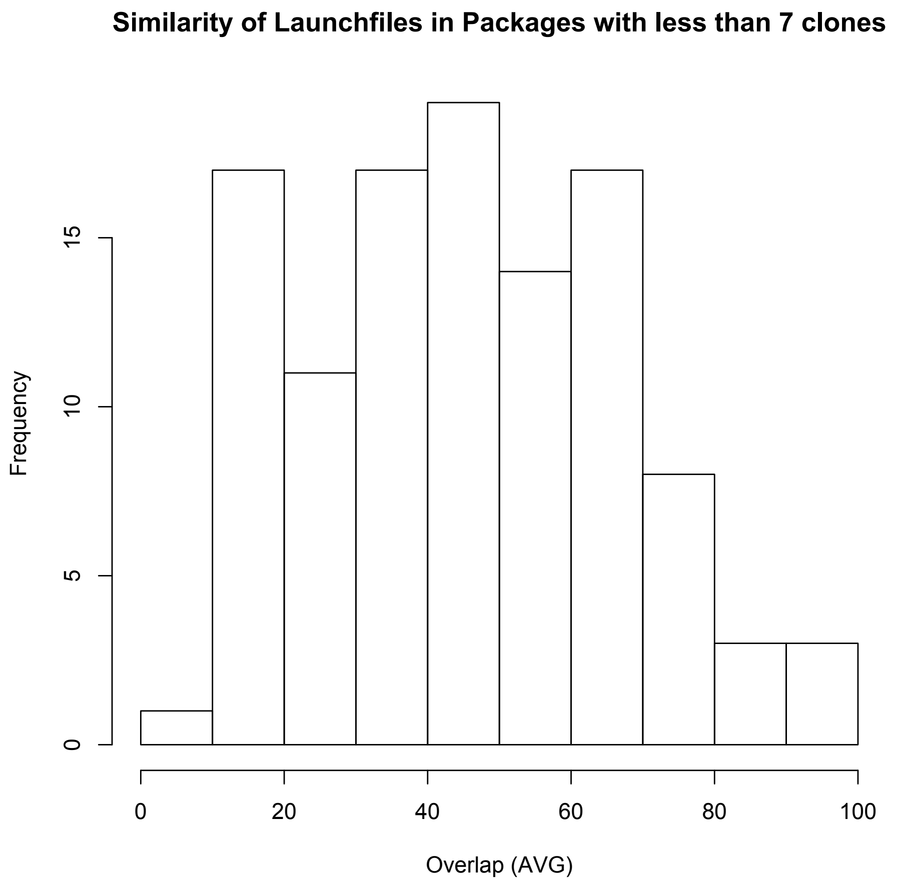


Figure 3.2: How similar are the launch files in packages with few clones.

3.7 RQ2: To what extent do ROS packages have code duplication in the ROS launch files?

Of the 1560 packages, 533 contain a launch file. 38% of these define only one launch file, 41% between 2 and 5 and 20% more than 5 launch files. We focused on intra-package code duplication, ergo, our target was 61% of the packages that contained more than one launch file: 330 packages. Of those 330 packages, 40% (133 packages) have clones. Considering the latter 133 packages, 110 of those (82.7%) contain 6 or fewer clones (see Figure 3.1): almost a half of packages have one clone (49.62%) and 21.23% of packages have two or three clones among their launch files.

Moreover, the packages that have less than 6 clones, present high similarity (on average) considering the launch files involved in these clone pairs

Figure 3.2 shows the frequency of packages with a certain similarity (on average) between their launch files (that belong to a clone pair).

The majority of the packages with clones have a similarity of 45% or more, which means that it is not rare that developers reuse important code snippets in launch files or in some

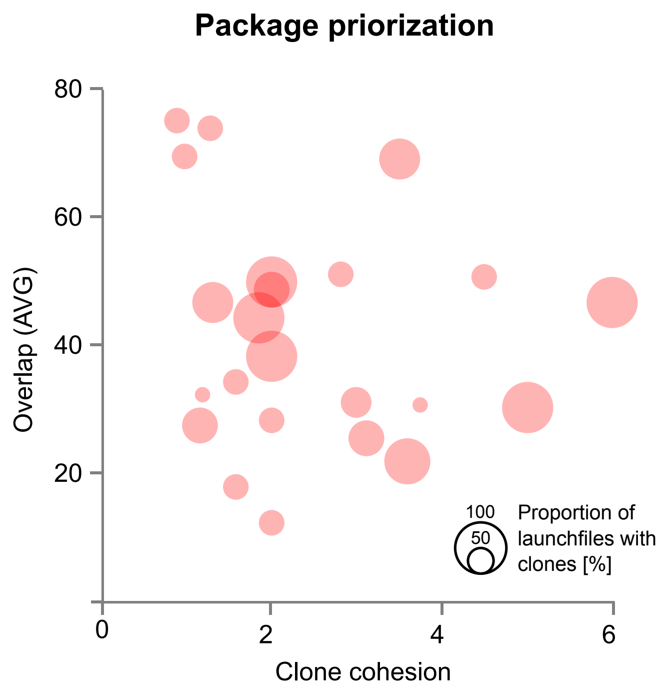


Figure 3.3: Packages with more than 7 clones

cases they copy&paste the whole file and edit certain lines.

Overall, 2/5 of packages present the results of certain code duplication activity. And in a considerable amount of cases, any launch file contains half (or more) of the lines of code from another launch file in the same package.

3.8 RQ3: What are the characteristics of the clones in such packages?

As we stated in the methodology (Section 3.5.3) we made use of three metrics for an in-depth study. Those three metrics are visualized in Figure 3.3 for the 23 packages that have at least 7 clones, this amount ensures the presence of at least 5 launch files in clone pairs. Big circles show packages with an important portion of their launch files involved in copy-and-paste activities. Packages away from the origin and approximately equidistant from both axes represent cases of several launch files that share big portions of code repeated in all of them with minor differences.

We performed a manual revision considering all the metrics aforementioned, we prioritized the 10 packages that are more relevant cases to study in depth. These packages are presented in Table 3.2.

Figure 3.4 shows how many launch files in the packages are involved in clone pairs. *(h)* and *(e)* are two packages that have more than 10 launch files and all of them present clones.

Table 3.2: Packages with more interesting clone cases

Package	**Files**	**Launch files**	**Launch files with Clones**	**Clones**	**Average Overlap**	**Clone Cohesion**
(a) hector_quadrotor_gazebo	20	10	8	28	68.86	3.5
(b) jsk_interactive_marker	128	24	11	31	51	2.82
(c) fanuc_lrmate200ic_support	65	21	10	45	50.29	4.5
(d) ueye_cam	22	5	5	10	49.76	2
(e) amcl	43	13	13	78	46.3	6
(f) openni_launch	15	11	7	21	30.66	3
(g) cob_bringup	77	72	19	71	30.4	3.74
(h) cob_controller_configuration_gazebo	18	11	11	55	30.17	5
(i) jsk_teleop_joy	70	12	8	25	25.32	3.13
(j) rtabmap_ros	214	36	31	112	21.59	3.61

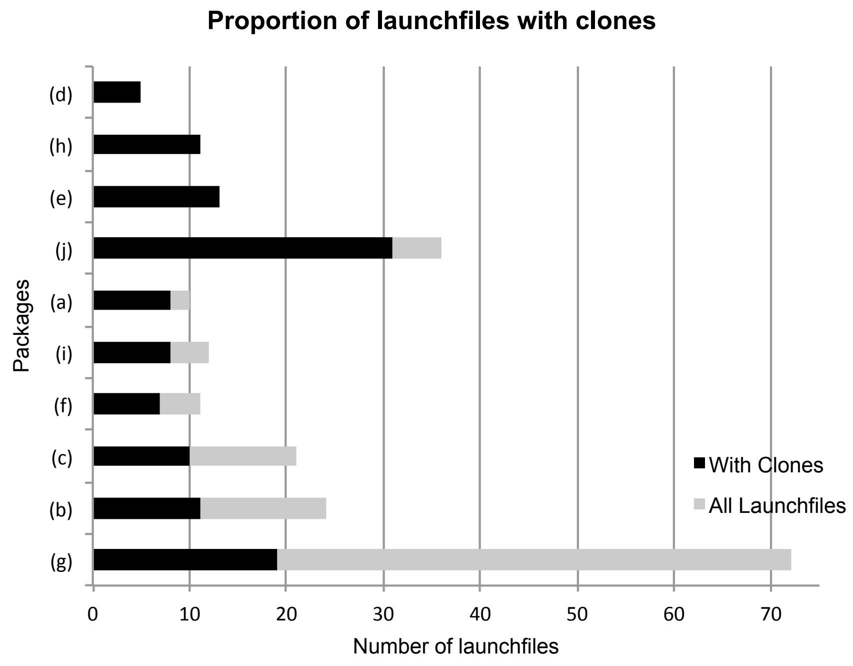


Figure 3.4: Proportion of Launch files with clones per package, ordered by proportion of number of launch files with clones.

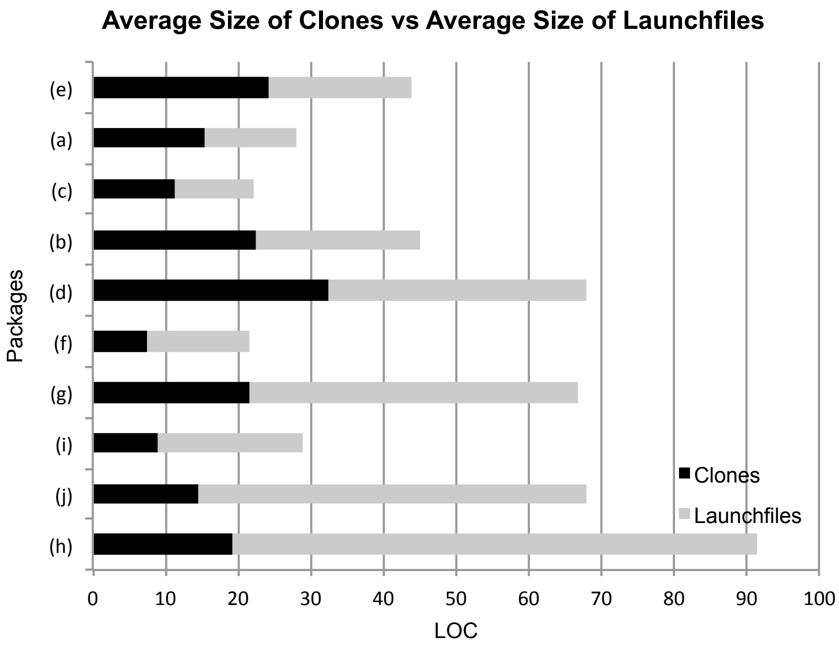


Figure 3.5: Size of all clones and its launch files, ordered by proportion of cloned code.

(j) contains a vast amount of launch files and over 86% of them present clones. *(g)* presents a lower proportion of launch files with clones, but as mentioned in Section 3.4, its amount of launch files is far further than the average and the number of launch files involved in clones and their cohesion are both high.

Figure 3.5 depicts what portion of the launch files with clones is the portion of code that is shared, on average. In *(a)* , *(b)* , *(c)* , *(d)* and *(e)* nearly 50% of the size of the launch files is covered by clone fragments. This means that a big portion of the whole launch file is identical to another launch file. For *(a)* , *(d)* and *(e)* , which are packages with high code cohesion of the clones, the duplicated code snippet could be shared with minor changes between many launch files. The other packages vary from 21% to 35% of cloned code, which is clearly a non-negligible portion.

From these 10 packages, we learned that there are two types of clone use. The first is where almost all the involved launch files have some code duplication but this is a small portion of the whole file (*e.g., (j)* and *(h)*). The second is where there are fewer clone cases but the cloned fragment represents a big proportion of the whole file, this means that the launch files can be described as one common code fragment with minor differences from file to file (*e.g., (b)* and *(c)*). *(e)* does not belong to either kind of clone use, it is one big group of launch files where each one is highly similar to almost all of the other launch files. This is one big code fragment repeated all over the launch files in its package.

3.9 RQ4: What is cloned in launch files with duplicated code?

Another question for us was to realize what kind of code is being cloned more frequently, and what kind of instructions are more commonly shared among cloned launch files. This can be done since the ROS Launch File XML format defines several tags, hence we can determine which tags are cloned the most. Arguably the most common tags are:

- *Node*: Specifies a node to be launched and the package where it is defined.
- *Include*: Specifies the path to another launch file whose tags will be imported.
- *Remap*: Allows to remap names, binding internal arguments defined in a node with others defined in the current launch file.
- *Env*: Sets values for environment variables that are valid under the scope of a node, launch file or certain machine.
- *Param*: Defines a single parameter to be set in the pool of global parameters (available for all running nodes).
- *Rosparam*: Allows massive parameter definition by importing them from an external YAML-formatted file or export current parameters to a file.
- *Arg*: Permits to abstract certain variables, delegating the concrete value to be set afterwards. This tag is meant to make launch files more abstract, favoring reusability through *include*.

We counted all the instances when the tags were cloned for each clone pair belonging to

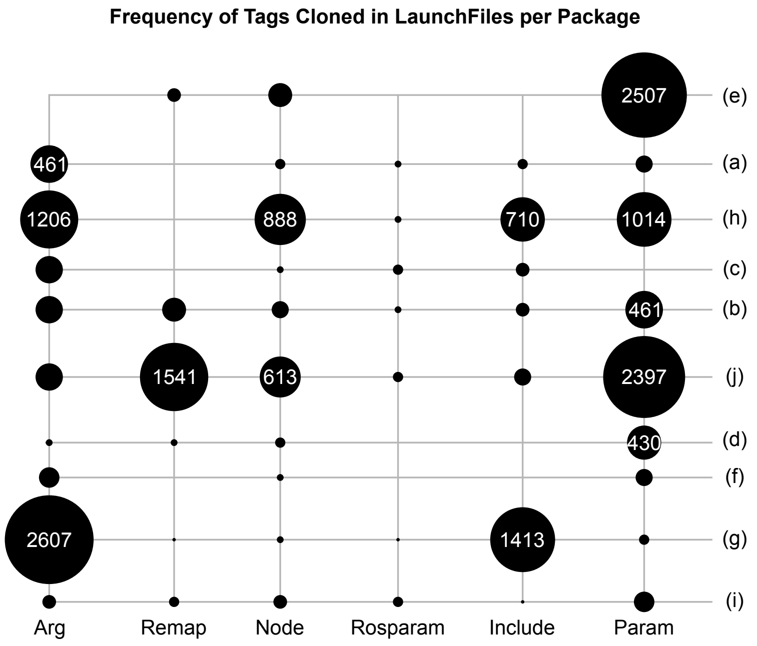


Figure 3.6: Which tags are cloned more often per package

the 10 prioritized packages. This is shown in Figure 3.6. We can see that there is no clear pattern of tag clone frequency among packages. For instance package *(j)* has many *Param* and *Remap* tag clones, however for *(g) Include* and *Arg* are usually cloned. Package *(h)* presents a regular amount of clones in all tags except for *Remap* and *Rosparam* . There was no use of the *Env* tag in the launch files under analysis. *Param* and *Arg* are used for similar cases: parameters are values that are used by nodes, and, arguments are given to nodes, hence, if we count them as a whole, we can see that their appearance in clone fragments is transversal to every package.

3.10 RQ5: What is the impact of code clone presence?

It is known that the presence of code clones refers to bad code quality. Although, their negative consequences can only be seen through the light of the evolution of the clones and the files in which they are present.

In order to establish the negative consequences of code clone presence in the prioritized packages (Table 3.2) we manually reviewed the history of selected launch files from them that have clones. We checked their commits on GitHub, looking for cross-changing situations, this is, when a change in a clone needs to be propagated among the involved launch files. We define four categories of such situations:

Value tweaking Hardware replacement or changes in test or deployment scenarios are common situations that involve changes in parameter values. These cases often impact tags belonging to clones yielding cross-changes situations where several launch files have to be updated. This was observed in packages *(f)* , *(b)* , *(a)* , *(j)* .

Changes in the file system Several *value tweaking* cases were related to modifications of the directory structure of packages. When this happens, all files referenced in `include`, `arg` or `rosparam` tags have to be updated. This situation was particularly recurrent in package *(a)* where the directory names were changed.

Package replacement In order to launch a node, its package name needs to be specified in the `node` tag. If a package is replaced or renamed, all references to it must be correspondingly changed. As an example, package *(j)* was affected by the renaming of the `rtabmap` package to `rtabmap_ros`. This required a change of all node tags referencing `rtabmap` in 8 launch files. It is important to say that this is not a rare situation, for instance, for example, the migration guide[9] from ROS Groovy to ROS Hydro points to the separation of the Stage package into two specific packages. This required all references from launch files to be modified.

Non-mutant clones We also observed several cases of groups of parameters or arguments that initially change but after some commits remain untouched. It appears that once stabilized, they become proven settings that are copied over to new launch files forming clones, accordingly generating potential cross-change cases. A further study needs to be executed in order to know if this is a common characteristic of ROS launch file clones.

3.11 Threats to Validity

For this study, we considered only type-1 clones and ignored the cases of type-2 or type-3. We may have found more cases of code snippets if we had considered those other two types. However, as we studied ROS configuration files, which rely on identifiers, these would have been ignored by an analysis based on type-2 and type-3, as these focus on syntax-related code snippets. Two code snippets in launch files can be very similar from a type-2 prism but they can describe two very different networks of topics and nodes: both have the same syntax but different identifiers (*e.g.,* topics and nodes' names). Nevertheless, this is only valid because of the kind of software artifact we studied: launch files. In case we studied source code such as C++ or Python code, clones type-2 and type-3 should be considered. In such a situation, it would be very interesting to search for these kinds of code clones and many studies performed on source code in other ecosystems could be applied.

We manually chose the 10 more interesting packages for the in-depth study as we needed to prioritize in order to find meaningful data for getting insights about the nature of duplication activities in ROS launch files. We may have used another criterion for choosing them as, for example, the top 10 most popular for users or the ones with more lines of code, or those with more dependencies to other packages, etc. We found important and non-trivial aspects of the code cloning activity in these artifacts following our customized criteria. Additional insights could be found following different criteria. An interesting endeavor would be to research code clones from an inter-package perspective. We focuses only on intra-package code clones as that was the nature of our motivational case. Our point of view could be enriched with this other perspective that may give interesting complementary findings.

[9]Migration Guide to ROS*Hydro*,`http://wiki.ros.org/hydro/Migration#Stage`

3.12 Conclusion

In this chapter, we show a first analysis of the ecosystem of the ROS robotics middleware focused on the presence of code clones in ROS launch files; configuration files that enable the setup and deployment of several robot processes at once.

We stated five research questions related to understanding better the use of these files and the code duplication activity involved in their development.

Answering the first question we found that ROS is a heterogeneous ecosystem in terms of the high variety of software artifacts that have to be used to develop robotics software. Ergo, ROS developers have to master different tools, languages, and technologies for their daily work. In this variety, this work focuses on code duplication of ROS launch files. Additional dedicated studies are needed to gather insights regarding source code duplication.

Second, from the packages that contained launch files, we saw that the majority (61%) define more than one launch file. We looked into the intra-package launch file code duplication and we found that 40% of them have clones: half of the packages have one clone. We studied the similarity of files involved in cases of code duplication they have; over 45% similarity: we can conclude that it is not rare that ROS developers copy&paste the whole file and edit a few lines.

In third place, we used 3 metrics to prioritize 10 packages that have at least 7 clones. We found that certain packages have more than 10 launch files and all of them contain clones. In other cases, we found that 5 packages have clones that represent half of their launch files. We also found cases of packages in which the same copied code fragment is present in many launch files with minor differences.

The four research question is related to the content that is actually being duplicated in the code clones. We did not find a clear pattern of tag repetition among code clones in different packages. In some cases, the tag *Param* is repeated and in others the tag *Arg* is more common.

In the last question, we characterized the cases in which there was a cross-changing situation due to changes in the duplicated clones. *Value Tweaking* such as hardware replacements or changes in test scenarios are common cases that trigger the propagation of changes in launch files that share a significant portion of identical code. Also, *Changes in the file system* and *Package replacement* were found to be typical cases of cross-changing situations. The fourth case is related to duplicated code that changed and after some commits, they started to stabilize: usually proven settings that need to be used in many launch files.

Finally, despite the identification of cross-changing situations, we set as future work to validate how frequent and harmful is the impact of these situations in the evolution of affected packages and the ROS ecosystem as a whole. In addition to that, it would be very interesting to observe how users and contributors face the aforementioned cases, how they manage the consequences, and if they have found any techniques to avoid them.

This initial study on the ROS ecosystem motivates a further and in-depth analysis of it.

Chapter 4

Understanding the Collaboration Dynamics of the ROS Ecosystem

In this chapter, we present the field study we conducted in order to obtain a better understanding of how people use ROS and what are their main issues using it. We also focused on discovering the collaboration dynamics in the ROS community. The field study is divided into three studies: a qualitative study via interviews and a focus group activity, and a quantitative study through an online open survey. We present the results of the field study reviewing the experience of using ROS: *Package Reuse Experience*, *Integrating a package*, *Package Reuse Failure*. We also show how users look for help and the way ROS users contribute to the ecosystem. This chapter is based on the work of the article *The robot operating system: Package reuse and community dynamics* [75].

4.1 Methodology

The methodology is a field study that is divided in three phases: interviews, a focus group and a survey to ROS users.

4.1.1 Interview Study

We started a study interviewing 19 developers that use ROS for their projects. The objective was to detect aspects of discomfort in their experience using ROS. The questionnaire (Annex A) covered a wide range of aspects of their developer experience: interviewees' background, general opinion regarding ROS, the perceived learning curve, particular or remarkable experiences using ROS, their knowledge about the available communication mechanisms that ROS provides, knowledge regarding the use of ROS artifacts and the roles of developers in their particular robotics projects.

We interviewed 19 developers and scientists from Chile and France working on robotics projects. Each interview was performed in person and on average the duration was one hour, and the period of the interviews was in Chile between March to August of 2016 and in France from September to November of 2016. Seven of the interviewees in Chile belong to a robotics laboratory at the University of Chile, the rest of the interviewees are located in different laboratories in France. None of the interviewees is a maintainer of a ROS package.

The interviews were recorded and then transcribed. The process was using an open coding process of these transcriptions analyzing 4075 sentences using 2673 primary codes and 1195 secondary codes from a universe of 257 possible codes. The codes emerged during the coding process. Using groups of related codes we summarize the data into memos that contain relevant parts of the interview for a given topic.

The outcomes from this study motivated the research questions: **RQ1**) What difficulties do users encounter when reusing ROS packages? **RQ2**) How do users contribute to the ROS ecosystem? and **RQ3**) What are the main contribution bottlenecks in the ROS ecosystem?

4.1.2 Focus Group Activity

We also performed a small focus group recruiting 4 participants from the Mechanical and Electrical Engineering department of the University of Chile who had varied experience in using ROS. The activity was driven by the feedback obtained in the interviews, specifically, it was oriented to the following topics: Package reusability, Package abandonment, and Community bottlenecks. The activity lasted 3 hours and was placed in June 2017. The focus group was also recorded and manually transcribed. There were subtopics that emerged from it: Expectations about packages, Debugging, Package Configuration, Abandoned Packages, Community Interactions, and Causes of missed contribution opportunities. We wrote a memo for each subtopic.

4.1.3 Survey study

In order to answer the research questions we carried out an open online survey of ROS users. We used the Survey Monkey survey platform. We invited the interviewees to take the survey via e-mail, and also made an open invitation to participate in a post published in ROS Discourse, shared on Twitter, and through ROS-related mailing lists. We obtained 119 responses up to August 21, 2017. The survey was voluntary and all the responses were treated anonymously. The majority of the participants were affiliated with a university (58%), followed by institutions in the private sector (24%), and lastly research centers (13%).

The survey consisted of 22 questions (Annex B) regarding three research questions. They can be grouped into 5 parts. The first part was about demographic information, background in robotics, level of ROS experience, and background in Software Engineering. The second part was related to their experience reusing ROS packages, the degree of dependence on packages, and the degree of familiarity with abandoned packages. The third part delves into the subject of package abandonment asking for reasons for package reuse failure. The fourth part asked about contribution activities: types of contributions made by the participants and the obstacles that they may have encountered when contributing. We asked also about their participation in the support systems (*e.g., ROS Answers*, *ROS Wiki*, mailing lists). Finally, we asked the reasons why a participant chose not to answer a question posted by a community member, e.g., on *ROS Answers* or *Stack Overflow*.

Table 4.1: Distribution of participants by level of experience using ROS.

User Type	**# Interviewees**	**# Survey participants**
Beginner	3 (16%)	31 (26%)
Intermediate	9 (47%)	39 (33%)
Advanced	7 (37%)	40 (41%)
Total	19 (100%)	119 (100%)

4.2 Results

In this section, we contrast the findings from the interviews with the survey results. Each interviewee has been assigned a numerical id and a level of experience with ROS: *Beginner* (*Beg*), *Intermediate* (*Int*) and *Advanced* (*Adv*). *Beginners* have up to 1 year of experience using ROS and/or use the basic features. *Intermediate* users have up to 3 years of experience and have used more advanced ROS features. *Advanced* users have 4 or more years of experience and/or have written packages from scratch. Their needs are more complex and they may have contacted package authors to solve their issues. Survey participants were classified by their amount of experience using ROS. The distribution of participants is shown in Table 4.1.

4.2.1 Package Reuse Experience

When developing a new feature, developers make use of existing packages to fully or partially fulfill that feature. The wide availability of packages to reuse is agreed upon among the interviewees as one of the main reasons for the success and popularity of ROS. Almost all of the surveyed participants have tried to reuse at least one community-maintained package (95%) and have dependencies on them (92%). From them, half of the participants have some or major dependencies which makes them sensitive to changes in those packages.

When a suitable package is found the next step for integration is to test it. This involves downloading, compiling, installing, configuring, launching, and monitoring the execution of the package. It is important to note that the integration phase is prone to fail: of the surveyed participants, 71% of them have reported it. In this situation, users tend to look for help using the knowledge from the ROS community.

This process of searching for a package and integrating it is a repeated cycle until the desired behavior is achieved or there is a technical incompatibility in the integration of the package. The interviewee I_{17}, who is an advanced user, expands on this point.

> We had a deadline some weeks ago. We [decided to] use ROS Control to move the robot around and we couldn't understand what was happening. Even now we don't get why we got this weird issue. The robot was so jerky so we wanted to stop it. [...] We looked at the code many times and just couldn't get it. And then we said: *OK, let's forget about ROS Control*, but a week before the deadline we rebuilt everything and it worked. ($I_{17,\text{Adv}}$)

4.2.2 Integrating a package

The package integration process is described below. First, the package needs to be *downloaded* via `apt-get` (in Debian-based OS') or cloning the repository. Second, the package needs to be *installed*, automatically via `apt-get` or using `catkin`[1], the default ROS building tool. Once it is installed, the next steps are the *configuration* process and *launched*: parameters, nodes, device identifiers and/or topic names must be set in order to properly launch it. Lastly, the user starts *monitoring* the behavior of the software by looking at the data flow, and nodes' statuses, among others, deciding whether the installed package responds as expected or if it needs to be reconfigured or avoided.

4.2.3 Why do users fail when reusing packages?

Around 74% of the surveyed participants that have tried to reuse 3rd party ROS packages report that they failed to do so (84 out of 113 participants). In the survey, we asked the participants about the reasons why they failed to reuse a package. We grouped the answers into 9 reasons from the responses given in the interviews. The list seems to cover all the reasons as only 4 responses were "Other reason". Five of the reasons were selected by at least a quarter of the surveyed participants:

1. *"The package was for an outdated ROS distribution"* (71%): Both *Beginners* (83%) and *Advanced users* (74%) selected this reason. For *Beginners*, due to lack of experience, they may not know that a package was outdated. In the case of *Advanced users*, they need more specific features that may not be implemented by active packages. Finally, in *Intermediate users* (59%), we think that they are more aware than *Beginners* about outdated packages, but unlike *Advanced users*, their needs are more standard.
2. *"I could not figure out how to use it (lack of documentation)"* (56%): As we expected, this reason was picked more by *Beginners* (61%) and *Intermediate users* (70%) than by *Advanced users* (44%). Since the documentation is needed by less experienced users, *Advanced users* can manage better without documentation by using other artifacts (*e.g.,* launch files, robot and world geometric modeling files, etc.) Note that the third responsibility of a package maintainer, according to community practices[2], is to "monitor and update package documentation". This result uncovers an important barrier to package reuse in ROS.
3. *"There was a bug that prevented the package from working properly"* (50%): *Advanced users* (59%) were more affected than *Beginners* (33%) and *Intermediate users* (48%). This could be explained as they use to work in more complex projects and depend on more packages. For specific features, there may not be a direct replacement for these buggy packages. As Mens [76] says, coming across buggy packages decreases the confidence in the ecosystem.
4. *"I did not succeed in configuring the package for my use case (launch files, etc.)"* (34%): Half of *Intermediate users* (52%) and a third of *Beginners* (33%) were affected by this reason. Configuring launch files requires deep knowledge of the robot being used and also how the package works. It is a tough task that documentation or pre-configured

[1]Catkin, ROS Wiki: 2015. `http://wiki.ros.org/catkin`.

[2]Maintenance Guide, ROS Wiki: 2017. `http://wiki.ros.org/MaintenanceGuide#Responsibilities_of_a_Maintainer`.

launch files can help to handle. We think that this is more frequent in *Intermediate user* because they start to face more complex projects and are not as experienced as *Advanced users* which were not as affected by this reason (28%).

5. *"I could not install the package"* (38%): The core packages are normally easy to install using `apt-get`. However, less popular and more specific packages usually need to be manually installed and built. Although fewer participants were affected by this reason, a third of *Intermediate users* (30%) and *Advanced users* (33%) reported this reason. A documented installation process is fundamental when considering a new package.

The other reasons for failing to install a package were: 6) *"The package is not compatible with my particular hardware"* (20%); 7) *"The package was for a newer version of ROS, mine is older"* (18%); 8) *"The package had performance issues"* (18%), and 9) *"I asked for help but could not find it."* (14%).

4.2.4 Looking for help

As is common in software development, users look for help online contacting more experienced users and looking for already existing solutions for their problems on Q&A websites. Our interviewees are no exception. Also, they consult for artifacts made by the community such as wiki pages and/or tutorials. Users also tend to contact package developers. The 4 sources for online support preferred by the survey participants are: *ROS Answers* (85%), *ROS Wiki* (79%), *GitHub* (59%), and *Interaction with Maintainers* (29%). They are discussed below.

■ ROS Answers

Searching Usually, ROS users tend to look for information regarding an issue through search engines. And posts from *ROS Answers* are one of the first links found. It is the first source of information for the majority of surveyed participants (85%). *Beginners* and *Intermediate user* look for installation/configuration guidance, clarification, and for code snippets. On the other hand, *Advanced users* besides looking for more specific issues, also participate in providing answers or moderating discussions. Two out of the seven *Advanced users* declared not to use *ROS Answers* very often due to particular problems that may not be already posted in the platform.

> I look for the issues in GitHub. Sometimes I search on Google first and then go to issues in GitHub. I never go to ROS Answers, I never found anything useful there. If searching does not return links to ROS Answers, I never go there. I don't think that I found something useful there. Maybe it is because [the packages] that I'm using are on GitHub, but not in actual ROS releases. ($I_{18,\text{Adv}}$)

Asking 18 of the 19 interviewees pointed out that in case of failing to reuse ROS packages, they use to fix it by themselves. In case of no solution is found, they ask colleagues for help. The very last option is to post a new question on *ROS Answers*. Our survey shows that the three main reasons why users evade asking new questions are: 1) 52% of *Beginners* think their questions are too specific, this percentage drops to 36% for *Intermediate users* and *Advanced users*; 2) 33% of the *Beginner* and *Intermediate users* think that their questions are a misunderstanding on their part (only 14% of *Advanced users* reported this reason); and

3) 32% of *Beginner*, 24% *Advanced users* and 15% *Intermediate users* think that it takes too much effort to write a full question. Other reasons include the presence of private/sensitive data that cannot be shared according to institutional policy, and long response times:

> On ROS Answers there was a question from 2012 that was exactly what I was looking for, but no one had answered it. ($I_{5,\text{Beg}}$)

Answering Questions on *ROS Answers* are usually answered by experienced users and specialists in certain robotics domains. Such users expect well-written questions and all the information/artifacts needed to provide a suitable answer. They also expect that the user who asked is available for further discussion or follow-up of the answer.

By November 19th of 2021, 62,441 questions have been posted to *ROS Answers*, of which 59.6% have been answered. An additional 2,575 questions have been tagged with the `ROS` label on *Stack Overflow* (50.9% answered). The answered/asked ratio for *ROS Answers* shows that it has an effective channel for finding help. Note however that this ratio is around 10% lower than that of popular communities on *Stack Overflow*: Python, Ruby-on-rails, and the R programming language all exceed an answer rate of 70%[3].

These are the main reasons why surveyed participants refuse to answer questions in *ROS Answers*: 1) lack of time (64%); 2) answering a question properly requires further interaction with people, to clear up the misunderstanding and ask for additional data/feedback (27%); 3) the setup data needed to replicate an issue is not available (23%); and 4) the hardware needed to replicate an issue is not available (20%). A minority of the participants claimed that they do not feel confident enough in their answers and/or think that another person would be able to give a better answer.

■ ROS Wiki

Searching The *ROS Wiki* is the main source of documentation and our interviewees consult it regardless of their level of experience with ROS. Surveyed respondents think that every package *must* document their features on the *ROS Wiki*, including current development status and any API specifications, configuration guide, and troubleshooting guide for common problems.

Creating and Editing pages As each package should have a *ROS Wiki* page, it should be created and updated by the package maintainer[4]. None of our interviewees maintains a public ROS package nor mentioned having edited existing *ROS Wiki* pages. Also surveyed participants rarely update or add missing documentation (only 25% do). Editing documentation is 10 percentage points below the next most popular form of contributing: submitting a new feature (36%).

■ GitHub

Searching for Code Examples A minority of *Advanced users* claimed that looking for code snippets is an important part of their workflow:

[3]Answer rate in *Stack Overflow* as of November 19th of 2021: Python 71%, Ruby-on-rails 73.4%, R 73.8%

[4]ROS Package Documentation Guidelines: 2016. `http://wiki.ros.org/PackageDocumentation`.

> [One of the main strengths of ROS in my opinion is] the support – the vast quantity of examples you can find: it's amazing. I haven't seen that in any other framework for robotics. You just search on Google and you find examples. There is always someone who has already done it [and shared it] ($I_{3,\text{Adv}}$)

Users often look for code snippets that show how to use a package, find useful parameters, or build workarounds. However, examples could be outdated and could be not relevant which is the case for abandoned packages.

Interacting with Maintainers through the Issue Tracker As *Advanced* interviews face more complex or particular issues, they avoid using *ROS Answers*. They interact directly with package maintainers through the package repository Issue Tracker.

Private Communication with Maintainers

In some cases, it is inevitable to contact the package maintainer directly. This way the communication can avoid revealing sensitive information in public channels. The quote below shows this behavior.

> If not I would just ask. Sometimes I don't like to ask directly in the GitHub page. When I want to ask someone I email directly to the authors. I feel weird exposing myself by publishing things publicly. ($I_{14,\text{Int}}$)

4.2.5 How do users contribute to the ROS ecosystem

According to our survey, contributions can be sorted by (descending) frequency: *Bug Reports*, *Q&A in ROS Answers*, *Bug Fixes*, *Feature requests & submissions*, *Documentation updates & additions.*

Bug Reports Bug reports are published using the Issue Tracker in the corresponding repository. This type of contribution is common regardless of the level of ROS experience but more frequent in *Advanced users* (77%) and *Intermediate users* (67%). It is considered a contribution because it benefits the package maintainer to have feedback about their work. Fixes, reported bugs or errors will benefit the whole community improving the health of the ecosystem.

Q&A in ROS Answers Contributions on this Q&A platform are mostly guided by *Advanced users* (Answering: 68%, Asking: 66%) rather than less experienced users. More than half of the *Intermediate users* (63%) ask new questions but only 26% claim to have answered a question. Of half of *Beginners* who had contributed, one-third of them had posted or answered a question.

Bug Fixes 76% of *Advanced* and 67% *Intermediate users* indicated that they submit bug fixes. Like bug reports, bug fixes directly benefit the package maintainer and indirectly benefit all the package users.

Feature requests & submissions These contributions are also handled through the Issue Tracker of the *GitHub* repository. *Feature requests* are valuable feedback on the users' needs regarding the use of the package, *Feature submissions* correspond to the implementation of new features that need to be reviewed by the package user. While both are done by non-beginners, for the first, around 46% is done by *Advanced users* and 40% by *Intermediate users*, for the second half of *Advanced users* claimed to have submitted new features and only 19% for *Intermediate users*.

Documentation updates & additions Updates or additions to package documentation are mostly made by *Advanced users* (updating: 37%, adding: 32%). This contribution benefits directly the potential users of the package but also the maintainers since it removes part of the burden of documenting the package.

4.3 Limitations of this study

Regarding the qualitative study, there was a restricted number of interviewees and from only two countries (Chile and France). This is due to the difficulty in getting access to interview participants. Consequently, this part of the study cannot be generalized. In order to remedy this, we conducted an open online survey over a broader audience from the ROS community. The survey was responded to by participants from a variety of countries and backgrounds.

Our interviewees were all scientists. Other ROS users (*e.g.,* practitioners in industry, students/teachers, hobbyists) may have different motivations, needs, and interests. Thus, it is possible that different profiles face different difficulties and experiences using ROS that would open up different findings. However, a significant proportion of surveyed participants (25%) were from the private sector. We compared the distribution of answers between scientists and non-scientists and they seem to be quite similar. While non-scientists may face different issues, they also struggle with the problems previously mentioned such as Package Abandonment.

Internal bias is present in the analysis of the transcription as the coding process was conducted by only me. In order to increase the reliability of the findings, I got feedback from my advisors on the coding process during meetings. This was done by checking the tags assigned to random sentences in the corpus. The summaries of the transcripts into memos were done collaboratively by my advisors and me.

Additional interviews, focus groups, and large-scale surveys are needed in order to increase the confidence in the findings of this study and be able to propose a stronger generalization.

4.4 Conclusions

In this chapter we presented the outcomes from two qualitative and one quantitative study to answer three research questions: **RQ1**) What difficulties do users encounter when reusing ROS packages? **RQ2**) How do users contribute to the ROS ecosystem? and **RQ3**) What are the main contribution bottlenecks in the ROS ecosystem?. The first consisted of interviews with 19 developers and scientist that are users of ROS. The second one, a focus group of 4 participants all of them undergrad students and also ROS users. The third is an online open survey that was answered by 119 individuals.

We found that the availability of packages to reuse is the major comparative advantage of ROS among its competitors with 95% of our surveyed participants having used at least one package maintained by the community. Despite, this high dependency of users to 3rd party packages, 74% of the responses claim that they have failed trying to reuse such packages. The main reasons are outdated packages and lack of documentation.

Regarding contributions, they are done mostly by *Intermediate* and *Advanced users*. we found that users contribute through several channels: the website *ROS Answers* for posting questions and answers (sometimes with code snippets, configuration examples, etc.), GitHub for feature requests, feature submissions (40% of Advanced and Intermediate users) and bug fixes (over 75% of Advanced users), and finally *ROS Wiki* for documentation contributions (mostly Advanced users).

The third research question will be answered in depth in the next chapter, in which the *Contribution Bottlenecks* are presented and recommendations to overcome them.

Chapter 5

Contribution Bottlenecks and Recommendations

Contributions among members of a software ecosystem are the basic dynamic for a healthy software ecosystem. As we previously introduced, in this chapter we aim to answer the third research question: *What are the main contribution bottlenecks in the ROS ecosystem?* In the first part of this chapter, we present a series of bottlenecks in contribution identified in the interviews and that were validated in the survey. We also delve into the most critical bottleneck: *Package Abandonment.* In the second part, we propose 5 recommendations to overcome the previously discussed bottlenecks. In the second part of the chapter, we present 5 recommendations based on current studies to overcome such bottlenecks. This chapter is also based on the work of the article *The robot operating system: Package reuse and community dynamics* [75].

5.1 Bottlenecks in Contribution

In this section, we present the 5 identified bottlenecks in contribution.

B1. Lack Of Time. This bottleneck is definitely the most frequent bottleneck voted by our surveyed participants. More than half of them (59%) indicated this bottleneck as a reason for not contributing. Lack of time holds up any kind of contribution mentioned in the previous chapter. As an example, the interviewee I_{13} claimed not to have contributed in answering questions:

> And what I should have done and didn't do was to reply to those who had those problems. I could do it, but at the time I was trying to develop something else so I forgot about it. ($I_{13,\text{Adv}}$)

B2. Package Abandonment. This bottleneck occurs when a public 3rd party package is not maintained anymore or the package maintainer is indefinitely unavailable. This situation implies that any contribution to the package will not be reviewed nor integrated. It is acknowledged by 90% of survey participants, and 63% of them claim to have experienced it regularly. The following quote, from an *Advanced user*, presents the case of an outdated package in which build-files had to be updated to the, by then, new building tool (`catkin`) to make it compatible with current ROS versions.

> It's common that you find a package and it's only compatible with ROS Fuerte and the only way to make it compatible with newer versions is to catkinize it. But the student who made it finished his thesis and left the package as it is [, abandoned]. They [, the community,] could make a "foster home" for these packages, so that they can be maintained. ($I_{4,\text{Adv}}$)

Community Managers are in charge of sharing knowledge regarding the contribution channels, and quality assurance guidelines as well as lowering barriers to entry that would hinder the possibility of contributions. The next three bottlenecks are related to their work.

B3. Lack of confidence in the value/quality of a contribution. 26% of the surveyed participants acknowledge having been in this situation. Mostly *Beginners* and *Intermediate users* hesitate to contribute because they are not sure of the correctness, quality, or value of their contribution.

> It's contradictory because I like the information from there, but normally when I know the answers I don't really contribute. I am not sure If I know how to answer properly [...], so normally I don't really contribute. ($I_{14,\text{Int}}$)

B4. The contribution could be too specific to my problem, domain, hardware, or research problem. *Intermediate users* and *Advanced users* think that the artifacts they work on may not be useful to be shared due to their specificity of them. This bottleneck is acknowledged by 31% of surveyed participants and impacts directly the availability of bug reports, bug-fixes.

> I have made my own packages but I have never published them. Because they are specific to my work, like, for example: we use multiple motors to control the humanoid robot [, for] those motors we made our own packages. They were very specific to our robots. ($I_{10,\text{Int}}$)

B5. Workflow is unknown or not clear. A minority (14%) of surveyed participants claim not to have a clear understanding of how to share their contributions because the workflow is unknown.

> Many developers don't follow REPs[1]. This means that they create artifacts in one way, and others do so in different way. This should be more uniform, but it is fine because people create what they want and they contribute it. This is one of the reasons why I didn't wanted to share my code: I write it [(the code)] in my own way, which is not the proper way to share it. I then switched focus to another project or task, and left it as is. ($I_{4,\text{Adv}}$)

[1]A ROS Enhancement Proposal (REP) is the official documentation of architectural and design decisions for the ROS middleware: 2000. `http://www.ros.org/reps/rep-0000.html`.

Steinmacher *et al.* [77] talk about the importance of this issue due its impacts on the willingness to collaborate by the community members. It also represents a big barrier to entry for newcomers. The *ROS Wiki* provides insights about how to contribute[2]. We are not sure if our surveyed participants and interviewees were informed about the availability of this documentation.

5.2 Package Abandonment

We have seen that this bottleneck is caused, in the robotics domain, by at least two phenomena. First, the development of some packages is the result of an investigation or a side product of research. After the robotics student has obtained the academic degree the development of the package is left unattended. Second, when a developer starts working on a package and leaves it unfinished to start working on another one which may not be a replacement for the first.

This situation is reported by 90% of surveyed participants and for 63% it is a common issue. For *Intermediate users* and *Advanced users* this bottleneck occurs from "very often" to "all the time", presumably because their use of 3rd party packages is more common than *Beginners*. The interviewee I_{17} describes this situation using the popular framework MoveIt, headlining not only the package abandonment barrier but also the lack of time:

> For example there is a package in MoveIt that is very popular. And a lot of people wanted to add some fixes because they were a lot of issues. All the people that were maintaining the package were like "gone". So there is now only one [developer], I guess, who is maintaining this package that everybody needs but nobody is really interested in maintaining. These are the strengths and the weaknesses to being open source, I guess. ($I_{17,\text{Adv}}$)

5.2.1 Impact on Package Reuse and Contribution Dynamics

We consider package abandonment to be a major issue. More than 92% of surveyed users depend on packages from the community. Half of *Advanced users* and *Intermediate users* depend moderately to highly on 3rd party packages. Hence, ROS users do rely on external packages to fulfill their projects. Moreover, it is important to highlight the high numbers related to failing to reuse also they acknowledged that they were dealing with an abandonment package. An *Intermediate user* who participated in the focus group claims that the scope of debugging doesn't reach the ROS infrastructure: "*[the debugging process starts] assuming that the bug is hidden somewhere in your own code*" — $I_{22,\text{Int}}$. This reflects the high confidence in external ROS packages which can make the debugging process difficult in case of reusing a buggy abandoned package.

We identified 3 ways in which the phenomenon of Package Abandonment threatens the trust in reusing 3rd party ROS packages:

[2]Get Involved, ROS Wiki: 2017. `http://wiki.ros.org/Get%20Involved`.

1. Abandoned packages weaken the ROS ecosystem reliability While reusing ROS packages is common in the workflow of ROS users, several interviewees mentioned that searching for packages can be time-consuming.

> So most of the time I look for something people have already made. I feel like I lose a lot of time searching a lot of things, trying packages that are, like, not working anymore. They were made for old versions of ROS, like Fuerte, but they are not working for Indigo or Kinetic. Then you update the package yourself, but if you don't have any documentation about the internals of the package, you don't necessarily know where to change stuff. You can lose a lot of time trying to use other packages. ($I_{18,\text{Adv}}$)

Users can spend a significant amount of time installing a package, configuring it, and understanding how to integrate it and then be disappointed when finding that the package is not being maintained anymore.

2. We cannot foresee which packages are abandoned When searching for packages to reuse users do not know how to filter out abandoned packages. They lose time before realizing the status of the package. Although there exists an effort to list such packages[3] users seem not to be aware of it. Is important to mention that the list is not exhaustive, there are still abandoned packages that may have not been reported.

3. A missing maintainer cuts off contribution flows When the maintainers are missing there are many common contributions that cannot be used by the community. Bug reports are ignored and documentation contributions cannot be added. In other contributions that take more effort such as bug fixes or feature contributions, the *pull requests* are not reviewed. This represents a waste of time for contributors who already claim to not contribute as much because of lack of time. *Advanced users* who tend to clarify more advanced or specific questions with maintainers do not have any responses to them. Lastly, questions in *ROS Answers* regarding such packages will remain unanswered. In summary, the absence of a maintainer creates a bottleneck in the evolution of a package, cutting off the contribution flow.

Not receiving an answer is a strong barrier for contributors, which may demotivate them and make them refrain from contributing [78]. This severely harms the contribution dynamics of the ecosystem, which is a fundamental form of interaction in open-source communities. Moreover, it is reported as a common barrier for peripheral contributors [79]: it ultimately threatens the health of the ROS community and its growth.

5.2.2 How do users handle these threats today?

Even in the case of encountering an abandoned package, users need to continue the development of their projects. Our interviewees reported two main workarounds to deal with this scenario.

The first workaround is to make a useful piece of code to be ROS-compatible. This process

[3]ROS Orphaned Packages: 2021. `http://wiki.ros.org/OrphanedPackage`.

is denominated *ROS-ify* and it is to wrap a reusable piece of software: a library, application, or an entire middleware. The wrapped software is converted into a ROS node that can “live” and interact with other nodes in a ROS application. Interviewees $I_{17,\mathrm{Adv}}$and $I_{18,\mathrm{Adv}}$ commented about having wrapped the OROCOS [26] robotic middleware after several attempts to use the official `ros_control` package[4] . These kinds of workarounds that, accordingly to our interviewees, are not rare, are not suitable in the long term.

The second workaround is to fork the repository of the abandoned package and adapt it without the supervision of an absent maintainer. Although this solution can be valuable for users who need to reuse, adapt or extend an abandoned package, its further maintenance is not guaranteed. Hence this solution is worthy individually but not for the ecosystem as a whole.

As we aforementioned, these two workarounds do not address the problem properly in terms of helping the ecosystem. The abandoned package is still there and new users will suffer the mentioned issues in reusing that package or contributing to it. Although the problem may be solved individually problems of duplicated effort can occur by many users *scratching-their-itches* with zero impact on the ecosystem’s health.

New collaboration strategies are needed to better communicate with users and collaborators for the sake to do the work of the absent maintainer. This is intended to benefit a healthy evolution of abandoned packages in the ROS ecosystem.

5.3 Recommendations for overcoming contribution bottlenecks

In the following we present in depth 5 recommendations for overcoming the previously described bottlenecks in contribution.

5.3.1 *Recommendation 1:* Identify and Predict Abandoned Packages

Even though package reusability is the most valued advantage of ROS, 74% of our surveyed participants report to have failed reusing it. In many cases it was because of trying to reuse an abandoned package. Users waste an important amount of time before realizing that situation. Currently the ROS community has no methods to identify an abandoned package or predict when a package will stop being maintained. Encountering abandoned packages makes the ecosystem less reliable.

The only way that a ROS user has to know if a package is abandoned is to review the list of *Orphaned Packages*[5]. This is a list of packages that are no longer being maintained. The list is updated by volunteers or the same maintainers of those packages letting know to the community that those packages will not be maintained in the future. This list may be

[4]ros_control - ROS Wiki: 2021. `http://wiki.ros.org/ros_control`.

[5]Orphaned Packages, ROS Wiki : 2021. `http://wiki.ros.org/OrphanedPackage`.

not updated nor be complete. As the ROS' Maintenance Guidelines[6], the maintainer is the person in charge of updating the state of the package. But in the case that the maintainer is already gone, it might not have updated the status so the community does not know that the package was abandoned.

Automation seems to be a reasonable solution in a community with members who lack of time or have doubts related to the workflow to notify the community about a package being abandoned. An automated approach does not suffer from those bottlenecks and, by mining software repositories may provide some support. There are examples using heuristics based on the (in)activity of a package development: (*e.g.,* number of commits last week, number of forks or pull requests) [80, 81], activity in the bug tracker or mailing lists, new contributors joining [82] or release frequency.

A community that acknowledges its abandoned packages can take action about it. It first should warn possible users about the status of such packages so they do not lose their time trying to make it work. Maintainers of packages that depend on abandoned packages should be warned too so they may find replacements for them or fork the repository to adapt it with minimal work and make it possible to keep using it. Although the last solution solves the problem for an individual, the situation may occur to other developers duplicating efforts in the ecosystem. This can be managed by communicating about this minimal maintenance even if the package user does not want to be the responsible one for the package evolution. In the second step, the community can make an open call for *adopters* of that package. This approach is done by communities like WordPress or Jenkins [83], they put an *adopt-me* tag on a package list web page. The community will know which packages need a maintainer or an urgent maintenance task that may benefit the whole ecosystem. In the case that there is no interest in maintaining such package, the community may provide a list of replacements for that dependency.

Another possibility to overcome this problem is to predict when a package is going to be abandoned: how probable is it that the development of a package will stop or that the contributions to this package will not be processed anymore? Several works [81, 84, 80, 85] agree in that the threshold of 1 year of inactivity is enough to be considered abandoned. This means that any attempt to predict the probability of abandonment can be achieved after one year. Coelho *et al.* proposed an approach for identifying abandoned projects in Github [82] that goes beyond using a threshold: an abandoned package may have little development activity. They classified projects using machine learning models, specifically, using random forest classifiers. The sample is relatively small (127 projects) and they obtained a precision of 80% and recall of 96%. They also, reusing their methods, provided a metric for abandonment prediction called *Level of Maintenance Activity (LMA)*.

Related to the approach based on heuristics there is a vast amount of studies: Constantinou *et al.* empirically studied developer retention in the RubyGem and NPM ecosystems [86]. They found that both weak commit intensity and a long period of inactivity in committing are associated to a high probability of abandonment within an ecosystem. Coelho *et al.* surveyed the maintainers of over a hundred of projects on *GitHub* in order to study the reasons

[6]Orphaned Packages, ROS Wiki : 2017. `http://wiki.ros.org/MaintenanceGuide#Role_of_a_Maintainer`.

for failure [85]. Note that this relates to our work since each ROS package is a project in the ROS ecosystem. The following are reasons that Coelho *et al.* confirmed as causes for project failure on *GitHub*: the project became functionally obsolete, the main contributor does not have enough time to work on the project or was no longer interested in the project. These heuristics should be taken into account for any predictor of abandoned packages.

5.3.2 *Recommendation 2:* Provide an Informative Package Repository

Mature and popular ecosystems such as LATEX document engine or programming languages as R or JavaScript rely on an informative and complete catalog of available packages to reuse: CTAN[7], CRAN[8] and NPM[9], respectively. In there, users can obtain information about the features of the package, the repository, its dependencies, installation and configuration guidelines, the contact of the maintainer, and the development status, among other information. ROS users may need more information such as troubleshooting experiences, performance benchmarks, hardware compatibility, or alternative packages. Some of the previous information can be obtained from other users on Q&A websites. As a consequence, users are more informed about the packages they need to reuse, their real availability, and usage guidelines in order to include them as a dependency in their projects.

The ROS community has two main sources of information about available packages. The first one is *ROS Index*[10], a community-maintained package index catalog. This effort covers many of the features previously mentioned: a list of repositories and the packages available in it, package description, the ROS distribution compatibility of the package, the version, the maintainer, the package dependencies, tutorials, and questions in *ROS Answers* tagged with the name of the package, among other characteristics. Many of the information fields come from an XML Package Manifest file filled by the maintainer except the related questions that are automated.

The second repository of package information is *ROS Wiki*. Packages are manually indexed on the *ROS Wiki* by redacting a dedicated page for a package. Although this task is, according to the guidelines, filled by the maintainer, users may update/correct the content of the page. Guidelines refer to documenting a package when possible before release but there are no standards for the minimum completeness and quality of this documentation.

As we have shown, both repositories of package information are manually updated, thus the level of completeness and accuracy of the information depends on the effort of individuals. This means that they are at risk of being outdated. For instance, abandoned packages are expected that maintainers do not update the documentation. Such packages may still appear as "active" when they are not.

Automation seems to be the way to go when considering the commented bottlenecks: lack of time, knowledge of contribution workflow, and package abandonment. A mechanism of

[7]CTAN: Comprehensive TeX Archive Network: 2021. `https://www.ctan.org`.
[8]The Comprehensive R Archive Network: 2021. `https://cran.r-project.org`.
[9]NPM - NodeJS Package Manager: 2021. `https://www.npmjs.com`.
[10]ROS Index: 2021. `https://rosindex.github.io`.

detecting or predicting abandoned packages would show this information in the *ROS Index* catalog warning potential users of an abandoned package not to rely on it. Storey *et al.* propose an approach of using a bot [87] for assisting in testing, support, and documentation. Regarding quality and completeness of documentation, automating its measurement [88] would invite contributors to improve them. These actions would help to improve the quality of currently available information provided in the package catalog facilitating the decision of reusing a package [89]. Moreover, sporadic contributors would be aware of contribution opportunities lowering common barriers of contribution [79]. Finally, showing the activity of packages, updates, and enhancements attracts new users and contributors [90]. Solutions like the aforementioned are necessary as it is rare that ROS client applications usually depend on the core stack of ROS and not on the packages maintained by the community [3]. Such an approach would help on promoting the use of 3rd party ROS packages considering their wide diversity.

5.3.3 *Recommendation 3:* Recommend contribution opportunities to qualified community members

Recommender Systems in Software Engineering are useful to provide information from large, heterogeneous, and complex data that could not be done by individuals [91]. A recommender system may find members of a community that are more suitable for making certain contributions to the ecosystem [92]. For instance, it could identify experts to answer a particular question in *ROS Answers* or owners of an expensive robot could replicate an existing bug associated with the hardware. Choosing the appropriate recommender system depends on the particular data set to work with [5]. For example, studies of expert finding in *StackOverflow* are based on Collaborative Filtering methods, but they may be not applicable for a smaller community as *ROS Answers* with 37 thousand of questions answered, vs *StackOverflow*: 31 million of answers: such methods are not directly reusable. We propose to use a hybrid approach: Content-Boosted Collaborative Filtering [93]. This mechanism complements the Collaborative Filtering (CF) approach with Content Based (CB) methods. The CF approach may use the user activity to model the willingness to collaborate and participate in answering a question. The CB would benefit from a rich vocabulary gathered from the terminology of the variety of disciplines under robotics, hardware names (robots, actuators, sensors, etc.), and software artifacts (package names, software technologies, type of files, etc.) The ROS community is comparatively smaller than other communities which makes the cold start problem a little bit more difficult to address due to the scarcity of information.

Another example of the use of a recommender system is to find qualified contributors to address a fragment of a normal contribution (*e.g.,* bug fixes, bug reports, update documentation, new feature requests, etc.) into smaller micro-tasks. Such fragmented tasks would cost less time and effort reducing barriers for contributors. As an example, a bug report and bug fix would be fragmented into verifying the completeness of the report, reproducing the bug, identifying affected modules, proposing a fix, reproducing the bug with the fix, and integrating the fix. Each one of those micro-tasks is much more manageable and easy to address than the whole task of reporting and fixing the bug itself. A system like an issue tracker may track the status of the contribution, estimation of effort of a micro-task, and level of completeness of the contribution and let the users vote for the integration of such contribution.

Such mechanisms directly address the bottlenecks B1, B2, B3 and B4, in the following ways:

B1. Lack of Time: As time is scarce, recommendation engines can reduce the time of members of the community to a minimum. In the case of Expert Recommendation Systems, users do not need to waste time looking for someone qualified to answer their questions. On the other side, experts do not need to look for knowledge contributions, a pre-selected set of questions are suggested because they are suitable for their expertise. In the same way, micro-tasks can be suggested and notified in order to invite peripheral contributors and One-Time Contributors [11] by lowering their barriers for contribution [79].

B2. Package Abandonment: Through the idea of micro-tasks, the community could keep a minimum of maintenance of an abandoned package. Such tasks are smaller and cost less effort, they can be suggested by a recommender system to qualified members of the community to address them.

B3. Lack of confidence in the value or quality of the contribution: Experts will be aware of the value of a possible contribution by being suggested as answerers of an unanswered question in *ROS Answers*.

B4. The contribution could be too specific to my problem, domain, hardware, or research problem: Developers realize that they can contribute to a specific problem, domain, hardware, or research problem because they are actively asked to contribute to an issue or unanswered question with that specificity.

These mechanisms may partially address the bottleneck *B5. The workflow is unknown or not clear* in the way that a contribution suggestion can remind or teach the Contribution Policies (*e.g.,* how to contribute guidelines). This can ease the possibility of future contributions. In the long-term, effective implementation of such recommendation mechanisms may increase the *truck factor* in packages and knowledge and encourage new contributors or maintainers for packages that are necessary for a healthy growth of the ROS ecosystem [94].

5.3.4 *Recommendation 4:* Limit breaking changes

The ROS ecosystem has experienced several cases of breaking changes that have had a significant impact on the community. An example of this is the upgrade of the ROS distribution ROS Groovy to ROS Hydro where the build system was changed from `ros build` to `catkin`. This situation, reported by I_4 meant that all abandoned packages were immediately outdated.

Another situation occurred in the release of ROS Kinetic where there were changes in the API of the build system and other core functions in the middleware which had to be updated by the maintainers. As a result, one-third of the packages available in the previous release (ROS Jade) were not compatible with ROS Kinetic.

By the time this investigation was carried on, another potentially large breaking change would occur with the introduction of ROS2, a complete reengineering of ROS written from

[11] A peripheral contributor who has had exactly one code contribution (i.e. a patch) accepted in that project" [79]

scratch. This new version of the middleware was motivated due to the need for new use cases (*e.g.,* collective of robots, small embedded systems) and new quality constraints (*e.g.,* real-time systems, secure communication mechanisms) that were not available in ROS1. As the website pointed out "ROS 2 will be built as a parallel set of packages that can be installed alongside and inter-operate with ROS 1" [12]. Nonetheless, retro-compatibility was not guaranteed [13].

> [You can use ros1_bridge for making use of not-yet-ported ROS1 packages] but you would need to learn a reasonable amount about packages in ROS1 as well to know how to use them properly. Thus exclusively focusing on ROS2 would leave you without significant functionality. Personally, if I were starting now, I would start with ROS1 and learn about what ROS2 can do without learning how to do it until I encountered something that could be done in ROS2 but not ROS1 (real time control, for example). Then I would learn the ROS2 stuff necessary for that. [...] Viewing ROS1 and ROS2 as an exclusive-or relationship is not a good approach. (ROS Discourse moderator, July 3, 2018 [14])

During this transition period, it was not rare that users may need to use packages from ROS2 when using ROS1 and vice versa. Such breaking changes might have seriously affected all packages already available which is the reason why users choose ROS over other robotic middleware. Nowadays, the two versions of ROS (1 and 2) live in parallel having packages for ROS 1 only, and ROS 2 only, and others compatible with the two versions. We have no evidence about the current impact of this breaking change in the ecosystem but the current status is that the latest stable ROS 1 release was in 2020 (ROS Noetic Ninjemys [15]) and there are newer stable ROS 2 releases (ROS Humble Hawksbill [16]) available and a new one under development (ROS Rolling Ridley [17]).

These breaking changes come from the core ROS developers, and package maintainers have to be aware of such deep changes in technology and workflows. These new technologies and workflows represent new barriers to entry for the current maintainers. Packages which do not adapt rapidly to these changes will be outdated in the short term and some of them could be even abandoned by maintainers who do not have the time or the will to update their packages. This situation is particularly dangerous for already abandoned packages that will not be adapted to ROS2.

Maintainers interested in continuing their projects will have to react. In such a case, it is possible that any of their dependencies could be maintained by a developer who is not interested in upgrading their package. In our survey, we saw that a noticeable portion of responses (66%) pointed out this situation: they were not able to reuse a package because

[12]Why not just enhance ROS 1, ROS 2.0 Design: 2018. `https://design.ros2.org/articles/why_ros2.html#why-not-just-enhance-ros-1`.

[13]Topic: "Discussion on ROS to ROS2 transition plan", ROS Discourse: 2018. `https://discourse.ros.org/t/discussion-on-ros-to-ros2-transition-plan`.

[14]Topic: "ROS1 or 2 for a newbie?", ROS Discourse: 2018. `https://discourse.ros.org/t/ros1-or-2-for-a-newbie/`.

[15]ROS Noetic Ninjemys: 2020. `https://wiki.ros.org/noetic`.

[16]ROS Humble Hawksbill: 2022. `https://docs.ros.org/en/humble`.

[17]ROS Rolling Ridley: 2022. `https://docs.ros.org/en/rolling`.

the dependency was outdated. A smaller amount of responses (16%) pointed out another scenario where the user is stuck in an older version of ROS due to reusing an outdated package then they cannot use packages that only work with newer versions of ROS. Concerns like these were exposed in the work of García *et al.* [66] in which users from the industry side are reluctant to adapt their dependencies to ROS2.

A study [95] from Bogart *et al.* delves into the topic of how different communities deal with breaking changes (Eclipse, R/CRAN, Node.js/NPM ecosystems). The Eclipse community prioritizes backward compatibility. The community R/CRAN wants to ensure that installing and updating packages is easy for their users. In the Node.js/NPM community, maintainers should be able to easily and quickly install and publish their packages. The Eclipse strategy seems to suit better in the ROS context considering the discussed bottlenecks in collaboration. In particular, Eclipse developers provide two APIs: a newer and more unstable for experts and another more stable for regular users. In other sub-communities of R, maintainers keep continuous and frequent communication with users, especially in case of changes in the API of relevant packages. In three of the ecosystems studied by Bogart *et al.* the release and migration activities are supported by automated mechanisms providing informative output for users and maintainers. This could also be profitable for the ROS community.

5.3.5 *Recommendation 5:* Motivate and encourage community contributions

According to some of our surveyed participants, there is a perception that in *ROS Answers* questions are answered after a long time being posted or not answered at all. They claim that "There is no point in doing so since people never answer". There is the risk that late answers can become obsolete or not needed anymore. This situation represents a strong barrier to entry for newcomers [78]. Although *Recommendation 3* may help in finding potential answerers or participants in a question discussion this does not guarantee that the community member will actually participate. In addition to this, Kolak *et al.* [3] reports that the growth of the members of the expertise-focused working groups in ROS is getting slower. A complementary approach considering motivation should take place.

Gamification is one of the approaches that could help to encourage participation. This approach has been successful in many software development activities [96, 97]. *ROS Answers* follows the same format as *StackOverflow* in terms of giving points to users by their participation in the platform, such participation increases their *karma* points. Users can also gain badges for specific contribution achievements, for example, *ROS Answers* provides two badges: *City Duty Badge* for voting up/down 100 different questions or *City Patrol Badge* which is given for the first flagged post. The full list of badges and the way they are obtained is available online[18]. These gamification techniques were effective in *StackOverflow* [98, 99, 100] so we can expect the same success in *ROS Answers*.

In light of the insight of our field study, we think that richer participation should be encouraged. For example, following up on questions or providing useful context information to existing questions. We propose to use those badges or similar techniques in other communication channels in the ROS community. ROS or robotics-specific badges could also be

[18]ROS Answers Badges: 2021. `https://answers.ros.org/badges/`.

helpful as incentives for enriching users' profiles. For example, in the case of a bug in a popular package, specific badges could encourage participation in micro-tasks related to this bug, such as replicating the bug in different robots or in different ROS distributions.

There also could be recognition for outstanding contributors. Public recognition on the website or in the ROSCon could be attractive incentives [101, 102].

Avelino *et al.* points out that newcomers are crucial for projects at risk of being abandoned to survive [103]. Events are a good opportunity for newcomers to know projects of their interest by involving in their development. The ROS community could concentrate the participation of newcomers in special events such as the *Hackfest* [19] of the GNOME desktop environment project. Similarly, in other communities such as Jenkins or WordPress, there are initiatives such as "Adopt a plugin" [20] that invites developers to take charge of an abandoned plugin of the ecosystem. In all these cases we consider them as passive approaches due to possible contributors coming across an announcement of contribution. In *Recommendation 1* and *3* we consider them as active approaches.

5.4 Conclusions

This chapter delves into the results of the field study presented in Chapter 4. We condensed the findings regarding problems in package reuse, contribution dynamics, and knowledge sharing into five common contribution bottlenecks:

B1. *Lack Of Time*: Nearly 60% of surveyed participants claim that time is the main barrier to contributing in some way to the ecosystem.

B2. *Package Abandonment*: 90% of our surveyed participants have faced problems reusing a package realizing that the package maintainer is gone.

B3. *Lack of confidence in the value/quality of a contribution*: *Beginners* and *Intermediate users* have doubts about contributing regarding their contribution is valuable to the community.

B4. *The contribution could be too specific to my problem, domain, hardware or research problem*: 31% of surveyed participants (mostly *Intermediate users* and *Advanced users*) do not contribute because their contribution may not be of interests to the community.

B5. *Workflow is unknown or not clear*: Some surveyed participants do not know the guidelines to properly share their contributions.

We deepen into the bottleneck of Package Abandonment which is frequently faced by 63% of our surveyed participants. This situation affects the reliability of the ROS ecosystem and it cannot be foreseen by users. It also impacts the contribution flow of projects as the maintainer misses bug reports, bug fixes, and other valuable contributions that cannot be reviewed or integrated. As *Advanced users* and *Intermediate users* do rely on 3rd party ROS packages this represents a serious threat to their projects. Although our interviewees presented two workarounds for handling this issue (*ROSify*-ing packages or forking repositories) they may

[19] Hackfests in the GNOME-related world: 2020. `https://wiki.gnome.org/Hackfests`.

[20] Adopt a Plugin - Jenkins - Jenkins wiki: 2021. `https://www.jenkins.io/doc/developer/plugin-governance/adopt-a-plugin/`.

solve the issue individually and for a short term. More robust and long-term solutions are needed for handling abandoned packages in a manner that does not undermine the reliability of the ROS ecosystem.

We provide a guideline with 5 recommendations to overcome the presented bottlenecks:

R1. *Identify and Predict Abandoned Packages*: an automated approach to identify abandoned packages is needed and also to predict the risk of packages being abandoned any time soon.

R2. *Provide an Informative Package Repository*: an informative package catalog automatically updated is recommended for users to know the installation and configuration guidelines, abandonment status of the package, and troubleshooting instructions among other relevant data to make an informed decision about choosing a package to reuse or not.

R3. *Recommend contribution opportunities to qualified community members*: recommender systems may be useful for recommending a micro-task to a community member to maintain an abandoned package or for answering a question in *ROS Answers*.

R4. *Limit breaking changes*: we propose an automated approach for releases and migration activities for a less dramatic transition when there are changes in the API of packages or the middleware.

R5. *Motivate and encourage community contributions*: ROS ecosystem may take ideas from other communities such as giving public recognition to outstanding contributors, arrange events like *hack fests* or campaigns for adopting abandoned packages, to motivate the participation of the community.

Although these recommendations are focused on the ROS cases some of them may be applicable for other ecosystems as well.

Chapter 6

Distributing Community Efforts in Knowledge Sharing Platforms: the case of ROS Answers

In this chapter we present a study of an Expert Recommender System for answerers in the Question & Answer community *ROS Answers*. We first develop the motivation and the objective of the study in Section 6.1. In Section 6.2, we present in detail the platform *ROS Answers*. Afterwards, in Section 6.3, we show the related work, giving special attention to the study of X. Zhang on which this work is based. After that, in Section 6.4, we present how we adapted Zhang's approach and the constraints we imposed to perform the experiment. In the evaluation methodology (Section 6.5) we introduce the three research questions to be answered in the study. Once these aspects are presented, we show the answers to the given research questions. Later, we discuss the results (Sections 6.6, 6.7 and 6.8) and present the threats to validity.

6.1 Motivation

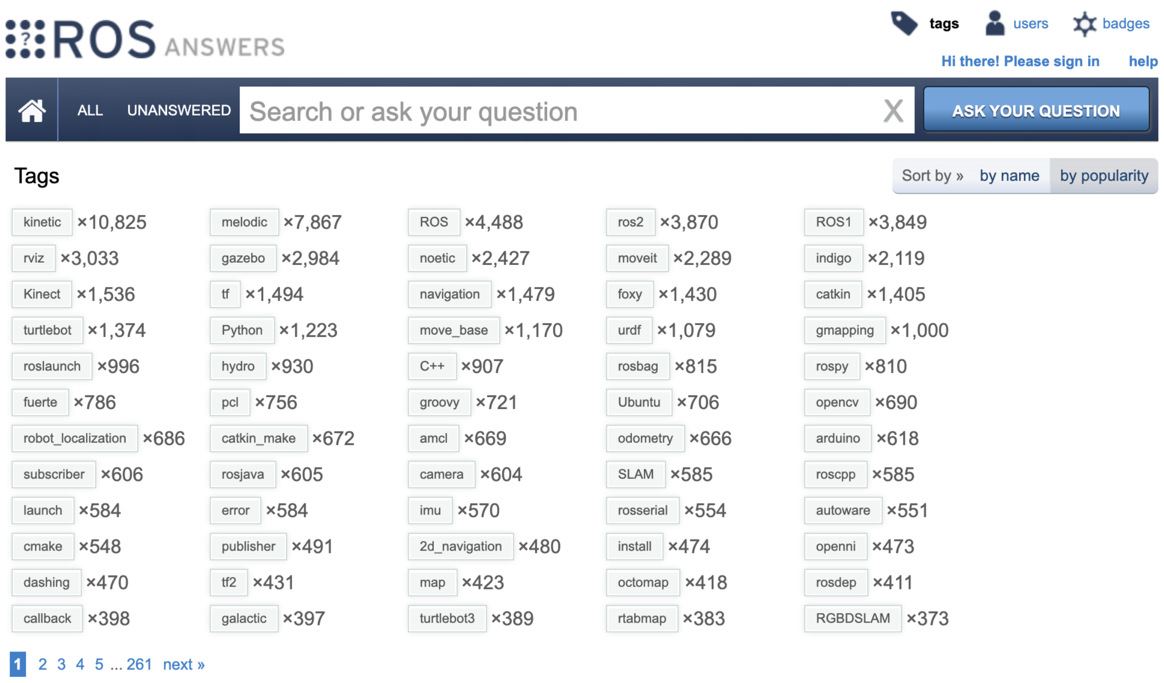
Question and Answer (Q&A) sites are used by the technical community as a way of sharing knowledge about programming languages, coding practices, bug workarounds, etc. With over 10 million users and 16 million questions, the Q&A site Stack Overflow is often the first choice for users trying to figure out how to use an API or library. Stack Overflow has successfully used gamification to improve the quality of questions and answers, assigning reputation points and badges based on participation [104]. More specialized communities can have their own Q&A sites, allowing the members to discuss and answer community-specific questions.

Such is the case of ROS Answers[1], the Q&A site for the Robot Operating System (ROS) project. ROS Answers is quite active, with over 80.000 users and 63.733 questions[2]. Like

[1]ROS Answers, Website, `https://answers.ros.org`

[2]Data by February 27th, 2022

Stack Overflow, badges and *karma* points to reward participation. Users earn *karma* points when other users up-vote their questions and answers. Unfortunately, the ratio of unanswered questions seems to be growing over time: from approximately 32.5% in September 2019 to approximately 35% in May 2020, to 40.8% in Feb. 2022. This is also slightly lower compared to popular sub-communities on *Stack Overflow* (e.g., below 27% for the C#, R, Ruby-on-rails, and Python sub-communities). Questions go unanswered for a variety of reasons. On the one hand, some questions are highly specialized, as there have been 8 stable ROS distribution releases over the past 10 years, each with its own set of packages and issues, supporting over 100 different types of robots. On the other hand, finding relevant questions to answer is not easy. Users can search for questions by tag, which are manually picked by question authors. However, these tags can be quite generic. For example, the first and third most popular question tags on ROS Answers (see Figure 6.1) are `kinetic` and `melodic`, two ROS releases (and the third most popular tag is `ROS`[3]).



Tags

kinetic ×10,825	melodic ×7,867	ROS ×4,488	ros2 ×3,870	ROS1 ×3,849
rviz ×3,033	gazebo ×2,984	noetic ×2,427	moveit ×2,289	indigo ×2,119
Kinect ×1,536	tf ×1,494	navigation ×1,479	foxy ×1,430	catkin ×1,405
turtlebot ×1,374	Python ×1,223	move_base ×1,170	urdf ×1,079	gmapping ×1,000
roslaunch ×996	hydro ×930	C++ ×907	rosbag ×815	rospy ×810
fuerte ×786	pcl ×756	groovy ×721	Ubuntu ×706	opencv ×690
robot_localization ×686	catkin_make ×672	amcl ×669	odometry ×666	arduino ×618
subscriber ×606	rosjava ×605	camera ×604	SLAM ×585	roscpp ×585
launch ×584	error ×584	imu ×570	rosserial ×554	autoware ×551
cmake ×548	publisher ×491	2d_navigation ×480	install ×474	openni ×473
dashing ×470	tf2 ×431	map ×423	octomap ×418	rosdep ×411
callback ×398	galactic ×397	turtlebot3 ×389	rtabmap ×383	RGBDSLAM ×373

Figure 6.1: Tags in ROS Answers

Given that users may need to wait a long time for a satisfactory answer [105], there is a barrier to entry for beginners, who may feel ignored if their questions go unanswered, but who also may lack the confidence to answer questions that they do have the expertise to answer [75]. Specialized knowledge can also be lost if experts on certain topics leave the community [106]. Most Q&A sites rely on highly-participating users to answer questions and curate answers. However, in smaller communities like ROS, this puts a higher workload on already busy community members, and tasks like answering questions are seen as low-priority [75]. Inevitably, these issues end up affecting the health of the ROS community, since specialized knowledge is not being cataloged, and may be lost as community members leave.

[3]ROS Answers Tag's (2022), Website, `https://answers.ros.org/tags/`

Objective

Expert Recommender Systems (ERS) focus on ranking the most capable candidates in a group of individuals to reach a desired objective or goal [107, 5]. In the particular case of ERS for Q&A communities, the intention is to find the best candidate to answer a target question. Considering the bottlenecks such as "*lack of time*", "*package abandonment*" and "*the contribution could be too specific to my problem*", we think that the unbalanced workload can be lessened by distributing the question-answering task among the community. Specifically targeting less active community members that are just as qualified to answer open questions. As such, our objective is not to find the best candidate but to instead find sufficiently qualified candidates who are also not as active on ROS Answers.

6.2 ROS Answers

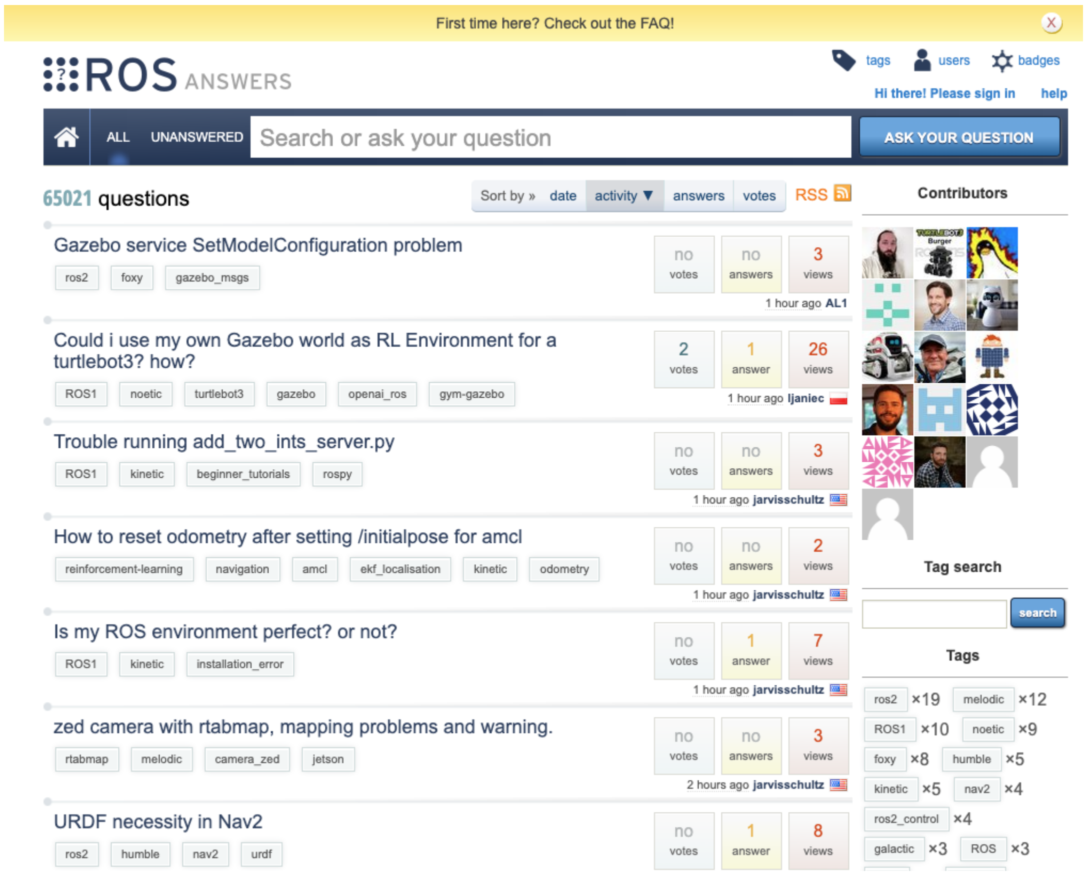
As was presented before, *ROS Answers* is the web platform where users can publish questions related to ROS and robotics, and also provide answers to these questions. According to the results of our survey (Section 4.2.4) ROS Answers is one of the first sources of information for troubleshooting by ROS users.

The topics of questions are varied but they are related to ROS versions, ROS usage, ROS libraries, robots, or any hardware that uses ROS, etc. Questions also can be related to the disciplines under robotics: software engineering, mechanics, electronics, or math, in the context of ROS usage. These topics of a question are represented by *tags*, a word that is linked by the user who asks the question. A question can have 1 or more tags that describe its content.

On the one hand, as we can see in Figure 6.3, the structure of a question consists of a title, a body text, a series of tags, and an author. A question can be commented by any user for clarification. Questions also have evaluation points (the zero with the up and down arrows) given by users of certain experience in the platform, they are commonly called *upvotes*.

On the other hand, the structure of an answer (Figure 6.4) is a text body with the answer. Answers as well as questions' bodies can contain pieces of code or snippets to ease reproducing an error, for example. Answers can also be commented by any user as a follow-up. Another important issue is the upvotes. Users can up/down vote for the quality or precision of a given answer. This is relevant because more voted questions are shown first from top to down. Also, the number of votes gives a sense of quality and confidence because it was approved by many people. An answer also has a mark of correct (the check symbol below the up/down votes) which is given by the author of the question and it means that that answer is the correct one or the one that fully answers the question.

ROS Answers has gamification features. Users can earn badges through the use of the platform and their participation by asking, answering, up/down voting, and commenting. There are a variety of badges regarding different actions, for example, there is a badge named *Civic duty* for those who up/down voted more than 100 times, *Famous question* for those who posted a question with 50 views or more, or *Critic* when a user down vote for the first time. In addition to badges, users gain points for their actions on the platform.


First time here? Check out the FAQ!

ROS ANSWERS

tags users badges

Hi there! Please sign in help

ALL UNANSWERED Search or ask your question ASK YOUR QUESTION

65021 questions

Sort by » date activity ▼ answers votes RSS

Gazebo service SetModelConfiguration problem
ros2 foxy gazebo_msgs
no votes no answers 3 views
1 hour ago AL1

Could i use my own Gazebo world as RL Environment for a turtlebot3? how?
ROS1 noetic turtlebot3 gazebo openai_ros gym-gazebo
2 votes 1 answer 26 views
1 hour ago ljaniec

Trouble running add_two_ints_server.py
ROS1 kinetic beginner_tutorials rospy
no votes no answers 3 views
1 hour ago jarvisschultz

How to reset odometry after setting /initialpose for amcl
reinforcement-learning navigation amcl ekf_localisation kinetic odometry
no votes no answers 2 views
1 hour ago jarvisschultz

Is my ROS environment perfect? or not?
ROS1 kinetic installation_error
no votes 1 answer 7 views
1 hour ago jarvisschultz

zed camera with rtabmap, mapping problems and warning.
rtabmap melodic camera_zed jetson
no votes no answers 3 views
2 hours ago jarvisschultz

URDF necessity in Nav2
ros2 humble nav2 urdf
no votes 1 answer 8 views

Contributors

Tag search

search

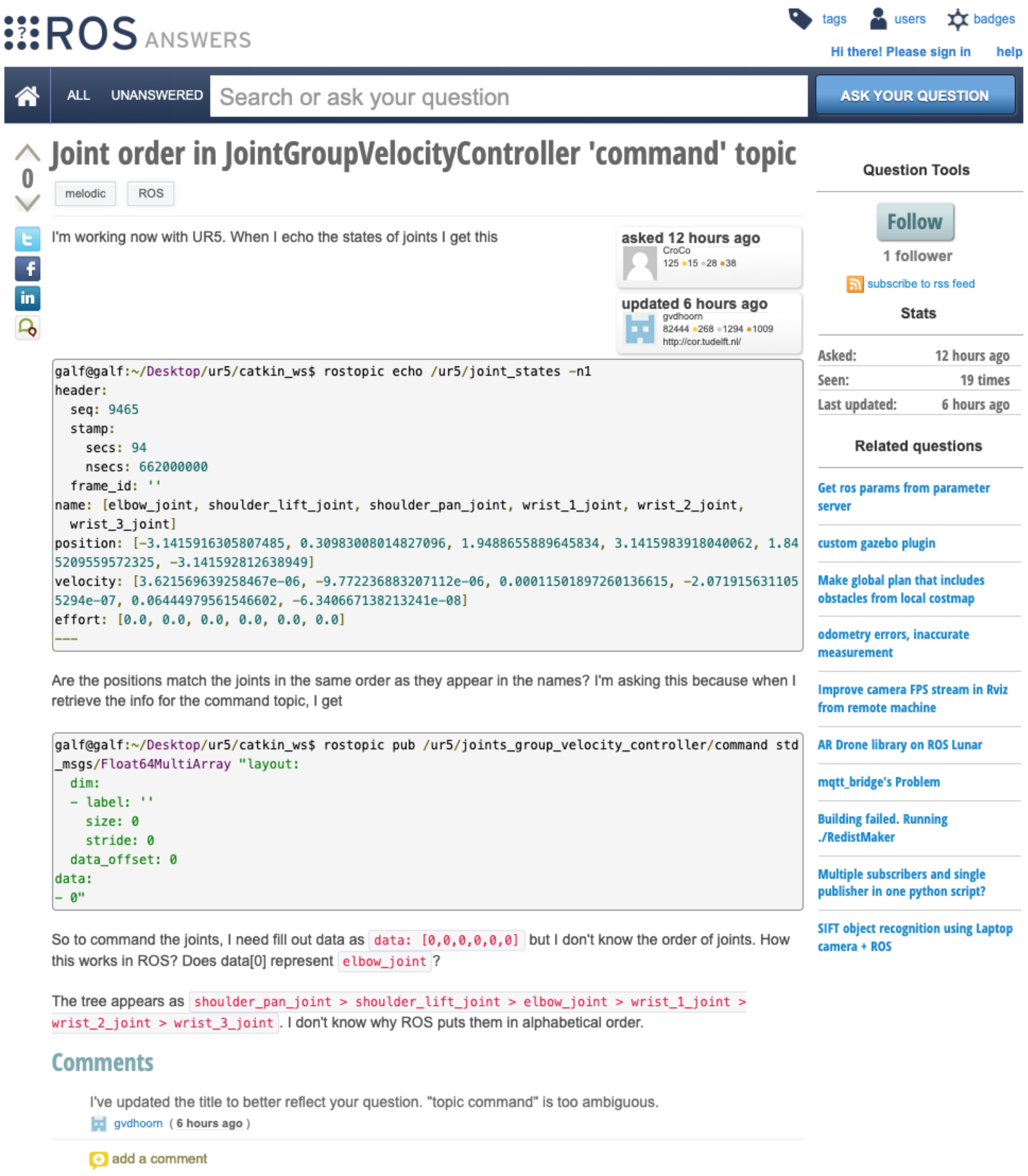
Tags

ros2 ×19 melodic ×12
ROS1 ×10 noetic ×9
foxy ×8 humble ×5
kinetic ×5 nav2 ×4
ros2_control ×4
galactic ×3 ROS ×3

Figure 6.2: Screenshot of the main page of ROS Answers

There is also a system of points, named *Karma points* that depicts the degree of tenure in ROS Answers and may be a good indicator of the expertise in ROS usage. Users gain points when a question or an answer posted by them is *upvoted*. Users lose *Karma* points if their contributions to ROS Answers are down-voted. The Karma points also give permissions and give the right to perform a variety of moderation tasks.

The distribution of *Karma points* follows a Power Law distribution, this is, a very very small amount of users have a large number of points while the big majority has few points. As we can see in Figure 6.5, the distribution in scale log-log depicts a clear line: this confirms the Power Law like distribution.


ROS ANSWERS

tags users badges

Hi there! Please sign in help

ALL UNANSWERED Search or ask your question ASK YOUR QUESTION

Joint order in JointGroupVelocityController 'command' topic

0

melodic ROS

I'm working now with UR5. When I echo the states of joints I get this

asked 12 hours ago
CroCo
125 ●15 ●28 ●38

updated 6 hours ago
gvdhoorn
82444 ●268 ●1294 ●1009
http://cor.tudelft.nl/

```
galf@galf:~/Desktop/ur5/catkin_ws$ rostopic echo /ur5/joint_states -n1
header:
  seq: 9465
  stamp:
    secs: 94
    nsecs: 662000000
  frame_id: ''
name: [elbow_joint, shoulder_lift_joint, shoulder_pan_joint, wrist_1_joint, wrist_2_joint,
  wrist_3_joint]
position: [-3.1415916305807485, 0.30983008014827096, 1.9488655889645834, 3.1415983918040062, 1.84
5209559572325, -3.141592812638949]
velocity: [3.621569639258467e-06, -9.772236883207112e-06, 0.00011501897260136615, -2.071915631105
5294e-07, 0.06444979561546602, -6.340667138213241e-08]
effort: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
---
```

Are the positions match the joints in the same order as they appear in the names? I'm asking this because when I retrieve the info for the command topic, I get

```
galf@galf:~/Desktop/ur5/catkin_ws$ rostopic pub /ur5/joints_group_velocity_controller/command std
_msgs/Float64MultiArray "layout:
  dim:
  - label: ''
    size: 0
    stride: 0
  data_offset: 0
data:
- 0"
```

So to command the joints, I need fill out data as `data: [0,0,0,0,0,0]` but I don't know the order of joints. How this works in ROS? Does data[0] represent `elbow_joint`?

The tree appears as `shoulder_pan_joint > shoulder_lift_joint > elbow_joint > wrist_1_joint > wrist_2_joint > wrist_3_joint`. I don't know why ROS puts them in alphabetical order.

Comments

I've updated the title to better reflect your question. "topic command" is too ambiguous.
gvdhoorn (6 hours ago)

add a comment

Question Tools

Follow

1 follower

subscribe to rss feed

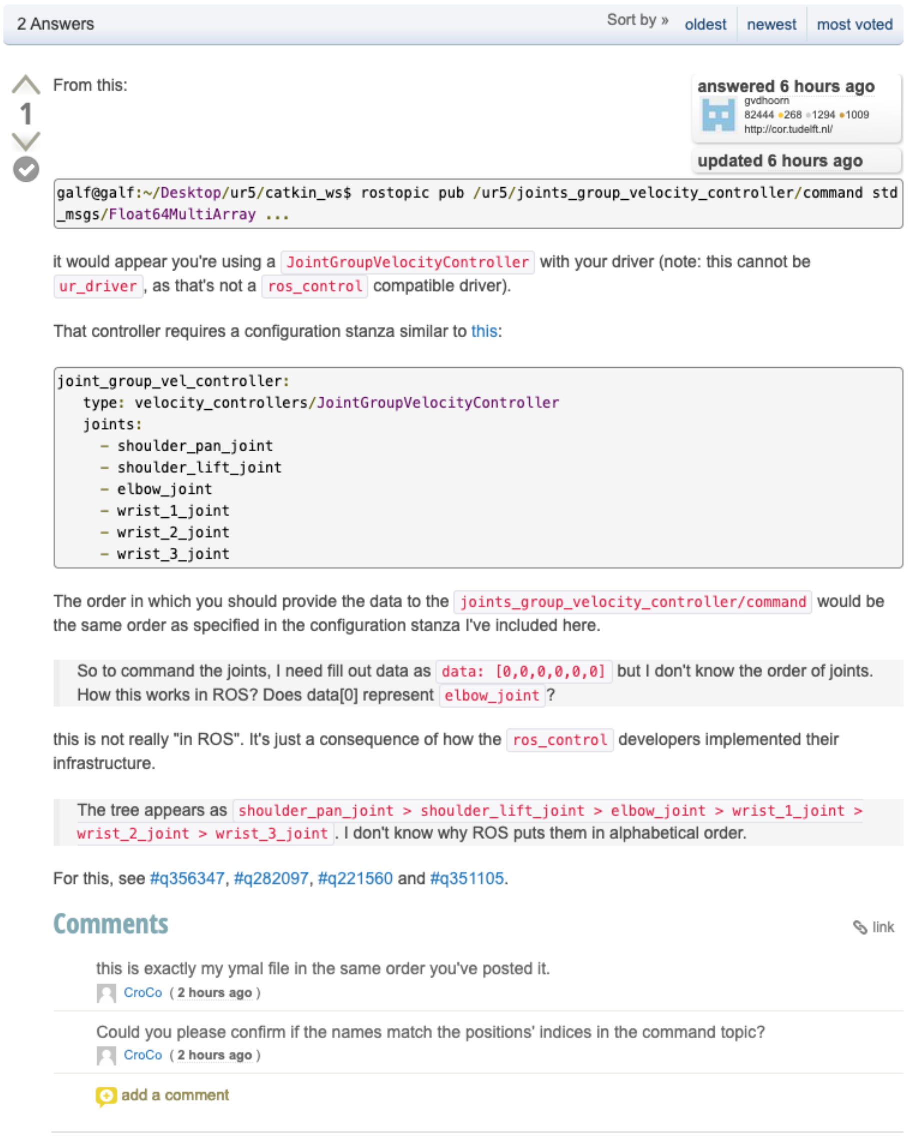
Stats

Asked:	12 hours ago
Seen:	19 times
Last updated:	6 hours ago

Related questions

- Get ros params from parameter server
- custom gazebo plugin
- Make global plan that includes obstacles from local costmap
- odometry errors, inaccurate measurement
- Improve camera FPS stream in Rviz from remote machine
- AR Drone library on ROS Lunar
- mqtt_bridge's Problem
- Building failed. Running ./RedistMaker
- Multiple subscribers and single publisher in one python script?
- SIFT object recognition using Laptop camera + ROS

Figure 6.3: A question in ROS Answers



2 Answers Sort by » oldest newest most voted

1

From this:

answered 6 hours ago
gvdhoorn
82444 •268 •1294 •1009
http://cor.tudelft.nl/

updated 6 hours ago

```
galf@galf:~/Desktop/ur5/catkin_ws$ rostopic pub /ur5/joints_group_velocity_controller/command std
_msgs/Float64MultiArray ...
```

it would appear you're using a `JointGroupVelocityController` with your driver (note: this cannot be `ur_driver`, as that's not a `ros_control` compatible driver).

That controller requires a configuration stanza similar to this:

```
joint_group_vel_controller:
   type: velocity_controllers/JointGroupVelocityController
   joints:
     - shoulder_pan_joint
     - shoulder_lift_joint
     - elbow_joint
     - wrist_1_joint
     - wrist_2_joint
     - wrist_3_joint
```

The order in which you should provide the data to the `joints_group_velocity_controller/command` would be the same order as specified in the configuration stanza I've included here.

> So to command the joints, I need fill out data as `data: [0,0,0,0,0,0]` but I don't know the order of joints. How this works in ROS? Does data[0] represent `elbow_joint`?

this is not really "in ROS". It's just a consequence of how the `ros_control` developers implemented their infrastructure.

> The tree appears as `shoulder_pan_joint > shoulder_lift_joint > elbow_joint > wrist_1_joint > wrist_2_joint > wrist_3_joint`. I don't know why ROS puts them in alphabetical order.

For this, see #q356347, #q282097, #q221560 and #q351105.

Comments

link

this is exactly my ymal file in the same order you've posted it.
CroCo (2 hours ago)

Could you please confirm if the names match the positions' indices in the command topic?
CroCo (2 hours ago)

add a comment

Figure 6.4: An answer in ROS Answers on May 15th 2022

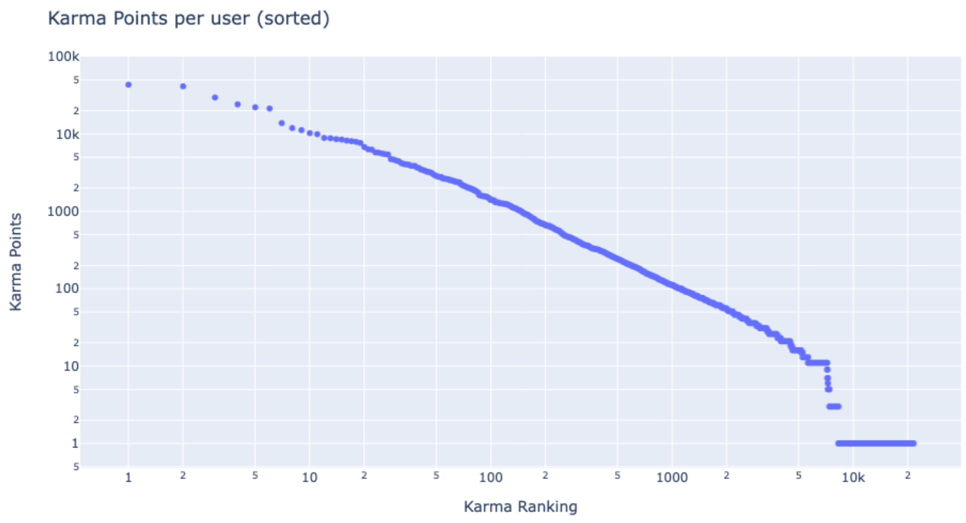


Figure 6.5: Distribution of Karma Points among users (log-log)

6.3 Related Work

Finding experts that can answer questions on Q&A sites has been widely studied by several authors which propose different strategies to address this challenge [5, 108, 109, 110]. The general idea is to bring questions to the attention of possible answerers, using Expert Recommender Systems based on simple metrics, language, and topic models, the topology of the user/question graph, as well as machine learning methods, among others.

Most of the recent work on recommendations for Stack Overflow is based on collaborative filtering methods [93], a technique that can filter out relevant items that a user might prefer on the basis of preferences by similar users. For example, Neshati *et al.* [107] propose a learning framework that extracts features from answerers' current performance on Stack Overflow, based on topic similarity, emerging topics, user behavior, and topic transitions and tries to predict questions that they may be able to answer in future. There are many other approaches Sumanth *et al.* [111] use Page-Rank and HITS algorithm to discover experts in certain subdomains (Java, JavaScript, C# and Python) of the StackOverflow. They found that some users are experts in more than one domain. The correctness of the result was used Discounted Cumulative Gain (DCG). Zhao *et al.* [112] using a dataset of the Quora[4] Q&A website propose a method from the viewpoint of learning ranking metric embedding. They compare it against top state-of-the-art methods such as AuthorityRank, TSPM, DRM, GRMC-AGM, and DeepWalk, obtaining better results than the latter (nDCG, Precision@1, and Accuracy).

An analysis of the behavior of expert and non-expert users was performed by Dana and Yair Movshovitz-Attias *et al.* [113]. They used a dataset of the activity of StackOverflow from 2008 to 2012. In their analysis, they found that expert users tend to ask more questions than non-experts and their answers are more prone to be accepted. They also found that the initial behavior of a user in the platform predicts their long-term participation. László Tóth *et al.*

[4]`http://www.quora.com`

propose [114] a novel approach for quality classification of questions from StackOverflow that combines natural language processing and deep learning, using the linguistic and semantic features of a question. It was applied on a dataset of 110.547 questions of StackOverflow obtaining a precision of 75% and a recall of 74%.

Alharthi *et al.* analysed [115] the relationships of sixteen different features of questions (from StackOverflow) and their relationships with their score. They applied Spearman's correlation of scores and question features on over 12.000 questions. They found that number of answers, favorites, and views have a strong relationship with the score of a question. Wang L. *et al.* worked on a personalized recommendation of new questions from StackOverflow [116]. Their approach considers both topic modeling and link structure. They propose a boosted version of the HITS algorithm named NEWHITS to link questions and user profiles. Their method is compared with other state-of-the-art algorithms (HITS, PageRank, LDA-Based, among others) and shown to improve HITS significantly.

In addition to Questions & Answers platforms, certain approaches include the technical skills of the expert candidates by seizing their GitHub history. Huang *et al.* created CPDScorer [117], a novel approach for modeling and scoring the skills of a developer. It uses data from GitHub such as source code quality (cyclomatic complexity, number of functions, among others) and from StackOverflow the quality of the answers that a developer provided (answer length, comment count, number of up/down votes). They present an expertise score which is a weighted sum of both aspects. Constantinou *et al.* [118] identify the developer's skills by using their data from GitHub and StackOverflow. They measured their commit activity and the files involved in those commits as well as their continuity over time. This data was contrasted against their language programming preferences in their StackOverflow profile. Xiong *et al.* show a novel approach to mining developer behaviors across GitHub and StackOverflow [119]. They used CART decision trees through usernames, user behaviors, and writing styles. After that, they explore the behaviors through both a T-graph analysis and LDA-based topic clustering of tags from these websites. They found that active issue committers are also active question answerers and their interests in terms of tags is correlated in both web platforms.

Yang *et al.* [120] claims that tags are more representative than the content(partially agreed by Calefato *et al.* [121]. Following that idea, they propose a tag-based expert recommendation system using a user/tag matrix modeled through a Gaussian distribution of a certain user/tag. They tested their approach using a dataset of 14986 tags and 1425 users and its performance is better than the state-of-the-art at that time (2014). Vadlamani *et al.* conducted a qualitative study about software development expertise to understand the reasons for participation in GitHub or StackOverflow [122]. They applied a survey with open-ended and close-ended questions and sent it to experts found on GitHub and StackOverflow. The analysis was performed through an open coding process with a result of over 40 categories from more than 320 quotes. Among their findings are that JavaScript and C/C++ are leading programming language skills, and that software development experts would have not only technical but also soft skills. Another analysis across GitHub and StackOverflow is performed by Vasilescu *et al.* [123]. They used a dataset of 46.967 users belonging to those platforms. Among their findings are that active GitHub committers tend to ask fewer questions but provide more answers. The other way around, more active StackOverflow answerers tend to make more commits in GitHub. Actually, for active committers to ask questions in StackOverflow catalyzes committing in GitHub and vice versa.

Unlike the aforementioned contributions, Liu *et al.* [124] seizes the data from Zhihu.com, the largest CQA website in China. They propose two expert recommendation systems based on graph convolution neural network (GCN): GCN-Doc which uses Doc2Vect for extracting text features before training, and GCN-Lstm with a long short-term memory network which extracts these features while training.

6.4 Approach

In this section, we introduce the model on which we are based: Zhang *et al.* [125] approach. Afterward, we present our adaptation to their method.

6.4.1 Zhang's *et al.* Expert Recommendation System

Among the variety of approaches of Expert Recommendation Systems, we chose the Contributor Recommendation Approach for Open Source Projects, *ConRec*, developed by Zhang *et al.* [125]. As *ConRec* leverages the developer's historical activity and also their technical interests we thought it would be applicable for our context. The intuition behind their approach is that in social coding, developers tend to form networks of groups of collaboration and these networks affect directly the choice of participation of other developers [125, 126, 127].

Their approach is based on two algorithms: *Weighed Collaborative Filtering Algorithm* and *Text Matching Based Recommendation Algorithm*: The first one filters out developers that have no experience with the target project's programming language, then it calculates the degree of relationship between candidates and the current developers of the target project, by inspecting the activity on the projects that they have participated in common. They define these relationships using these equations:

$$R_{\mathrm{d}p}(\mathrm{d},p) = \frac{Commit(\mathrm{d},p)}{\sum_{\mathrm{i}=1}^{|U_p|} Commit(U_p[\mathrm{i}],p)} \tag{6.1}$$

Equation 6.1 represents the relationship between a developer (d) and a project(p), U_p is the universe of pre-existing developers.

The relationship between developers (Equation 6.2) is represented by using the Vector Space similarity algorithm, and P_A is the projects where the developer A has committed to:

$$R_{\mathrm{dd}}(A,B) = \frac{\sum p:\{P_A \cap P_B\} R_{\mathrm{dp}}(A,p) * R_{\mathrm{dp}}(B,p)}{\sqrt{\sum p : P_A R_{\mathrm{dp}}^2(A,p) * \sum p : P_B R_{\mathrm{dp}}^2(B,p)}} \tag{6.2}$$

Equation 6.3 calculates the relation between developers (D_p denotes the pre-existing developers of a project p).

$$result(A, p) = \sum_{\mathrm{d}:D_p} R_{\mathrm{d}p}(\mathrm{d}, p) * R_{\mathrm{dd}}(A, \mathrm{d}) \tag{6.3}$$

Finally the results are ranked by the value of the $result(A, p)$ with all potential developers of a project p.

The second one attempts to complement the previous one for such cases in which there is a lack of previous relationships among candidates and developers of the target project. It scores users based on the relationship of the candidates with the technical terms of the target project. They calculate the relation between terms and developers using a TF-IDF algorithm [128]. This relationship is depicted by the Equation 6.4 where d is the developer and t a term, and P_t all the projects with term t, T is the entire set of terms.

$$R_{\mathrm{dt}}(\mathrm{d}, t) = \sum_{p:P_t} R_{\mathrm{d}p}(\mathrm{d}, p) * log\frac{|\bigcup_{x:T} P_x|}{P_t} \tag{6.4}$$

The score calculation is in the Equation 6.5, where $T_{\mathrm{d}p}$ denotes the set of terms that developer d and project p have in common:

$$result(\mathrm{d}, p) = |T_{\mathrm{d}p}| \sum_{t:T_{\mathrm{d}p}} R_{\mathrm{dt}}(\mathrm{d}, t) \tag{6.5}$$

6.4.2 Adapting Text Matching Based Recommendation Algorithm to the Tag-Map Based Algorithm

Zhang *et al.* [125] proposes the model *ConRec* which provides a ranking of potential contributors across the open source community for given projects. As they describe their approach, it leverages the developer's historical activities so they can match their technical interests and experience. Technically, *ConRec* combines collaborative filtering with text matching for ranking candidates. The text-matching approach reuses the relationship between a developer and the projects they have participated in but also considers technical terms associated with the target project. This approach seemed appropriate to be adapted in the Q&A context due to the use of historical activity and the interests of the candidates for, our this case, answering a question.

We adapted their text-matching approach, designing our *Tag-Map Based Algorithm*[5]. Instead of matching a developer for a project we try to find a good answerer to a certain question. The technical terms, used in *ConRec*, in our case are the *tags* declared for a question. The equivalence of concepts used by Zhang *et al.* and our approach is shown in the Table 6.1.

Let U be the set of all users registered in ROS Answers, Q the set of all questions posted in ROS Answers. $Activity(u, q)$ is the count of participation of u in q considering comments

[5]In the following we call it: *TMBA*

Table 6.1: Adaptation of our approach to Zhang's *et al.*

Zhang's *et al.* approach	**Our approach**
Project	Question
Developer Candidate	Answerer Candidate
Project's Term	Question's Tag
Target Project's Developers	Question's Author
Relationship Developer-Project ($R_{\mathrm{d}p}$)	Relationship User-Question (R_{uq})
Relationship Developer-Term ($R_{\mathrm{d}t}$)	Relationship User-Tag (R_{ut})

and answers plus 1 if the user is the author (*e.g.,* an author who also commented twice and provided one answer will have an activity count of 2+1+1=4, an author that only posted a question will have an activity count of 1) and U_q is the set of all users that participated in the question's thread: author, answerers and commenters. We define the relationship between a user $u \in U$ and a question $q \in Q$ as:

$$R_{uq}(u,q) = \frac{Activity(u,q)}{\sum_{\mathrm{i}=1}^{|U_q|} Activity(U_q[\mathrm{i}],q)} \tag{6.6}$$

In equation 6.6, we can see that $\sum_{\mathrm{i}=1}^{|U_q|} Activity(U_q[\mathrm{i}],q)$ is the sum of comments, answers + 1.

Let T be the corpus of all tags defined in ROS Answers, Q_t the set of all questions tagged under the t tag, and T_{uq} the set of all tags that match the user u and the question q.

We define the relationship between a user u and a tag t as:

$$R_{ut}(u,t) = \sum_{q \in Q_t} R_{uq}(u,q) \cdot log \frac{|\bigcup_{x \in T} Q_x|}{|Q_t|} \tag{6.7}$$

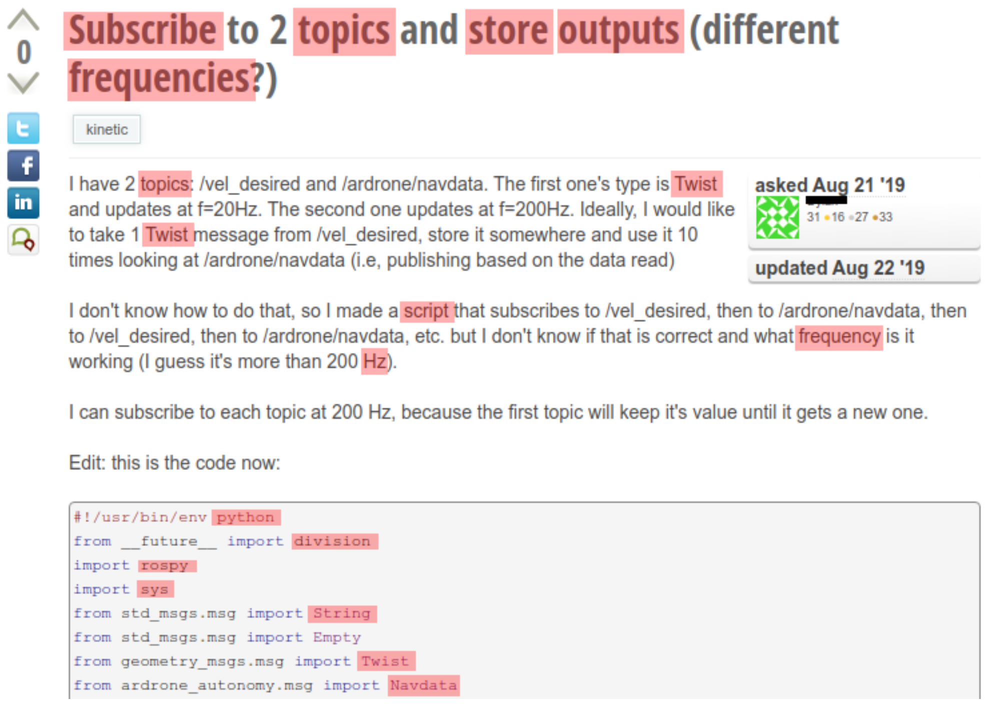
Finally, having T_{uq} as the set of tags that are present in question q and user u, the score for a candidate u for participating in a question q is:

$$tmba\_score(u,q) = |T_{uq}| \cdot \sum_{t \in T_{uq}} R_{ut}(u,t) \tag{6.8}$$

The algorithm then, for a target question will have to calculate and rank the `TMBA_score` for all the users in *ROS Answers*. We can see that the algorithm is, for the score of a question $q \in Q$, a user $u \in U$ and a tag $t \in T_{uq}$, $O(|T_{uq}| \times |Q_t| \times |U_q|) = K_{uqt}$. To get the matrix of all the scores for all questions against all users it would be $O(|U| \times |Q| \times K_{uqt})$.

6.4.3 Tag Extension

Nearly 80% of questions in *ROS Answers* have up to 4 tags (the mean is 3.2 and the median is 3 tags). We have seen that sometimes the body of the question or even the title is very



0

Subscribe to 2 topics and store outputs (different frequencies?)

kinetic

I have 2 topics: /vel_desired and /ardrone/navdata. The first one's type is Twist and updates at f=20Hz. The second one updates at f=200Hz. Ideally, I would like to take 1 Twist message from /vel_desired, store it somewhere and use it 10 times looking at /ardrone/navdata (i.e, publishing based on the data read)

asked Aug 21 '19

[redacted]

31 •16 •27 •33

updated Aug 22 '19

I don't know how to do that, so I made a script that subscribes to /vel_desired, then to /ardrone/navdata, then to /vel_desired, then to /ardrone/navdata, etc. but I don't know if that is correct and what frequency is it working (I guess it's more than 200 Hz).

I can subscribe to each topic at 200 Hz, because the first topic will keep it's value until it gets a new one.

Edit: this is the code now:

```
#!/usr/bin/env python
from __future__ import division
import rospy
import sys
from std_msgs.msg import String
from std_msgs.msg import Empty
from geometry_msgs.msg import Twist
from ardrone_autonomy.msg import Navdata
```

Figure 6.6: Example of extending tags identifying existing tags in the title and body of the question. In the image, in red we found existing tags that could be used for a more accurate description of the question. Originally the question has 1 and we can extend it to 15 tags

descriptive in terms of the keywords involved. Keywords are words that are in the corpus of tags but were not manually entered by the question's author. We think that describing the question not only in terms of the tags manually entered, but also by keywords may help to discriminate among the users and increase the recall, *i.e.*, more users qualified to participate in the question thread. An example of this can be seen in Figure 6.6.

The user's tag profile is generated by the tags of the question in which the user has participated. Their profile can also be enriched with extended keywords from questions. Once questions have extended tag representation user's profile can be reprocessed to add those new tags.

The new profile of a question or a user considers previous tags plus the extended keywords. For extending we manually checked all the tags that appear in at least 3 questions (5.141 tags are used in 2 questions or less are ignored) and we added 83 tags to the stop-words list.

Figure 6.7 shows a histogram of the number of tags per question. Here we can see that common questions in *ROS Answers* have few tags (80% has up to 4). Normally the title and body of the questions are more descriptive than just the tags. Considering the rich corpus of tags, we experimented with extending tags for Questions and for Users. It could be interesting to evaluate the performance of the TMBA when we extend the tags of questions but not the users, or extend users and questions against the scenario of no extension of questions or users. This opens 4 possible scenarios of extending tags that could be used to evaluate the TMBA using these tags extensions. These scenarios are:

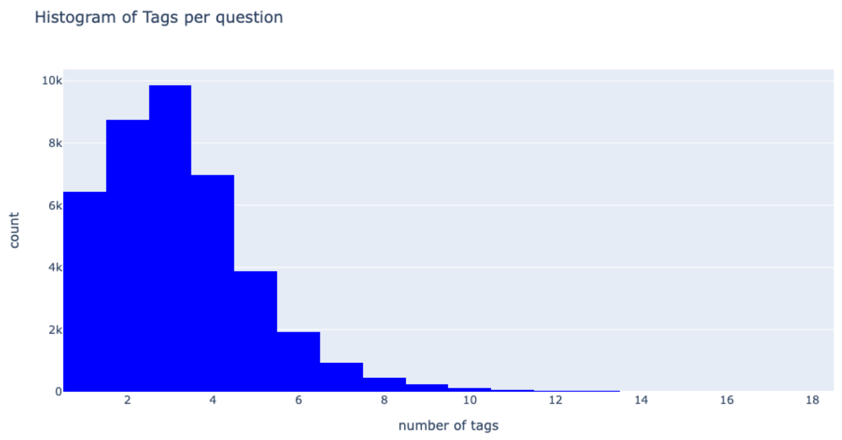


Figure 6.7: Frequency of tags in questions

1. Scenario A (Q U): no tag extension for questions nor users
2. Scenario B (Q+ U): tag extension for questions only
3. Scenario C (Q U+): tag extension for users only
4. Scenario D (Q+ U+): tag extension for questions and users

We evaluated these four scenarios to see if extending tags made any difference.

6.5 Evaluation Methodology

We evaluated *TMBA* to answer three research questions.

RQ1 Does TMBA predict the actual answerer of a question?
RQ2 Does TMBA recommend qualified users?
RQ3 Does TMBA facilitate the workload distribution of answering questions?

The first question aims to measure the cases when the algorithm gave the highest score to the user that actually answered the question and the answer was accepted. For the second one, we used the recall measurement as how many participants of a question are in the first k users in the ranking of scores given by the algorithm. In the third one, we check if the algorithm recommends users that are not the first in the existing Karma ranking so the workload can be distributed among the community. With these questions, we can deliberate whether *TMBA*: (1) recommends a qualified answerer, and (2) promote the participation of less active users.

In the following subsections, we expand on the way we evaluate and answer the research questions

RQ1 -Distribution of Correct Answerers

We first evaluated TMBA by seeing if it recommends the correct answerer and how close to the first positions it is recommended. Recommending the user that provided the correct answer in the first place is the best performance expected by an ERS. We want to see how TMBA performs in this way. A good performance would be a histogram with a high frequency of hits in the first places and a low frequency in further places of the ranking.

To contrast better the performance of TMBA we can visually inspect the Empirical Cumulative Distribution Function. It is a discrete function that is defined at a point x as the proportion of elements in the dataset that are less than or equal to x [129]. Being n the size of the dataset, the formula of such functions is the following:

$$F_n(x) = \frac{\text{number of elements in the dataset } \leq x}{n}$$

The best distribution would be the one whose curve grow faster and approaches to 1 with a high value in the Y axis.

RQ2 -Recall@k

We define a participant of a question as any user that participates in the thread of a question, this means the author and answerers. If a question has 2 participants, it means that only one user provided an answer to that question.

Common rank-based evaluation metrics in Expert Recommender Systems are: Precision at top n ($P@n$), Recall at top n ($R@n$), Mean Reciprocal Rank (MRR), among others. On the one hand, $P@n$ metrics are meant to measure the proportion of experts among the first n recommended users. On the other hand, $R@n$ is meant to show the fraction of experts (among all experts) that are recommended, given n results. For instance, a high $R@n$ and a low $P@n$ would say that our approach is almost all the participants in a given question although they are a non-negligible proportion of users that are not participants of it. *A priori* we do not know if the non-participants recommended are qualified or not but they share similar positions in the ranking with actual participants. Therefore, we consider that $R@n$ is more appropriate to measure the quality of our approach to recommending qualified users. A good result would be that our approach shows a high $R@n$ for a low n.

A good performance to this metric would be to have high values of recall (the closer to one, the better) with lower values of k.

RQ3 -Distribution workload of TMBA

Karma score is, in the *ROS Answers* platform, a value that represents the degree of participation of the user in providing valuable answers[6]. Without an expert recommender system, a user in need of help would appeal to users in the first positions of the Karma Ranking[7]. This affects the workload of the community because the most active users would be demanded to answer questions while other less active but qualified users would participate in the discussion seeking for a correct answer for a given question. We compared the value of the Karma points

[6]FAQ - ROS Answers, Website,`https://answers.ros.org/faq/#:~:text=When%20a%20question%20or%20answer,users%20based%20on%20those%20points.`

[7]Users sorted by Karma Points - ROS Answers, Website, `https://answers.ros.org/users/?sort=reputation`

Table	Records
ros_answers	128.433
ros_question	40.995
ros_user	21.388
ros_question_answer	128.433
ros_question_tag	127.363
ros_tag	14.253
ros_user_tag	149.518

Table 6.2: Records per table

of the first user recommended by TMBA. A good performance would be to recommend users with higher positions in the Karma ranking.

6.5.1 Collected Data

We used a data set of questions, users, and tags from ROS Answers. We build a database from data gathered scraping its website using the `ros_gh`[8] tool. The tool was run on 29 January 2019 [9].

The database contains 40.995 questions, 21.388 users, and 14.253 tags (more details on Table 6.2). From those questions, we used only the questions with two or more participants (the question's author plus more users), which means, with at least one answer. We work with a universe of 22.293 questions. Regarding tags, as it was mentioned in Section 6.4.3, the 14.253 tags were filtered using a standard list of stop-words. Another filter was imposed by the frequency of use of the tags: we considered only those that were used in at least 3 questions. In total, a corpus of 9.112 tags was used.

6.5.2 Calculation of R_{uq} and R_{ut} table

The *TMBA* score is based on the values of $R_{ut}, \forall u \in U, \quad \forall t \in T$ which, depends on the value of $R_{uq}, \forall u \in U, \quad \forall q \in Q$. We present the algorithms used to calculate these matrices.

We first need to calculate the relationship between a user and the questions in which they have participated. To do that, we calculate these values using 6.6. This is a very simple algorithm in which we iterate over all the question ids and all the user ids as they are given as input. For each pair of a user and a question the number of $Activity(u, q)$ which is the number of answers and comments posted by the user u in question q. $Question\_Activity(q)$ is the number of answers and questions in q.

Now, for the calculation of the R_{ut} matrix we follow a similar approach. We iterate over

[8]Braulio López, ros_gh, (2020), GitHub repository, `https://github.com/elbraulio/ros_gh`

[9]The present study was interrupted due to a series of factors external to the thesis development such as a social revolt in Chile and a long pandemic. However, the sample is representative due to the nature of the data: the intrinsic characteristics of the ROS community. We expect that a fresher sample of data would lead to similar results. In concrete, this study reflects the behavior of the ROS community in that period of time which is consistent with the results and discussions of the previous chapters.

Algorithm 1 Calculation of R_uq table

```
$R_{uq} = Matrix(|Users| \times |Questions|)$
for $q \in Questions$ do
    for $u \in Users$ do
        $R_{uq}(u,q) \leftarrow Activity(u,q)/Question\_Activity(q)$
    end for
end for
return $R_{uq}$
```

all tags in $Tags$ and all users in $Users$.

Algorithm 2 Calculation of R_ut table

```
$R_{ut} = Matrix(|Users| \times |Tags|)$
for $t \in Tags$ do
    for $u \in Users$ do
        if |Questions_with_tag(t))| != 0 then
            $K \leftarrow log(|Q|/|\text{Questions\_with\_tag(t))}|$
            $R_{ut}(u,t) \leftarrow K \cdot \Sigma_{q \in Q_t} R_{uq}(u,q)$
        end if
    end for
end for
return $R_{ut}$
```

6.6 RQ1: Does TMBA predict the actual answerer of a question?

The first evaluation considers where, in the ranking given by the *TMBA*, the correct answerer is positioned. This is: for each question, we get the position in the ranking in which the user that answered it correctly is placed. We performed this evaluation for each scenario: A (Q U), B (Q+ U), C (Q U+) and D (Q+ U+). See the histograms in Figure 6.8. The results are plotted as histograms, in the X axis there are the ranking positions from 0 to 100, and the Y axis represents the number of cases in which the correct answerer was positioned in that place. The vertical bar in the right of the histogram are the cases in ranking 100 and also those cases in which the correct answerer was not found in the ranking of the first 100. A positive result would be an histogram with high frequencies in first positions decreasing the frequencies in further places, picturing a Power Law kind of distribution with a thin tail.

All the scenarios follow the same shape of higher frequencies in the first positions and lower as it goes to further places. We can also observe a vertical bar at the right of each histogram. That is the count of users that were positioned in the 100th place but also those users that did not appear in the first 100th places of the ranking. The height of the first positions and the height of position 100 (answerer not found) differs from scenario to scenario. In a quick look, we can observe that Scenario C presents the best performance because the concentration in the first places is higher than in the other scenarios and the bar at 100 is shorter than in the other scenarios. In the opposite, the worst is in Scenario D.

We can confirm this by interpreting the Empirical Cumulative Distribution Functions of the four distributions. In a perfect recommender system, the curve would be a vertical line up followed by a horizontal line, something like an upper-left corner. This is because all the correct answerers would have been ranked in first place. Therefore, the steeper the curve is, the better performance of *TMBA* in that scenario. We plot the different functions for each dataset (*i.e.,* each Scenario) at the Figure 6.9.

We can observe that the three curves have a similar logarithmic-ish shape growing fast in the first values and then reducing the tilt of the curve. We can also point out that at the end of every curve, there is a vertical line that represents the cases of *TMBA* in which the *TMBA* in that scenario did find the correct answerer among the 100 first positions. However, it is clear that Scenario C (green curve) presents the best curve: its steep is the highest and also reaches the highest value at the end of the curve. The difference between the other curves is noticeable.

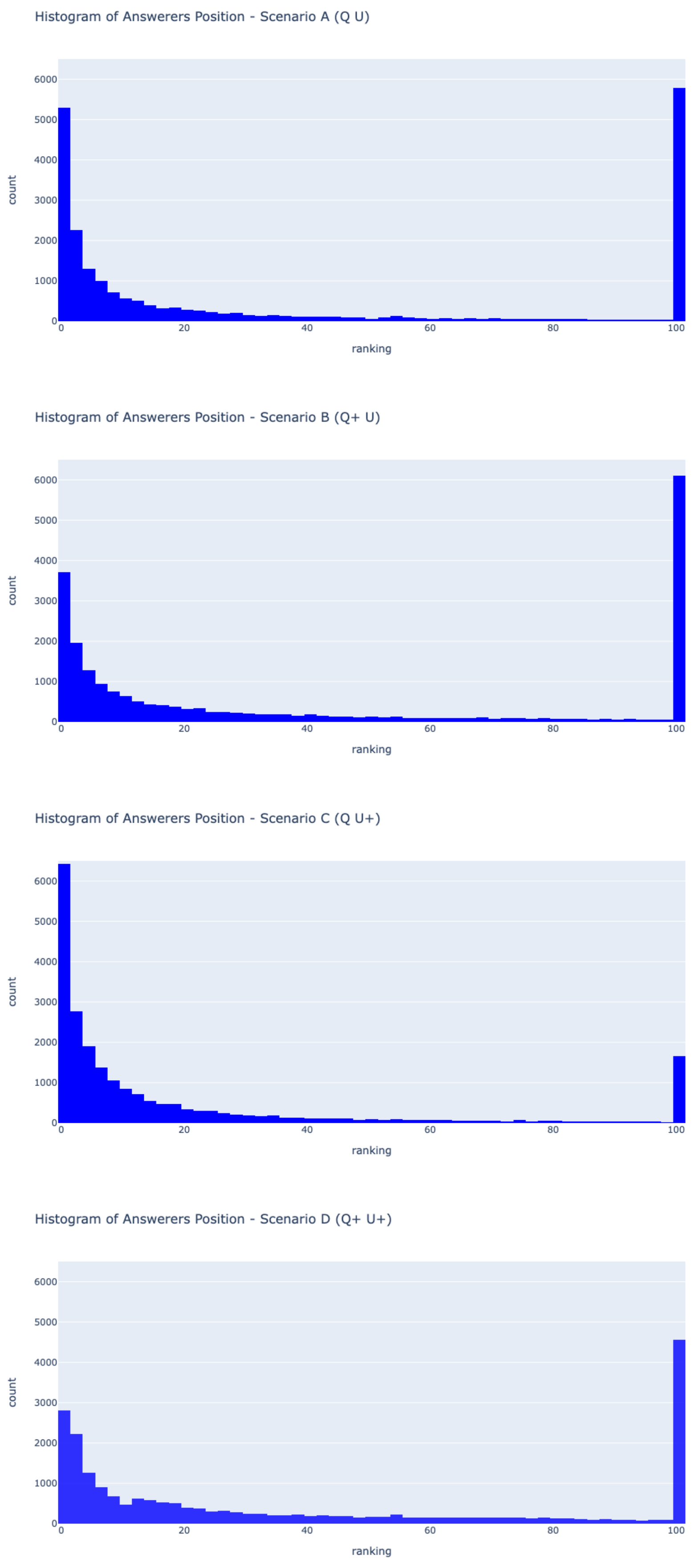


Figure 6.8: Histogram of Hits - Scenarios A, B, C and D

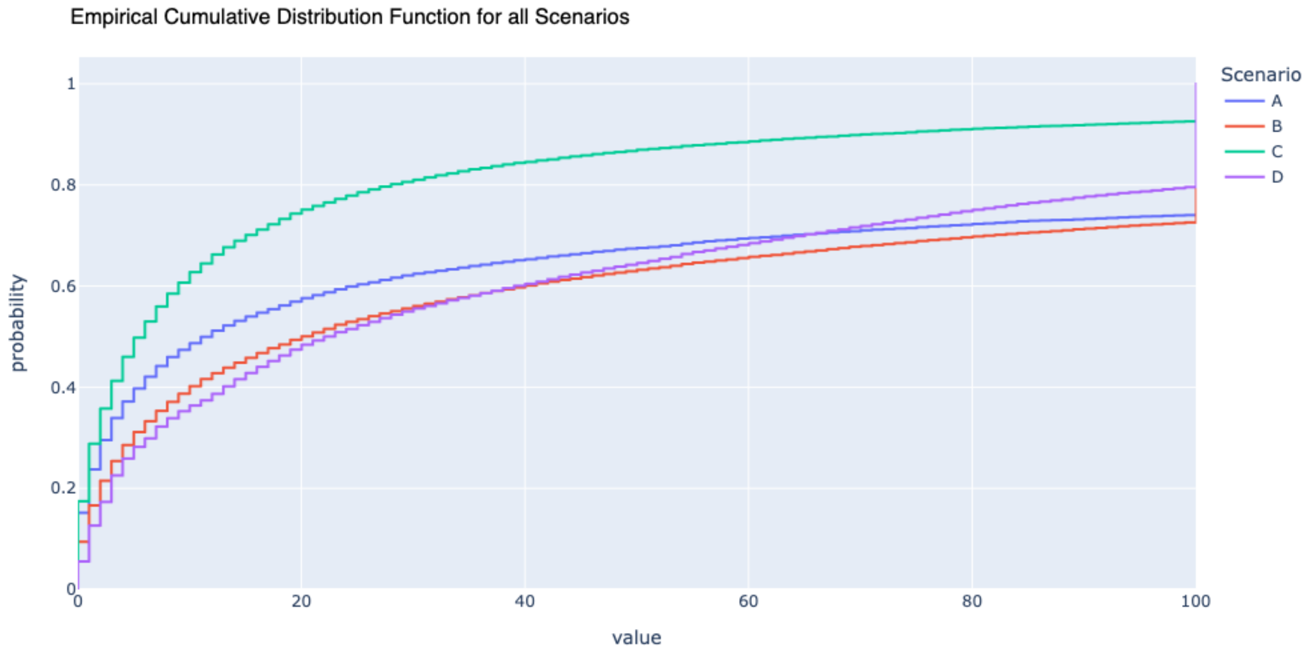


Figure 6.9: Empirical Cumulative Distribution Functions for each Scenario

6.7 RQ2: Does TMBA recommend qualified users?

The second evaluation of the *TMBA* is the recall@k of the ranking.

The motivation behind this evaluation is to confirm or deny that *TMBA* recommends qualified users for answering a given question. We consider that the participants of a question are competent users in the search for an answer to a question.

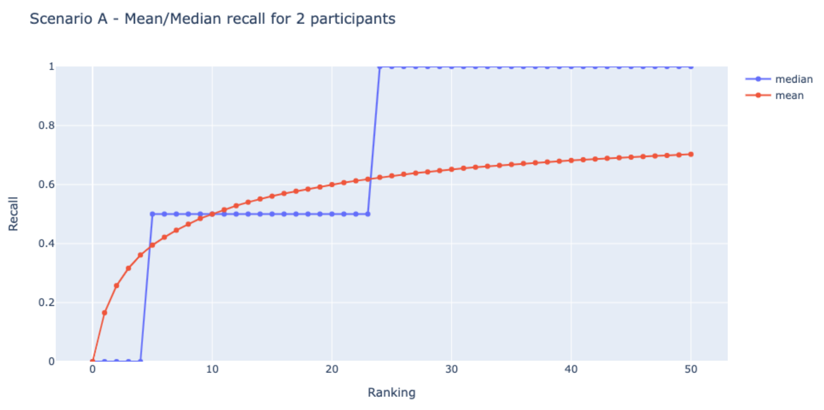
It is calculated by counting the number of participants in the first k positions of the ranking and dividing it by the number of participants of a question.

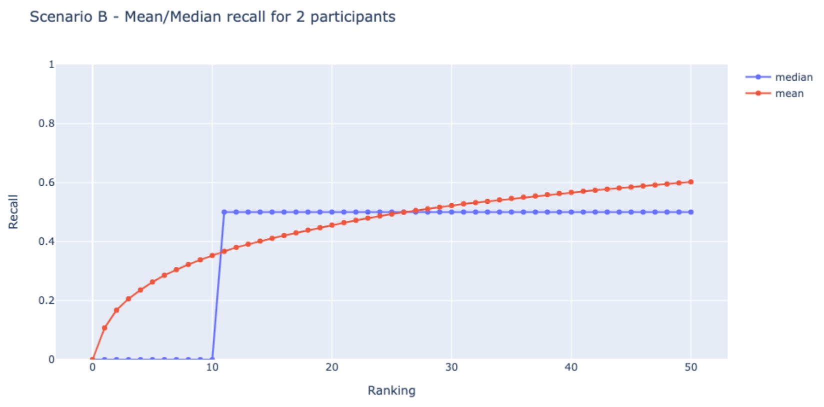
In this evaluation, we split the evaluation according to the number of participants to a question. Recall that a participant of a question is either the author or an answerer of a question. We evaluated the recall@k in all questions in each group of questions that have from 2 to 10 participants. For each recall calculation, we plotted the median and the mean of the distribution of recall@k in each position in the first 50 positions.

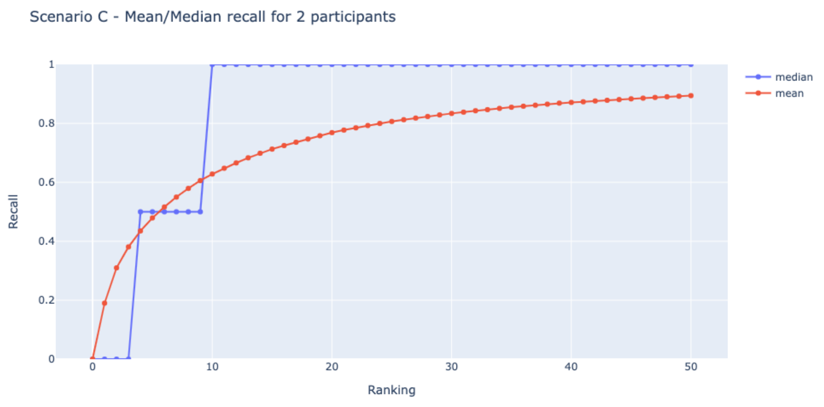
In the following figures we can see the results for the different scenarios for questions with 2 participants (See Figure 6.10).

In blue we have the median of the distribution of recall@k and in red the mean. To compare the results we have to see how fast the median reaches 1 and the slope of the mean. For $n = 2$ we can see that Scenario C provides the best result, the median reaches 1 at $k = 10$ and also the curve of the mean grows faster than the other scenarios.

As n grows it is more difficult for *TMBA* to find all participants in a question with few candidates, hence, the median and mean curves grow towards 1 in a slower fashion. This







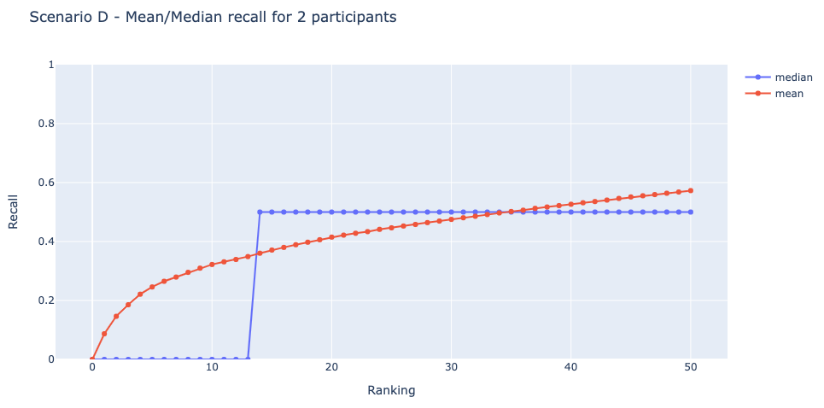


Figure 6.10: Recall for 2 participants

occurs in all scenarios. Comparing the figures of all results[10] the Scenario C performs better in most of the values of n. The exceptions are the cases $n = 7$ and $n = 8$ in which case Scenario A performs slightly better than Scenario C. However, the trend is that Scenario C is the one with the best performance overall.

We can see this phenomenon in Figure 6.11 and Figure 6.12. Although Scenario C provides the best performance in comparison with the other scenarios (see Annex for details), the recall is lower in $n = 5$ and $n = 10$ than in $n = 2$. This is because the probability of finding all 5 or 10 participants of the question is lower. We can comment that this kind of question with various participants represents questions with high discussion. It could be related to clarifications, different points of view expressed in the thread, or a complex question. It is more difficult for *TMBA* and any other ERS to find qualified users to answer such kinds of questions.

We can conclude that the behavior of TMBA in Scenario C is better than in other scenarios and its performance for recommending qualified users decreases as the more complex or controversial the question is (high amount of participants)

[10]All the figures of the results (36 figures) for $n > 2$ are shown in the Annex Chapter C.

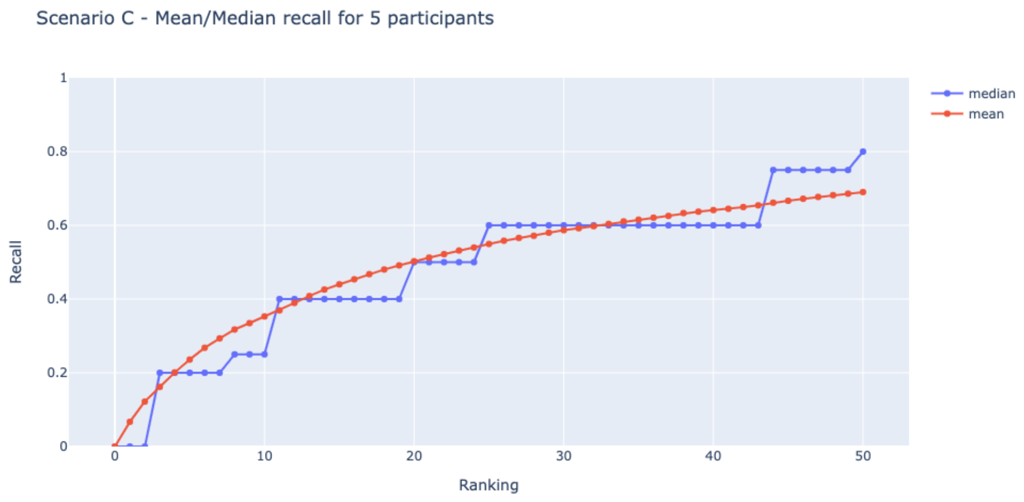


Figure 6.11: Recall - Scenario C - 5 participants

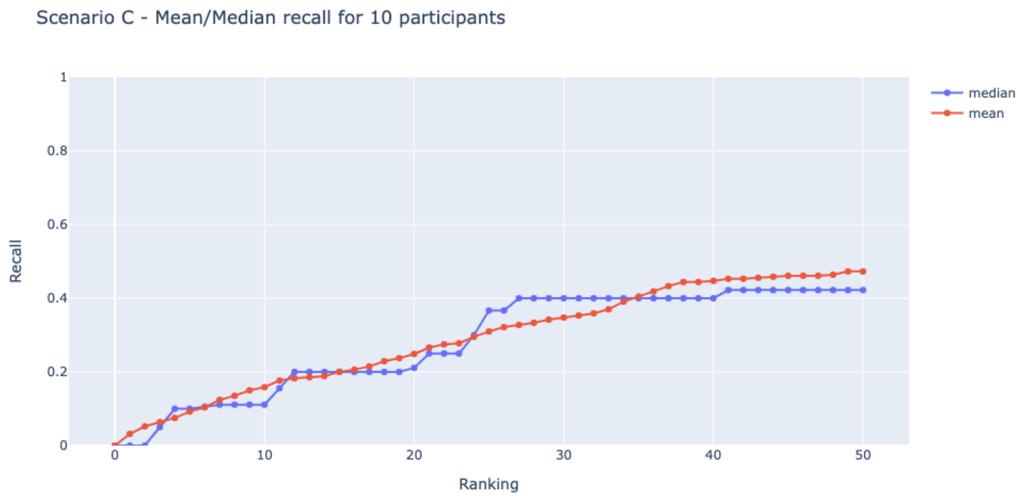


Figure 6.12: Recall - Scenario C - 10 participants

6.8 RQ3: Does TMBA facilitate the workload distribution of answering questions?

The third evaluation of *TMBA* is the workload distribution among users of *ROS Answers*. To do that we use the Karma points of users and the Karma ranking[11]. Karma points are values associated to users based on their participation in *ROS Answers*. A user gains Karma a number of points when a given answer or a posted question gets an *up-vote*[12]. Therefore, a user with high Karma points is a user that has high participation in posting questions or providing answers that the community considers valuable.

The motivation behind this is that we expect that the recommender system promotes users that have less participation (fewer Karma points) in answering questions in order to reduce the workload of users with higher participation. The solution without a recommender system would be to go and ask the users in the first position of the Karma ranking. We expect to ameliorate this situation by relying on the use of *TMBA*.

In order to observe that, we observed how many times a user was recommended in the first, second, and third place and compared this against their position in the Karma Ranking. A "good" performance of *TMBA* would be to show higher values in users with higher places in the Karma Ranking and less in the first positions. As mentioned, we also wanted to know that if it was not the case, how *TMBA* performs in recommending the second and third bests.

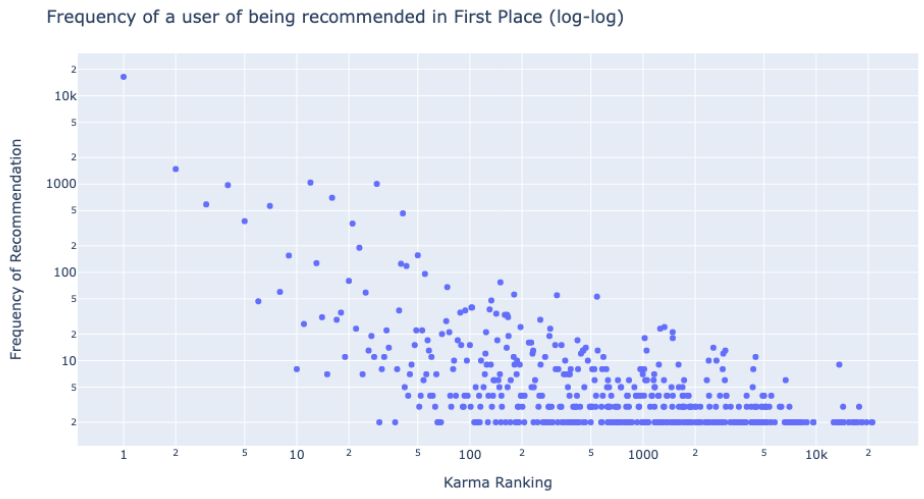
As Scenario C presented the best performance in the two previous analyses we applied this evaluation using rankings from Scenario C. In the Figure 6.13 we can see the three scatter plots of the frequency of recommendation: First Place Recommendation (1PR), Second Place Recommendation (2PR), and Third Place Recommendation (3PR), against the Karma ranking. As there were huge differences between frequency values and the amount of users is also large, we proceeded to plot them in scale log-log. For the analysis, we will separate users highly participative as the ones in positions 1 to 36, and the rest as less participatory. This threshold is set because the *ROS Answers* website shows the users ranked by Karma in batches of 36 users, hence, the more participatory users are the ones that are on the first page of users.

We can observe that the three figures show quite similar distributions. High values in the first places and lower them as the ranking goes to further places. Notice that the first user in the Karma ranking is recommended more times in the 1PR (Figure 6.13a) than the 2PR (Figure 6.13b), but the second user use to appear more frequently in the 2PR and 3PR. And the third user is more recommended than the first one in the third position (3PR).

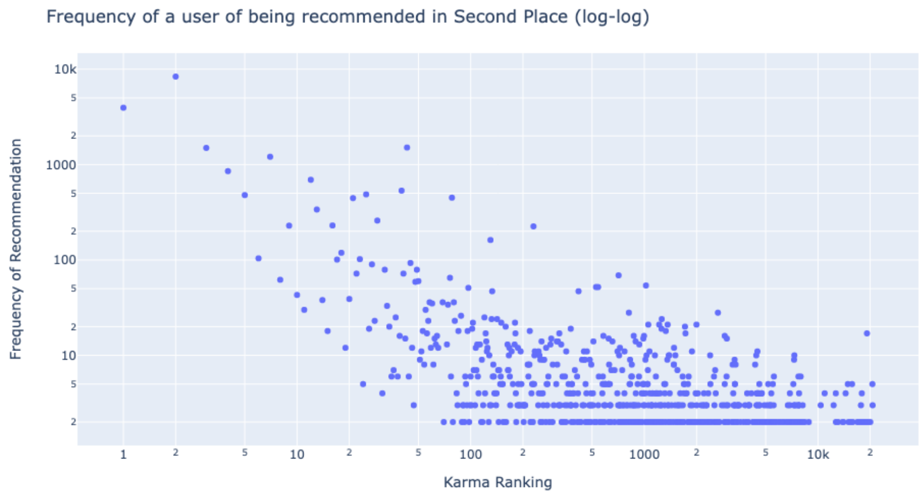
The right side of the figures shows the users further in the Karma ranking (*i.e.,* less participation in the *ROS Answers*). In the three figures, we can observe that the majority has frequency values under 100 with a high concentration under 10. The distribution is somehow similar from ranking 100 to 90.000 with many cases of 5 or 2 times of being recommended. In the 2PR and 3PR charts, we can identify a couple of outliers. For the 2PR, in position

[11] Karma Ranking - ROS Answers, Website, `https://answers.ros.org/users/?sort=reputation`

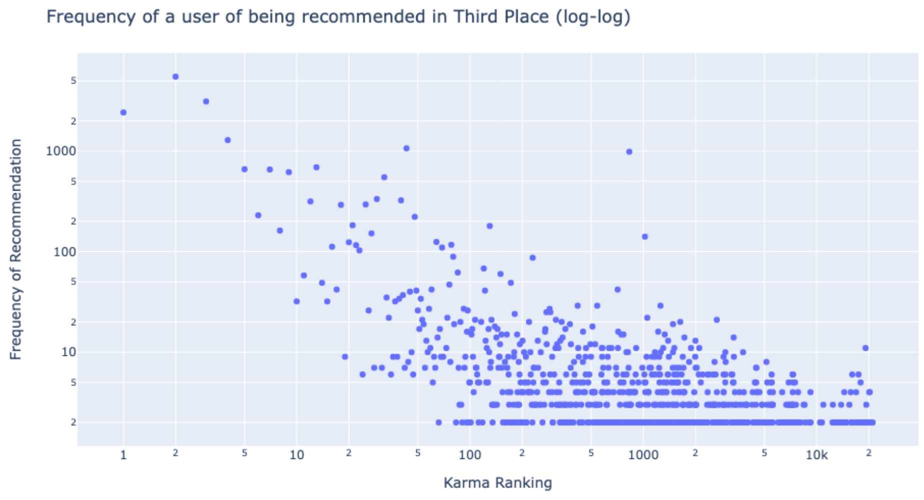
[12] A vote in favor of a given answer or comment for a question. In the top-left corner of the screenshot in Figure 6.6, there are two arrow-shaped buttons. Those are up and down votes.



(a) First Place Recommendation (1PR)



(b) Second Place Recommendation (2PR)



(c) Third Place Recommendation (3PR)

Figure 6.13: Frequency of recommendation users in 1st, 2nd and 3rd place sorted by Karma Ranking

43 there is a case of a user being recommended 1509 times (similar to the 3rd user). In the 3PR, the same user (43rd) has 1066 recommendations, which is close to the 4th place.

In conclusion, we can confirm that the three cases show a distribution similar to a Power Law which means that *TMBA* does not help in recommending less participative users. Furthermore, it highly recommends the three first users in the three cases.

6.9 Discussion

The objective of the study is to test whether TMBA can find qualified people to reduce the workload of answering questions in the *ROS Answers* platform. We used TMBA through 22.293 questions to rank users who are likely to answer a given question due to their participation in *ROS Answers*, each user is profiled by tags. We extended the tags used in Questions and Users having 4 combinations that we call Scenarios A (Q U), B (Q+ U), C (Q U+) and D (Q+ U+). We explored the compliance of the objective of the study through 3 evaluations.

The first is a *Sanity Check* of the TMBA approach. It shows where in the ranking are the users who gave the accepted answers. We evaluated this in each scenario. The scenario with the best performance was Scenario C. Although there is a non-negligible amount of users in the tail of the histogram, 62.7% of times TMBA recommends the correct answerer in the first 10 positions. Also, the ECDF curves show clearly the prevalence of Scenario C over the others as its curve is steeper at the beginning and grows higher than the rest. With this result, we can conclude that extending tags in Questions (Scenarios B and D) only creates noise in the relationship answerer-question. But extending tags in Users only may help to *TMBA* to correctly find the answerer of a question (Scenario C), more than the case with no tag extension (Scenario A).

The second evaluation was to verify if TMBA recommends qualified users to answer a given question. To do that we employed the measurement of recall@k metric because it provides a measurement of how many actual participants of the answer of a question are ranked in the first places. To apply this metric we had to split the questions to analyze by the number of participants in that question. From 2 participants, the author and an answerer, to 10. This was applied considering the 4 scenarios. As well as in the first evaluation, Scenario C performs better in the majority of cases (in cases 7 and 8 participants, Scenario A performs slightly better). Although, the performance of the recall@k (in all scenarios) of TMBA worsens when the number of participants increases: it's more difficult to obtain all the participants in the first place. This can be explained because questions with a greater number of participants can be more difficult to answer: as there are many participants there is a deeper discussion or the questions needed more clarifications from the author. We can say that TMBA provides few qualified candidates for answering a given question. Considering these 2 results it can be seen that extending the tags associated with a user helped to get better results. Adding extra tags in questions only adds noise worsening the performance of TMBA. Extending tags to users may help users with less participation, ergo fewer tags, to have a more robust profile to have a better score.

In third place, we evaluated the ability to distribute the workload of answering questions among users, focusing on promoting less engaged users. This objective goes in the direction of enriching the collaboration dynamics in the community and distributing the knowledge among users. In order to observe this we got the distribution of Karma points of the users that were recommended in the first place for answering a question using Scenario C. As can be seen in Figure 6.13, TMBA promotes mainly users with high Karma in the majority of the cases. Only 14% of promoted users have minor participation in *ROS Answers*, so even though users with less Karma were better profiled by adding extra tags, this profile improvement did not seem to improve achieving the desired result: a better workload distribution. This can be contrasted with the sanity check: users with high Karma are commonly the ones that answer correctly an answer and, as TMBA passes the sanity check, it is normal that TMBA promotes mainly users with high Karma. In conclusion, as a minor amount of less participative users is promoted, in most cases TMBA does not distribute the workload of answering questions in a uniform way.

We did not obtain similar results to Zhang *et al.*. The results show that the variables meant to be optimized are not: to provide qualified users to answer and to distribute the workload more uniformly. Among the reasons for these results is that the nature of *ROS Answers* is different than in other Q&A sites due to the high diversity of topics that involve robotics and in some cases the need for deep understanding in such knowledge areas for participating in the thread of a question. Also, questions can be related to a specific hardware, equipment, or robot that only a few people own. In this way, the TMBA approach seems to be limited. We consider that the TMBA approach does not fit well as a recommender system to be implemented in *ROS Answers*.

As threats to validity, we can consider that the dataset was obtained in 2019 and the community may have changed. We expect not, since there has not been a substantial change in the way that the community works (for example, the emergence of Community Managers, and restrictions in contributions). In addition, we considered only the tags that appeared in at least 3 questions. Its inclusion could have altered the results in terms of worsening the scores due to the specificity of the terms. Although we consider that ruling out them removes unnecessary noise in the way the scores are calculated. Finally, We excluded questions with an unusually large amount of participants, specifically, those questions with more than 10 participants (9 answerers + the question's author). From the universe of possible questions, we considered 98,9% of them.

6.10 Conclusions

This chapter presents the study of an Expert Recommender System for a Question & Answers community, specifically ROS Answers. The main motivation and objective of this study are not to find the most qualified user to answer a question but to find a *qualified answerer* who does not participate much in the platform. This is because we saw that the majority of the answers are given by a small number of users, and these users are prone to be presented as answerers of given answers. We pretend to balance the workload by distributing it. We adapted an approach of a paper by Zhang *et al.* [125] which takes into account the participation in the past of the user in questions with similar topics.

We saw that the tags in a question were few and could be complemented by the presence of tags in the title and body of the question. This tag extension can be performed in questions and users. We then have 4 scenarios considering or not the extension of tags. These scenarios are:

1. Scenario A (Q U): no tag extension for questions nor users
2. Scenario B (Q+ U): tag extension for questions only
3. Scenario C (Q U+): tag extension for users only
4. Scenario D (Q+ U+): tag extension for questions and users

We evaluated *TMBA* to answer three research questions.

RQ1 Does TMBA predict the actual answerer of a question?
RQ2 Does TMBA recommend qualified users?
RQ3 Does TMBA facilitate the workload distribution of answering questions?

For the first question, we checked when the TMBA positioned in the first place to the correct answerer of the question. This is the user whose answer was given as correct or accepted. We performed this experiment for each scenario. The results showed poor performance in Scenarios B and D, and C has the best performance. Actually this scenario positions correctly the best answerer 62.7% times. TMBA slightly passes the sanity check in Scenario C.

The second question makes use of the recall@k metric to study the property of TMBA to recommend qualified answerers. The ground truth in this evaluation is that the participants of a question are competent and qualified users for answering the question. Using this approach, a good performance of TMBA for a given question is that it recommends the participants of the question in the first place. We run the experiment over the four scenarios aforementioned. The results are similar to that in the first research question: Scenario C presents the best performance. The results also show that although the performance of TMBA is fairly good for questions with few participants (2 or 3), the more participants a question has the more difficult it is for TMBA to present all answerers in the first place. This can be explained by that questions with many participants tend to be more complex than others in which the answer seems to be more straightforward. We answer then that TMBA in Scenario C recommends qualified users for questions that are not complex or controversial.

Finally, the third question aims to reveal if TMBA distributes the workload in answering questions. This was studied by observing which kind of users does TMBA recommends as 1st, 2nd, and 3rd choice. The results show that TMBA 86% of cases recommends users that have high Karma points. This affirmation is not only valid for the users recommended in the first place but also for users recommended in the second and third places. In conclusion, we can argue that TMBA does not facilitate the workload distribution of answering questions.

Chapter 7

Final Remarks and Future Work

In this chapter we summarize the contributions of this dissertation. We also expand on the confirmation and rejection of the hypotheses and the achievements of the proposed objectives. Finally, we develop on the future work and final remarks.

7.1 Contributions

In the following we restate on the four contributions of this work.

7.1.1 Methodology and Evidence of Code Duplication in ROS Launch files

A first study on the ROS Ecosystem was made by looking at the ROS Launch files: a software artifact of ROS itself that is in charge of launching processes and describing the connections between them in order to achieve a robot behavior. We saw that ROS launch files are present in almost every non-trivial ROS package, and some of them define many launch files. Our study was focused in intra-package code duplication on these files.

We define a code clone as a portion of code of at least 7 lines that is repeated in two or more launch files. We found that 40% of the ROS packages that define more than one launch file have code clones. More than 80% have 6 clones or less: almost half of the packages have one clone, and 1/5 have two or three clones. Also, the similarity of the launch files was over 45%, which means that is normal that developers to reuse code snippets in several launch files.

Through the use of 3 metrics: average overlap, the proportion of launch files in clone pairs, and clone cohesion (ratio of the number of clones and launch files involved in code duplication), we prioritized 10 packages for an in-depth analysis. We obtained several insights, one of them is that in some packages, the clone represents a big portion of the launch files thus the different launch files are a single code snippet repeated all over different files with minimal differences.

We also studied the Git history of the prioritized packages to see the impact of the presence of code duplication in ROS launch files. We found that *value tweaking*, *changes in the file system*, and *package replacement* are common cross-changing situations in which a change in one clone needs to be propagated all over the involved files. We also found cases in which, after initial changes in the launch files, the code clone remains intact and is reused in other files.

7.1.2 Evidence on Socio-Technical aspects of the ROS Ecosystem

We performed a field study that combines both a qualitative study(interviews and open coding) and a quantitative one (survey) and we gained valuable insights into the socio-technical aspects of the ROS ecosystem. This contributes specifically to the field of Empirical Software Engineering. We obtained observations about the package reuse experience, looking for help activities and platforms, and contributing artifacts.

On package reuse, we identified that the majority of surveyed users have reused at least one community maintained (95%) and a similar proportion claims to have a least one dependency on such packages. However, 71% of surveyed users report having had the situation of having failed in integrating them into their projects. The reasons are varied: the package was for an outdated ROS distribution, lack of documentation, and the presence of bugs in such packages are the most frequent.

Related to looking for help, the main platform for doing that is ROS Answers, a Q&A website about topics related to ROS use and robotics. Surveyed answerers are more prone to search for already existing questions than asking. They first try to scratch their own itches looking for solutions by themselves or ask a colleagues instead of asking questions. Answering is also not a popular practice because lack of time. Other sources of help are ROS Wiki in which the most common practice is searching for information rather to creating or editing pages. Finally, Advanced users claimed to search for snippets in the GitHub platform where they can also interact with maintainers.

Finally, we have evidence about the artifacts that are contributed to ROS by users. Advanced and Intermediate users, due to their experience, tend to contribute the most. For example, 77% of Advanced users claimed to has reported a bug and bug fixes. In the ROS Answers platform, Intermediate users report to have asked questions while Advanced have more participation both asking and answering questions. Other less popular artifacts of contribution are Feature requests and documentation.

This contribution represents an important snapshot of the way ROS users interact in the ecosystem. And give hints to community managers on how they can ameliorate or facilitate the contribution dynamics among the community members.

7.1.3 Recommendations for addressing contribution bottlenecks

From the field study, we distilled five main bottlenecks in contribution dynamics in the ROS ecosystem. They are:

- **B1. Lack Of Time**: nearly 60% of surveyed participants claim this bottleneck as the main barrier to contributing.

- **B2. Package Abandonment**: this is when the maintainer leaves the project without notifying. 90% of surveyed participants have encountered abandoned packages.
- **B3. Lack of confidence in the value/quality of a contribution**: this barrier affects mostly Beginners and Intermediate users. They are not confident about the correctness, quality, or value of their contribution.
- **B4. The contribution could be too specific to my problem, domain, hardware, or research problem**: Intermediate or Advanced users think that their contribution could be not valuable due to the specificity of the contribution.
- **B5. Workflow is unknown or not clear**: although a minority, some users claimed not to know how to contribute, which platform to use, and how.

In addition to this, we proposed five recommendations to overcome these bottlenecks and ameliorate their symptoms. They are:

- **R1. Identify and Predict Abandoned Packages**: automated approaches can ameliorate the scenario of abandoned packages. They can facilitate the identification of such packages and foresee the risks of an already maintained package being abandoned.
- **R2. Provide an Informative Package Repository**: is important to have an automatically updated catalog of packages available to be reused. Installation, configuration, and troubleshooting activities can be present in that catalog.
- **R3. Recommend contribution opportunities to qualified community members**: in the context of volunteer contributions and the specificity of the robotics fields could be benefited of expert recommender systems. These systems could be used for finding maintainers for abandoned packages or answerers of unanswered questions in ROS Answers.
- **R4. Limit breaking changes**: an automated approach for upgrading and migration among versions would benefit users for working the last version of ROS without having to struggle with incompatible versions of third-party packages.
- **R5. Motivate and encourage community contributions**: other communities have successful cases of motivating their community to achieve tasks that improve the ecosystem's health. Giving rewards, and organizing fests and campaigns are examples of that.

The identification of contribution bottlenecks is a valuable piece of information as it allows ROS community managers and core members to acknowledge the main bumps in contribution dynamics. And providing recommendations to give solutions to face such problems.

7.1.4 Novel approach for actively distributing knowledge sharing workload in Software Ecosystems

Adapting the approach of Zhang *et al.* [125] for recommending developers for open source projects, we designed a novel expert recommender system for increasing the participation of qualified less participatory users in the ROS Answers Q&A platform.

This approach is called Tag Map Based Algorithm since the main entity in the calculation is the tags associated with the question to find answerers for. Tags are words that help on describing the content of a question. This approach calculates the relationship of the user with questions in which it has already participated and also the relationship of a user and a tag.

Usually, expert recommender systems aim to find the best possible answerer for a given question. These systems tend to recommend users that have high participation in asking and answering questions. Our idea is to find qualified users able to answer a given question but we want them to be less participatory than those users who always answer questions. This way the workload can be better balanced among the whole community. Although this approach shows not being successful in its purpose, the problem proposed is novel itself.

7.2 Discussion on the Objectives and Hypotheses

■ *Goals*:

We can conclude that almost all the proposed goals were achieved. Each chapter represents one goal. Chapter 3 refers to the first goal *Study clone activity in ROS Launch files* and consists of an in-depth study of these ROS artifacts. Chapter 4 has a correspondence with the second goal *Characterize the collaboration dynamics in the ROS community*, the field study shows how users use ROS, how they look for help, and what they contribute to the ecosystem. As for the third goal, *Identify the more important bottlenecks in contributions and provide recommendations to address them*, in Chapter 5 we addressed it by identifying 5 contribution bottlenecks and giving the same number of recommendations. Lately, on Chapter 6 we addressed the fourth goal, *Recommend qualified members for knowledge contributions*. We presented a tag-based approach to recommend users that are less participatory but qualified members of the ROS Answers community to answer a given question. We consider that this fourth goal was not achieved since although the system recommends qualified answerers, it only recommends less participatory 1/4 of the cases.

■ Hypothesis 1: *By performing code analysis of ROS launch and configuration files, it is possible to find issues in the ROS ecosystem*

In Chapter 3 we performed a complete analysis of a specific ROS artifact: ROS launch files. These artifacts are configuration files used to declare processes and connections between them in order to launch a robotics program.

In particular, we analyzed the duplication of code fragments in ROS launch files. In there we identified that 40% of eligible packages, packages with more than one launch file, present cases of code duplication. Moreover, half of them have two or three clones (fragments of duplicated code). Also, the majority of the aforementioned packages have a similarity of over 40%.

We also performed an in-depth analysis of 10 out of 23 packages with a high amount of clones (over 7), the most relevant according to the average overlap, clone cohesion, and proportion of launch files involved in clone cases versus all launch files defined with clone cases. We found that at least 6 have more than 60% of their code with clone cases, and 5 packages have more than half of the whole launch file repeated.

These results show the existence of a clear pattern of code duplication activity over this software artifact. There are well-known negative consequences that duplicated code can trigger when propagating changes in duplicated fragments [63]. Having duplicated code is a latent source of bugs or inconsistencies, and, in the latest case, duplication of efforts. Copy-and-pasting damages the software ecosystem.

Hence, the **hypothesis 1 is confirmed**: the ROS ecosystem's health can be analyzed by performing code duplication analysis on ROS launch files.

■ Hypothesis 2: *Contribution bottlenecks in ROS can be attributed to socio-technical issues within the ecosystem*

In Chapter 4 and Chapter 5 we presented a field study on the experience of ROS users when reusing ROS packages and contributing to the ecosystem. It consisted of a set of interviews, a focus group and an open online survey (both can be inspected in Annex A and B).

We identified that contributions are mainly done by Intermediate and Advanced users. They reported several situations that hamper the possibility to contribute. It varies from type of contribution: question or answer in ROS Answer, bug fix, feature requests, documentation. We summarized these findings into 5 contribution bottlenecks already presented in the referenced chapter and in the contribution Section 7.1.3.

The *B1. Lack Of Time* is a phenomenon typical of voluntary activity as the contribution in an open-source project is. For *B3. Lack of confidence in the value/quality of a contribution* and *B4. The contribution could be too specific to my problem, domain, hardware, or research problem*, we consider them as the same root: there is no acknowledgment that every contribution is valuable and it does not matter how specific it is. Regarding *B5. Workflow is unknown or not clear* we think that it is more a technical issue as the way to contribute to it is not clear enough. Finally, the second bottleneck *B2. Package Abandonment* is definitively both a social and a technical issue. Social because it is an active contributor that disappears and dependant users and contributors are not able to participate in using nor contributing to a package. And technical because of the lack of mechanisms to predict, identify and overcome this phenomenon.

In conclusion, we can **confirm hypothesis 2**: contribution bottlenecks in ROS are attributed to socio-technical issues.

■ Hypothesis 3: *Expert Recommendation Systems based on the use of tags are useful for finding less participatory qualified candidates for answering questions on the Q&A site on a robotics software ecosystem*

In Chapter 6 we presented a novel approach of Expert Recommendation System for Q&A communities: the Tag Map Based Algorithm (TMBA). This is based on the work of Zhang *et al.* [125], which proposes a recommender system for software developers in software projects based on their participation in similar projects, in GitHub. We adapted their approach by replacing developers with users of the Q&A community ROS Answers, and software projects by posted questions.

We applied this system over the ROS Answers platform, the Q&A website *de facto* for robotics questions using ROS. In there, users post questions and provide answers, and points are given by the value of their questions/answers: Karma points. Users with high Karma points tend to be the most participatory users in ROS Answers. The objective when designing TMBA was not only to provide qualified answerers but also to help with a better workload distribution in this community by recommending less participatory qualified users.

The goals of this system were studied through three research questions: RQ1. Does TMBA predict the actual answerer of a question? (Sanity Check), RQ2 Does TMBA recommends qualified users?, and RQ3 Does TMBA facilitates the workload distribution of answering questions?

For the first question we saw that TMBA recommends better when extending the tags for Users only, and it slightly passes the Sanity Check. For the second question, the TMBA behaves well when tested in questions with fewer interactions between participants (author and answerers). This can be explained that when more users participate in searching for an answer the more complex or controversial the question is. The results for the third question were that TMBA tends to promote members with high karma most of the time even for the 2nd and 3rd position in the ranking this trend of recommending by Karma points does not change.

In conclusion, the TMBA approach, most of the time, does not recommend less participatory qualified users. Whereby we proceed to **partially reject the hypothesis 3** as there can be other more sophisticated approaches that can actually work on the intended goal.

As one of the reasons for the low performance of TMBA is the simplicity of Zhang's approach. If we compare more recent approaches they profit from techniques using machine learning, more specifically, natural language processing, that we think could have benefited the search for qualified users. Also, if we were interested in recommending less participatory users we might have included the factor of the Karma points in the algorithm.

7.3 Future Work

The future work of this dissertation is based on the following points:

- *Analysis of code duplication in other artifacts*: Code duplication studies are commonly applied to source code. It would be very interesting to know how is the copy-and-paste activity in such artifacts.
- *Compare findings regarding the socio-technical evidence*: We found many interesting aspects of how users interact with the ROS software and also in the community. This study can be repeated in other software ecosystems with similar characteristics: diversity of knowledge, high specialization in different disciplines, and/or use of heterogeneity of hardware.
- *Confirmation of bottlenecks in other ecosystems*: We think that bottlenecks such as *lack*

of time, *package abandonment* and *workflow unknown* are probably common bottlenecks in other software ecosystems. Although, we don't know the impact on such ecosystems and what they have done to mitigate the effects of them. Also, it would be interesting to investigate the other bottlenecks that are partially explained by the characteristics of the ROS ecosystem. Finally, another question that shows up is if there are other particular bottlenecks in such ecosystems.

- *Applicability of recommendations in other ecosystems*: Another path for future work can be to study how the recommendations provided can be applicable or if they are useful in other ecosystems. This, considering that in, for example, larger communities, some of our recommendations could be not applicable.
- *Willingness to participate in answering questions*: the approach of TMBA was to better distribute the workload of answering questions by recommending less participatory users. Although, there is something we do not know: are such kinds of users willing to participate more? A study that tackles this question would be benefited from the fifth recommendation which is related to motivating the community to contribute.
- *Distribution workload in maintenance tasks*: one of the approaches mentioned in the recommendations was to divide the maintenance tasks of abandoned packages into smaller tasks. Thus, with an appropriate expert recommender system, qualified members of the ecosystem could participate, for example: by replicating bug reports or reviewing pull requests, lowering the workload of the maintainer, or actually maintaining an abandoned package.

Bibliography

[1] S. Ceriani, M. Migliavacca, Middleware in robotics (2013) 1–10.

[2] A. Afzal, C. L. Goues, M. Hilton, C. S. Timperley, A study on challenges of testing robotic systems, in: 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST), 2020, pp. 96–107. `doi:10.1109/ICST46399.2020.00020`.

[3] S. Kolak, A. Afzal, C. Le Goues, M. Hilton, C. S. Timperley, It takes a village to build a robot: An empirical study of the ros ecosystem, in: 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2020, pp. 430–440.

[4] S. Arumugam, R. Kumar, A. R. Prasad, Wireless Robotics : Opportunities and Challenges, Wireless Personal Communications 70 (2013) 1033–1058. `doi:10.1007/s11277-013-1102-3`.

[5] X. Wang, C. Huang, L. Yao, B. Benatallah, M. Dong, A survey on expert recommendation in community question answering, Journal of Computer Science and Technology 33 (4) (2018) 625–653.

[6] N. Nikzad-Khasmakhi, M. A. Balafar, M. Reza Feizi–Derakhshi, The state-of-the-art in expert recommendation systems, Engineering Applications of Artificial Intelligence 82 (February) (2019) 126–147. `doi:10.1016/j.engappai.2019.03.020`.
URL `https://doi.org/10.1016/j.engappai.2019.03.020`

[7] N. Eghbal, Working in public: The making and maintenance of open source software, Stripe Press, 2020.

[8] T. Lozano-Perez, Robot programming, Proceedings of the IEEE 71 (7) (1983) 821–841. `doi:10.1109/PROC.1983.12681`.

[9] A. Thompson, Evolving electronic robot controllers that exploit hardware resources, Vol. 929, Springer Berlin Heidelberg, 1995. `doi:10.1007/3-540-59496-5{\_}332`.
URL `http://scholar.google.com/scholar?hl=en{&}btnG=Search{&}q=intitle:No+Title{#}0$\backslash$nhttp://link.springer.com/chapter/10.1007/3-540-59496-5{_}332$\backslash$n<GotoISI>://WOS:A1995BF02F00048`

[10] S. Thrun, W. Burgard, D. Fox, Probabilistic robotics, Vol. 45, 2005. `doi:10.1145/`

504729.504754.

[11] D. Bakken, Middleware, Encyclopedia of Distributed Computing (2002).
URL http://www.cs.ucr.edu/vana/cs253/papers/middleware-bakken.pdf

[12] N. Mohamed, J. Al-Jaroodi, I. Jawhar, Middleware for robotics: A survey, 2008 IEEE International Conference on Robotics, Automation and Mechatronics, RAM 2008 (2008) 736–742doi:10.1109/RAMECH.2008.4681485.

[13] A. Elkady, T. Sobh, Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography, Journal of Robotics 2012 (2012) 1–15. doi:10.1155/2012/959013.

[14] M. Quigley, K. Conley, B. Gerkey, J. FAust, T. Foote, J. Leibs, E. Berger, R. Wheeler, A. Mg, ROS: an open-source Robot Operating System (2009).
URL http://www.willowgarage.com/sites/default/files/icraoss09-ROS.pdf

[15] H. Utz, S. Sablatnög, S. Enderle, G. Kraetzschmar, Miro - Middleware for mobile robot applications, IEEE Transactions on Robotics and Automation 18 (4) (2002) 493–497. doi:10.1109/TRA.2002.802930.

[16] J. Jackson, Microsoft robotics studio: A technical introduction, IEEE Robotics and Automation Magazine 14 (4) (2007) 82–87. doi:10.1109/M-RA.2007.905745.

[17] C. Côté, Y. Brosseau, D. Létourneau, C. Raïevsky, F. Michaud, Robotic software integration using MARIE, International Journal of Advanced Robotic Systems 3 (1) (2006) 055–060. doi:10.5772/5758.

[18] I. a. D. Nesnas, R. Simmons, D. Gaines, C. Kunz, A. Diazcalderon, T. Estlin, R. Madison, J. Guineau, M. McHenry, I. H. Shu, D. Apfelbaum, CLARAty: Challenges and steps toward reusable robotic software, International Journal of Advanced Robotic Systems 3 (1) (2006) 023–030. doi:10.5772/5766.

[19] E. Hourdakis, G. Chliveros, P. Trahanias, ORCA - a physics-based robotics simulation environment that supports distributed systems development, Journal of Software Engineering for Robotics 5 (September) (2014) 13–24.

[20] P. Fitzpatrick, G. Metta, L. Natale, Towards long-lived robot genes, Robotics and Autonomous systems 56 (1) (2008) 29–45.

[21] M. E. Munich, J. Ostrowski, P. Pirjanian, ERSP: A software platform and architecture for the service robotics industry, 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS (2005) 3225–3232doi:10.1109/IROS.2005.1545468.

[22] N. Ando, T. Suehiro, T. Kotoku, A software platform for component based rt-system development: OpenRTM-Aist, Simulation, Modeling, and Programming for Autonomous Robots (2008) 87–98doi:10.1007/978-3-540-89076-8-12.

[23] N. Ando, T. Suehiro, K. Kitagaki, T. Kotoku, W. K. Yoon, RT-Middleware: Dis-

tributed component middleware for RT (Robot Technology), 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS (2005) 3555–3560doi:10.1109/IROS.2005.1545521.

[24] O. Michel, Webots TM : Professional Mobile Robot Simulation, International Journal of Advanced Robotic Systems 1 (1) (2004) 39–42. doi:10.1.1.86.1278.

[25] B. P. Gerkey, R. T. Vaughan, A. Howard, The Player / Stage Project : Tools for Multi-Robot and Distributed Sensor Systems, Proceedings of the International Conference on Advanced Robotics (ICAR 2003) (Icar) (2003) 317–323. doi:10.1.1.10.491.
URL http://robotics.usc.edu/{~}gerkey/research/final{_}papers/icar03-player.pdf

[26] H. Bruyninckx, Open robot control software: the OROCOS project, Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164) 3 (2001) 2523–2528. doi:10.1109/ROBOT.2001.933002.

[27] S. C. Ahn, J. H. Kim, K. Lim, H. Ko, Y.-M. Kwon, H.-G. Kim, UPnP approach for robot middleware, in: Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on, 2005, pp. 1959–1963. doi:10.1109/ROBOT.2005.1570400.

[28] S. Magnenat, P. Rétornaz, M. Bonani, V. Longchamp, F. Mondada, ASEBA: A modular architecture for event-based control of complex robots, IEEE/ASME Transactions on Mechatronics 16 (2) (2011) 321–329. doi:10.1109/TMECH.2010.2042722.

[29] M. Broxvall, B.-S. Seo, W. Kwon, The peis kernel: A middleware for ubiquitous robotics, in: In In Proc. of the IROS Workshop on Ubiquitous Robotic Space Design and Applications, 2007, pp. 212–218.
URL http://aass.oru.se/{~}asaffio/Papers/Files/Files/lab/mathias{_}iros07.pdf

[30] Y. Jonghun, K. Saehwa, H. Seongsoo, The Robot Software Communications Architecture (RSCA): QoS-aware middleware for networked service robots, 2006 SICE-ICASE International Joint Conference (2006) 330–335doi:10.1109/SICE.2006.315702.

[31] P. Gil, Data centric middleware for the integration of wireless sensor networks and mobile robots, Group (2007).

[32] T.-H. K. T.-H. Kim, S.-H. C. S.-H. Choi, J.-H. K. J.-H. Kim, Incorporation of a Software Robot anda Mobile Robot Using a Middle Layer (2007). doi:10.1109/TSMCC.2007.905850.
URL http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=4343989

[33] M. Friedmann, J. Kiener, S. Petters, D. Thomas, O. von Stryk, Modular software architecture for teams of cooperating, heterogeneous robots, in: Proc. IEEE International Conference on Robotics and Biomimetics (ROBIO), 2006, pp. 613–618. doi:10.1109/ROBIO.2006.340270.

[34] Orocos Real-Time Toolkit, `http://www.orocos.org/rtt`.
URL `http://www.orocos.org/rtt`

[35] K. Gadeyne, BFL: Bayesian filtering library, `http://www.orocos.org/bfl` (2001).
URL `http://www.orocos.org/bfl`

[36] R. E. Kalman, A New Approach to Linear Filtering and Prediction Problems, Transactions of the ASME Journal of Basic Engineering 82 (Series D) (1960) 35–45. `doi:10.1115/1.3662552`.

[37] P. Del Moral, Nonlinear filtering: Interacting particle resolution, Comptes Rendus de l'Académie des Sciences - Series I - Mathematics 325 (6) (1997) 653–658. `doi:10.1016/S0764-4442(97)84778-7`.

[38] MadeInGermany project, AutoNOMOS Labs, `http://autonomos-labs.com/vehicles/made-in-germany/`.
URL `http://autonomos-labs.com/vehicles/made-in-germany/`

[39] Orca Robotics - History, `http://orca-robotics.sourceforge.net/orca_doc_history.html`.
URL `http://orca-robotics.sourceforge.net/orca{_}doc{_}history.html`

[40] Common Object Request Broker Architecture (CORBA), `http://www.corba.org/`.
URL `http://www.corba.org/`

[41] ZeroC - The Internet Communications Engine (Ice), `https://zeroc.com/ice.html`.
URL `https://zeroc.com/ice.html`

[42] E. Gamma, R. Helm, R. E. Johnson, J. Vlissides, Design patterns: elements of reusable object-oriented software, Design 206 (1995) 395. `doi:10.1093/carcin/bgs084`.
URL `http://www.cs.up.ac.za/cs/aboake/sws780/references/patternstoarchitecture/Gamma-DesignPatternsIntro.pdf`

[43] iCub.org - an open source cognitive humanoid robotic platform, `http://www.icub.org/`.
URL `http://www.icub.org/`

[44] R. B. Rusu, S. Cousins, 3D is here: Point Cloud Library (PCL), Proceedings - IEEE International Conference on Robotics and Automation (2011) 1–4`doi:10.1109/ICRA.2011.5980567`.
URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5980567`

[45] G. Bradski, The OpenCV Library, Dr Dobbs Journal of Software Tools 25 (2000) 120–125. `doi:10.1111/0023-8333.50.s1.10`.
URL `http://opencv.willowgarage.com`

[46] Turtlebot robot, `http://www.turtlebot.com`.
URL `http://www.turtlebot.com`

[47] PR2: Personal Robot 2, `https://www.willowgarage.com/pages/pr2/overview`.
URL `https://www.willowgarage.com/pages/pr2/overview`

[48] Towards Milestone 3, Willow Garage, `https://www.willowgarage.com/blog/2009/08/13/towards-milestone-3`.
URL `https://www.willowgarage.com/blog/2009/08/13/towards-milestone-3`

[49] G. Walck, U. Cupcic, T. O. Duran, V. Perdereau, A Case Study of ROS Software Re-usability for Dexterous In-Hand Manipulation 5 (May) (2014) 36–47.

[50] HANDLE Project Website, `http://www.handle-project.eu/`.
URL `http://www.handle-project.eu/`

[51] G. Walck, U. Cupcic, T. O. Duran, V. Perdereau, A case study of ros software re-usability for dexterous in-hand manipulation, Journal of Software Engineering for Robotics 5 (1) (2014).

[52] M. Lungu, L. Michele, T. Gîrba, R. Robbes, The Small Project Observatory: Visualizing software ecosystems, Science of Computer Programming 75 (4) (2010) 264–275. `doi:10.1016/j.scico.2009.09.004`.
URL `http://dx.doi.org/10.1016/j.scico.2009.09.004`

[53] D. Coleman, I. Sucan, S. Chitta, N. Correll, Reducing the Barrier to Entry of Complex Robotic Software: a MoveIt! Case Study, arXiv preprint arXiv:1404.3785 5 (May) (2014) 3–16. `arXiv:arXiv:1404.3785v1`.
URL `http://arxiv.org/abs/1404.3785`

[54] E. Ceseracciu, D. Domenichelli, P. Fitzpatrick, G. Metta, L. Natale, A. Paikan, A middle way for robotics middleware 5 (September) (2013) 42–49.

[55] A. Hora, R. Robbes, N. Anquetil, A. Etien, S. Ducasse, M. T. Valente, How Do Developers React to API Evolution ? The Pharo Ecosystem Case, 31st IEEE International Conference on Software Maintenance 2 (2015) 10.

[56] P. Estefo, R. Robbes, J. Fabry, Code duplication in ROS launchfiles, in: Chilean Computer Science Society (SCCC), 2015 34th International Conference of the, IEEE, 2015, pp. 1–6.

[57] Automated Driving With ROS At BMW., `http://roscon.ros.org/2015/presentations/ROSCon-Automated-Driving.pdf`.
URL `http://roscon.ros.org/2015/presentations/ROSCon-Automated-Driving.pdf`

[58] C. K. Roy, J. R. Cordy, A Survey on Software Clone Detection Research, Queen's School of Computing TR 115 (2007) 115. `doi:10.1.1.62.7869`.
URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.62.7869$\backslash$nhttp://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.67.9931&rep=rep1&type=pdf`

[59] A. Lakhotia, J. Li, A. Walenstein, Y. Yang, Towards a clone detection benchmark suite and results archive, in: Program Comprehension, 2003. 11th IEEE International Workshop on, IEEE, 2003, pp. 285–286.

[60] H. A. Basit, D. C. Rajapakse, S. Jarzabek, Beyond templates: a study of clones in the stl and some general implications, in: Proceedings of the 27th international conference on Software engineering, ACM, 2005, pp. 451–459.

[61] J. Mayrand, C. Leblanc, E. M. Merlo, Experiment on the automatic detection of function clones in a software system using metrics, in: Software Maintenance 1996, Proceedings., International Conference on, IEEE, 1996, pp. 244–253.

[62] K. A. Kontogiannis, R. DeMori, E. Merlo, M. Galler, M. Bernstein, Pattern matching for clone and concept detection, in: Reverse engineering, Springer, 1996, pp. 77–108.

[63] Z. Li, S. Lu, S. Myagmar, Y. Zhou, Cp-miner: Finding copy-paste and related bugs in large-scale software code, Software Engineering, IEEE Transactions on 32 (3) (2006) 176–192.

[64] A. Monden, D. Nakae, T. Kamiya, S.-i. Sato, K.-i. Matsumoto, Software quality analysis by code clones in industrial legacy software, in: Software Metrics, 2002. Proceedings. Eighth IEEE Symposium on, IEEE, 2002, pp. 87–94.

[65] J. H. Johnson, Identifying redundancy in source code using fingerprints, in: Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: software engineering-Volume 1, IBM Press, 1993, pp. 171–183.

[66] S. García, D. Strüber, D. Brugali, T. Berger, P. Pelliccione, Robotics software engineering: A perspective from the service robotics domain, in: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2020, Association for Computing Machinery, New York, NY, USA, 2020, p. 593–604. `doi:10.1145/3368089.3409743`. URL `https://doi.org/10.1145/3368089.3409743`

[67] M. Reichardt, K. Berns, On Software Quality-motivated Design of a Real-time Framework for Complex Robot Control Systems, Proceedings of the 7th International Workshop on Software Quality and Maintainability (SQM) (2013).

[68] M. Pichler, B. Dieber, M. Pinzger, Can i depend on you? mapping the dependency and quality landscape of ros packages, in: 2019 third IEEE international conference on robotic computing (IRC), IEEE, 2019, pp. 78–85.

[69] D. Kohler, Rosjava, `http://www.ros.org/wiki/rosjava` (may 2012).

[70] B. Marthi, Roslisp, `http://wiki.ros.org/roslisp` (june 2015).

[71] S. Bragagnolo, L. Fabresse, J. Laval, P. Estefó, N. Bouraqadi, Pharos: a ros client for the pharo language, http://car.mines-douai.fr/category/pharos/ (2014). URL `http://car.mines-douai.fr/category/pharos/`

[72] G. Robles, J. M. González-Barahona, D. Izquierdo-Cortazar, I. Herraiz, Tools for the Study of the Usual Data Sources found in Libre Software Projects, International Journal of Open Source Software and Processes 1 (1) (2009) 24–45. doi:10.4018/jossp.2009010102.

[73] S. Ducasse, T. Gîrba, A. Kuhn, L. Renggli, Meta-environment and executable meta-language using Smalltalk: an experience report, Journal of Software and Systems Modeling (SOSYM) 8 (1) (2009) 5–19. doi:10.1007/s10270-008-0081-4.

[74] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, E. Merlo, Comparison and evaluation of clone detection tools, Software Engineering, IEEE Transactions on 33 (9) (2007) 577–591.

[75] P. Estefo, J. Simmonds, R. Robbes, J. Fabry, The robot operating system: Package reuse and community dynamics, Journal of Systems and Software 151 (2019) 226–242.

[76] T. Mens, B. Adams, J. Marsan, Towards an interdisciplinary, socio-technical analysis of software ecosystem health, CEUR Workshop Proceedings 2047 (2017) 7–9. arXiv:1711.04532.

[77] I. Steinmacher, T. U. Conte, C. Treude, M. A. Gerosa, Overcoming open source project entry barriers with a portal for newcomers, in: Proceedings of the 38th International Conference on Software Engineering, ACM, 2016, pp. 273–284.

[78] I. Steinmacher, M. Aurelio, G. Silva, D. F. Redmiles, A systematic literature review on the barriers faced by newcomers to open source software projects, Information and Software Technology 59 (44) (2015) 67–85. doi:10.1016/j.infsof.2014.11.001.

[79] A. Lee, J. C. Carver, A. Bosu, Understanding the Impressions, Motivations, and Barriers of One Time Code Contributors to FLOSS Projects: A Survey, in: 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), IEEE, 2017, pp. 187–197. doi:10.1109/ICSE.2017.25.

[80] T. Mens, M. Goeminne, U. Raja, A. Serebrenik, Survivability of software projects in gnome–a replication study, SATToSE 2014—Pre-proceedings (2014) 79.

[81] J. Khondhu, A. Capiluppi, K.-J. Stol, Is It All Lost? A Study of Inactive Open Source Projects, in: Proceedings of the 9th International Conference on Open Source Systems, Springer, Berlin, Heidelberg, 2013, pp. 61–79.

[82] J. Coelho, M. T. Valente, L. L. Silva, E. Shihab, Identifying unmaintained projects in github, in: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 2018, pp. 1–10.

[83] J. Wallen, What happens when developers leave their open source projects? (Jul 2020). URL https://thenewstack.io/what-happens-when-developers-leave-their-open-source-proje

[84] J. L. C. Izquierdo, V. Cosentino, J. Cabot, An Empirical Study on the Maturity of the Eclipse Modeling Ecosystem, Proceedings - ACM/IEEE 20th International Conference

on Model Driven Engineering Languages and Systems, MODELS 2017 (2017) 292–302doi:10.1109/MODELS.2017.19.

[85] J. Coelho, M. T. Valente, Why modern open source projects fail, in: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ACM, 2017, pp. 186–196.

[86] E. Constantinou, T. Mens, An empirical comparison of developer retention in the RubyGems and npm software ecosystems, Innovations in Systems and Software Engineering 13 (2-3) (2017) 101–115. arXiv:1708.02618, doi:10.1007/s11334-017-0303-4.

[87] M.-A. Storey, A. Zagalsky, Disrupting developer productivity one bot at a time, in: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, ACM, New York, NY, USA, 2016, pp. 928–931. doi:10.1145/2950290.2983989.

[88] G. A. A. Prana, C. Treude, F. Thung, T. Atapattu, D. Lo, Categorizing the content of github readme files, Empirical Software Engineering 24 (3) (2019) 1296–1327.

[89] R. G. Kula, D. M. German, A. Ouni, T. Ishio, K. Inoue, Do developers update their library dependencies?: An empirical study on the impact of security advisories on library migration, Empirical Software Engineering 23 (1) (2018) 384–417. arXiv:1709.04621, doi:10.1007/s10664-017-9521-5.

[90] V. Midha, P. Palvia, Factors affecting the success of Open Source Software, Journal of Systems and Software 85 (4) (2012) 895–905. doi:10.1016/j.jss.2011.11.010.

[91] M. P. Robillard, W. Maalej, R. J. Walker, T. Zimmermann (Eds.), Recommendation Systems in Software Engineering, Springer Berlin Heidelberg, 2014. doi:10.1007/978-3-642-45135-5.

[92] M. Gasparic, A. Janes, What recommendation systems for software engineering recommend: A systematic literature review, Journal of Systems and Software 113 (2016) 101–113.

[93] P. Melville, R. J. Mooney, R. Nagarajan, et al., Content-boosted collaborative filtering for improved recommendations, Aaai/iaai 23 (2002) 187–192.

[94] J. Mateos-Garcia, W. E. Steinmueller, The institutions of open source software: Examining the Debian community, Information Economics and Policy 20 (4) (2008) 333–344. doi:10.1016/j.infoecopol.2008.06.001.

[95] C. Bogart, C. Kästner, J. Herbsleb, F. Thung, How to break an API: cost negotiation and community values in three software ecosystems, in: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2016, ACM Press, 2016, pp. 109–120. doi:10.1145/2950290.2950325.

[96] T. Dal Sasso, A. Mocci, M. Lanza, E. Mastrodicasa, How to gamify software engineer-

ing, in: Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on, IEEE, 2017, pp. 261–271.

[97] D. J. Dubois, G. Tamburrelli, Understanding gamification mechanisms for software development, in: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2013, 2013, p. 659. doi:10.1145/2491411.2494589.

[98] B. Vasilescu, V. Filkov, A. Serebrenik, StackOverflow and GitHub: Associations between software development and crowdsourced knowledge, in: Proceedings - SocialCom/PASSAT/BigData/EconCom/BioMedCom 2013, 2013, pp. 188–195. doi:10.1109/SocialCom.2013.35.

[99] B. Vasilescu, Human aspects, gamification, and social media in collaborative software engineering, in: Companion Proceedings of the 36th International Conference on Software Engineering, ACM, 2014, pp. 646–649.

[100] S. Grant, B. Betts, Encouraging user behaviour with achievements: an empirical study, in: Proceedings of the 10th Working Conference on Mining Software Repositories, IEEE Press, 2013, pp. 65–68.

[101] A. Atiq, N. Zealand, Monetary Rewards for Open Source Software Developers, in: ICIS-RP, 2014, pp. 1–10.

[102] S. Gosain, The Impact of Ideology on Effectiveness in Open Source Software Development Teams, MIS Quarterly 30 (2) (2006) 291. doi:10.2307/25148732.

[103] G. Avelino, E. Constantinou, M. T. Valente, A. Serebrenik, On the abandonment and survival of open source projects: An empirical investigation, in: 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), IEEE, 2019, pp. 1–12.

[104] H. Cavusoglu, Z. Li, K.-W. Huang, Can gamification motivate voluntary contributions? the case of stackoverflow q&a community, in: Proceedings of the 18th ACM conference companion on computer supported cooperative work & social computing, 2015, pp. 171–174.

[105] B. Li, I. King, Routing questions to appropriate answerers in community question answering services, in: Proceedings of the 19th ACM international conference on Information and knowledge management, 2010, pp. 1585–1588.

[106] T. Claburn, Nice people matter? npm may stand for not politely managed – job cuts leave staff sore (Apr 2019).
URL https://www.theregister.com/2019/04/01/npm_layoff_staff

[107] M. Neshati, Z. Fallahnejad, H. Beigy, On dynamicity of expert finding in community question answering, Information Processing & Management 53 (5) (2017) 1026 – 1042. doi:https://doi.org/10.1016/j.ipm.2017.04.002.
URL http://www.sciencedirect.com/science/article/pii/S0306457316305386

[108] M. Z. Al-Taie, S. Kadry, A. I. Obasa, Understanding expert finding systems: domains and techniques, Social Network Analysis and Mining 8 (1) (2018) 57.

[109] N. Nikzad-Khasmakhi, M. Balafar, M. R. Feizi-Derakhshi, The state-of-the-art in expert recommendation systems, Engineering Applications of Artificial Intelligence 82 (2019) 126–147.

[110] Z. Roozbahani, J. Rezaeenour, H. Emamgholizadeh, A. Jalaly Bidgoly, A systematic survey on collaborator finding systems in scientific social networks, Knowledge and Information Systems 62 (10) (2020) 3837–3879.

[111] P. Sumanth, K. Rajeshwari, Discovering top experts for trending domains on stack overflow, Procedia computer science 143 (2018) 333–340.

[112] Z. Zhao, Q. Yang, D. Cai, X. He, Y. Zhuang, Expert finding for community-based question answering via ranking metric network learning., in: Ijcai, Vol. 16, 2016, pp. 3000–3006.

[113] D. Movshovitz-Attias, Y. Movshovitz-Attias, P. Steenkiste, C. Faloutsos, Analysis of the reputation system and user contributions on a question answering website: Stackoverflow, in: 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2013), IEEE, 2013, pp. 886–893.

[114] L. Tóth, B. Nagy, D. Janthó, L. Vidács, T. Gyimóthy, Towards an accurate prediction of the question quality on stack overflow using a deep-learning-based nlp approach., in: ICSOFT, 2019, pp. 631–639.

[115] H. Alharthi, D. Outioua, O. Baysal, Predicting questions' scores on stack overflow, in: 2016 IEEE/ACM 3rd International Workshop on CrowdSourcing in Software Engineering (CSI-SE), IEEE, 2016, pp. 1–7.

[116] L. Wang, B. Wu, J. Yang, S. Peng, Personalized recommendation for new questions in community question answering, in: 2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), IEEE, 2016, pp. 901–908.

[117] W. Huang, W. Mo, B. Shen, Y. Yang, N. Li, Cpdscorer: Modeling and evaluating developer programming ability across software communities., in: SEKE, 2016, pp. 87–92.

[118] E. Constantinou, G. M. Kapitsaki, Identifying developers' expertise in social coding platforms, in: 2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), IEEE, 2016, pp. 63–67.

[119] Y. Xiong, Z. Meng, B. Shen, W. Yin, Mining developer behavior across github and stackoverflow., in: SEKE, 2017, pp. 578–583.

[120] B. Yang, S. Manandhar, Tag-based expert recommendation in community question answering, in: 2014 IEEE/ACM International Conference on Advances in Social Networks

Analysis and Mining (ASONAM 2014), IEEE, 2014, pp. 960–963.

[121] F. Calefato, F. Lanubile, N. Novielli, How to ask for technical help? evidence-based guidelines for writing questions on stack overflow, Information and Software Technology 94 (2018) 186–207.

[122] S. L. Vadlamani, O. Baysal, Studying software developer expertise and contributions in stack overflow and github, in: 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2020, pp. 312–323.

[123] B. Vasilescu, V. Filkov, A. Serebrenik, Stackoverflow and github: Associations between software development and crowdsourced knowledge, in: 2013 International Conference on Social Computing, IEEE, 2013, pp. 188–195.

[124] C. Liu, Y. Hao, W. Shan, Z. Dai, Identifying experts in community question answering website based on graph convolutional neural network, IEEE Access 8 (2020) 137799–137811.

[125] X. Zhang, T. Wang, G. Yin, C. Yang, H. Wang, Who will be interested in? a contributor recommendation approach for open source projects., in: SEKE, 2017, pp. 363–369.

[126] J. Hahn, J. Y. Moon, C. Zhang, Emergence of new project teams from open source software developer networks: Impact of prior collaboration ties, Information Systems Research 19 (3) (2008) 369–391.

[127] G. Madey, V. Freeh, R. Tynan, The open source software development phenomenon: An analysis based on social network theory (2002).

[128] J. Ramos, et al., Using tf-idf to determine word relevance in document queries, in: Proceedings of the first instructional conference on machine learning, Vol. 242, Citeseer, 2003, pp. 29–48.

[129] F. M. Dekking, C. Kraaikamp, H. P. Lopuhaa, L. E. Meester, A modern introduction to probability and statistics, 1st Edition, Springer Texts in Statistics, Springer, London, England, 2005.

Annexes

Annex A

Interview Questionnaire

Part 1: Background and general opinion about ROS

1. What is your graduate and/or postgraduate degree / what are you studying?
2. How long have you been programming with ROS?
3. How do you mainly use ROS? Or contribute to ROS?
4. Do you have a software engineering background? For you, what is Software Engineering?
   (a) Does this apply to ROS?
5. What are the key strengths of ROS in your opinion?
6. What are the principal weaknesses / areas of improvement of ROS?
7. How does ROS compare with the alternatives? Do you know any? Which ones?
8. Do you usually spend more time developing new modules, configuring existing modules or adapting 3rd party modules?
9. What are in your opinion the most common errors when using ROS?

Part 2: Learning curve and first steps

10. How much time do you think someone needs to reach an acceptable expertise in ROS?
11. Which activities are key for developing that expertise?
12. What do you do when you face an unknown bug or behaviour of your program?
    (a) Do you ask in a mailing list?
    (b) Do you ask in ROS Answers?
    (c) Do you go to the documentation of the relevant packages?
    (d) Do you visit the Github page of the relevant packages?
13. What do you think about the mechanisms of communication provided by the ROS community? (ROS Answers, discourse.ros.org, Mailing lists, ROS Wiki, Github pages . . .)
14. What do you think about the documentation of ROS packages? Their tutorials, examples, explanations, demos, etc.

Part 3: Experiences using ROS

15. Describe the latest hard to find bug you had to fight with while using ROS? Is this a usual kind of bug?
16. ROS as a middleware and a framework provides mechanisms for developing applications. Have you ever experienced that you have noticed a lack of a solution or that you had to somehow bypass a ROS solution (or hacking a ROS solution) due to it not fulfilling your requirements?

17. What has been the most . . . in your experience working with ROS?
    (a) Boring or repetitive task or tool feature.
    (b) Easy, straightforward tasks.
    (c) "Magical" task: you do it because you know it is necessary but you do not know what is it for. If anything goes wrong, you would not know how to fix it.
    (d) Good quality tool.
18. Have you ever experienced issues due to the upgrade of a ROS distribution?
19. In your experience, typically how many modules you interact with?
20. Have you experienced issues configuring a set of modules? Can you give examples?
21. Have you had issues of hard to understand launch files? Can you give examples?
22. ROS has many types of files (launch files, yaml, package .xml files, other .xml files, build files, etc.).
    (a) Can you explain to me the role of each of the files?
    (b) What are the 3 more frequently used files?

Part 4: Communication Mechanisms

This section asks questions about the five communication mechanisms provided by ROS (see Sect. 2.1).

23. Which one have you used?
24. Which one have you never heard of / used?
25. In which contexts would you use each one?
26. Have you encountered issues using the ROS communication mechanisms? Can you give examples?
27. What are the main constraints or concerns when choosing between mechanisms?
28. Do you combine communication mechanisms? How?
29. Have you ever used another communication mechanism or a custom one?

Part 5: Software Artifacts & Developer Roles

30. What are the common tasks of a developer when programming a robot behaviour? Please list all the examples you can.
31. Can you group these tasks into a role of a developer?
32. For all roles, how many people play this role? (only 1, 2 or 3, everyone)
33. Which software artifacts are related to each task?

Annex B

Survey Questionnaire

Part 1: Demographic and basic information

1. Select which institution you are affiliated with:
   (a) University
   (b) Private sector
   (c) Research Center
   (d) I am not affiliated with an institution
   (e) Non-governmental Organization (NGO)
   (f) School
   (g) Local government agency (eg. municipality, ministry)
   (h) Other, please specify
2. Country of your institution (Leave it blank if you are not affiliated with an institution).
3. What is your background in robotics? For example: computer vision, simulation, navigation, etc.
4. How experienced are you with ROS?
   (a) Between 0 to 6 months
   (b) From 6 months to 1 year
   (c) Up to 2 years
   (d) Up to 3 years
   (e) Up to 4 years
   (f) Up to 5 years or more
5. Do you have any background in Software Engineering?
   (a) Not familiar with the term "Software Engineering"
   (b) Heard about SE, but don't know the definition
   (c) Know what it is, but no background in this area
   (d) Have taken some Software Engineering courses
   (e) Have a strong background in Software Eng.

Part 2: General Background

6. Have you tried reusing a 3rd party ROS package?
   (a) Yes
   (b) No
7. How dependent are your projects to 3rd party ROS packages?
   (a) Only on core packages

    (b) Few dependencies
    (c) Some dependencies
    (d) Major dependencies

8. How often do you encounter the "Abandoned Package" phenomenon in practice?
    (a) Never
    (b) Rarely
    (c) Sometimes
    (d) Very often
    (e) All the time

Part 3: Reusing Packages

9. Have you ever failed to reuse a 3rd party ROS package?
    (a) Yes
    (b) No
10. Why did you fail reusing it? (Leave it blank if you have not failed reusing)
    (a) The package was for an outdated ROS distribution.
    (b) I could not figure out how to use it (lack of documentation).
    (c) There was a bug that prevented the package from working properly.
    (d) I did not succeed in configuring the package for my use case (launch files, etc.) I could not install the package.
    (e) The package is not compatible with my particular hardware.
    (f) The package was for a newer version of ROS, mine is older.
    (g) The package had performance issues.
    (h) I asked for help but could not find it.
    (i) Other, please specify.
11. How often do you find tests in 3rd party ROS packages?
    (a) Never
    (b) Rarely
    (c) Occasionally
    (d) Frequently
    (e) Very frequently
12. How often do you find ROS Bags in 3rd party ROS packages?
    (a) Never
    (b) Rarely
    (c) Occasionally
    (d) Frequently
    (e) Very frequently
13. How relevant to you are the following types of documentation when reusing a 3rd party ROS package?
    - General documentation about the purpose of the package and its features.

- API documentation.
- Guidelines for configuring the package launch files.
- Troubleshooting experiences from other users.
- Issue tracker information.
- Benchmarks and/or information about its performance.
- Information about the robot(s) where it has been tested on.
- Other.

For each type of documentation, indicate if it is:

(a) Not relevant
(b) Hardly relevant
(c) Somewhat relevant
(d) Relevant
(e) Highly relevant
(f) I do not know

14. When reusing a package, where do you get help? Check all that apply:
    (a) ROS Answers.
    (b) ROS Wiki.
    (c) I search for examples on Github.
    (d) StackOverflow.
    (e) Colleagues.
    (f) I ask the package maintainers on Github.
    (g) Mailing list.
    (h) Discourse.
    (i) Other, please specify.
15. Which of the above options do you use most frequently? (Same alternatives as in #14)
16. How dependent are your projects to 3rd party ROS packages?
    (a) Only on core packages.
    (b) Few dependencies.
    (c) Some dependencies.
    (d) Major dependencies.

Part 4: Contributions

17. What type of contribution have you made to 3rd party ROS packages? (Leave it blank if you have not made a contribution)
    (a) Reported a bug.
    (b) Question post in ROS Answers.
    (c) Submitted a bug fix.
    (d) Answer post in ROS Answers.
    (e) Asked for a new feature.

(f) Submitted a new feature.
(g) Added missing documentation.
(h) Updated documentation.
(i) Experience report post.
(j) Another type of contribution.

18. Which of the above options of type of contribution is the most frequent? (Same alternatives as in #17)
19. Have you ever failed to make a contribution to a 3rd party ROS package? If so, why? (Leave it blank if you have not)
    (a) I did not have enough time to make a contribution.
    (b) I think my issue is too specific to my project, hardware and/or research.
    (c) I was not confident that the contribution was good enough (in terms of correctness and/or quality).
    (d) The package was abandoned, so no one will receive or integrate it.
    (e) I did not know how to make a contribution (I am not aware of a workflow).
    (f) I forgot.
    (g) It might have been a misunderstanding on my part.
    (h) I was not sure who to send the contribution to I did not know which software license to use.
    (i) My institution/contract has privacy policies that forbid me from making contributions to open-source projects or sharing work funded by it.
    (j) I did not know you could contribute to packages.
    (k) I did not know how to use GIT or Github well enough to send my contribution (e.g. pull requests).
    (l) The maintainer or other users probably fixed the problem already.
    (m) Another reason, please specify.
20. Which of the above reasons is the most frequent? (Same alternatives as in #19)

Part 5: Support-like contributions

21. Have you ever felt like you could have answered a question on a Q&A site, Mailing list or forum, but in the end you did not submit an answer?
    (a) Yes
    (b) No
22. Why did not you do submit an answer? (If you answered No to question #21, leave it blank)
    (a) It takes too much time to replicate the situation described in question.
    (b) It requires follow-up questions before answering.
    (c) It requires setup data that usually is not provided (ROS Bags, etc.).
    (d) It requires hardware I do not have access to.
    (e) Another reason, please specify.

Annex C

Recall@k figures

In the following pages we show the figures of the median and average of the Recall values for at every place in the TMBA ranking for scenarios A, B, C and D. The blue line shows the median and the red the average. The best performance is shown then the blue and the red lines grow fast: the blue curves achieves the value 1 in the first places of the ranking and the red curve's tilt is more vertical at the beginning.

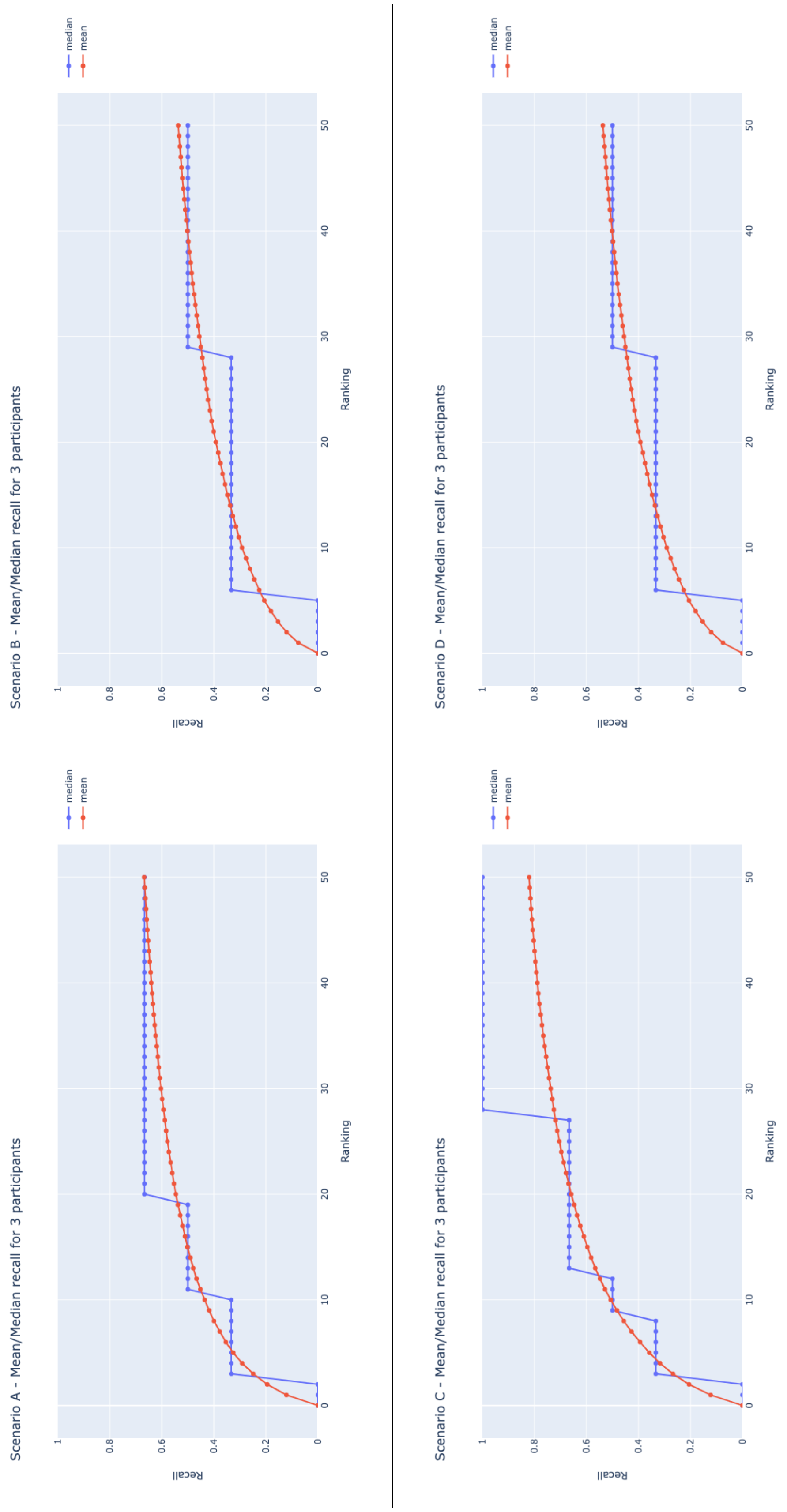


Figure C.1: Recall for 3 participants

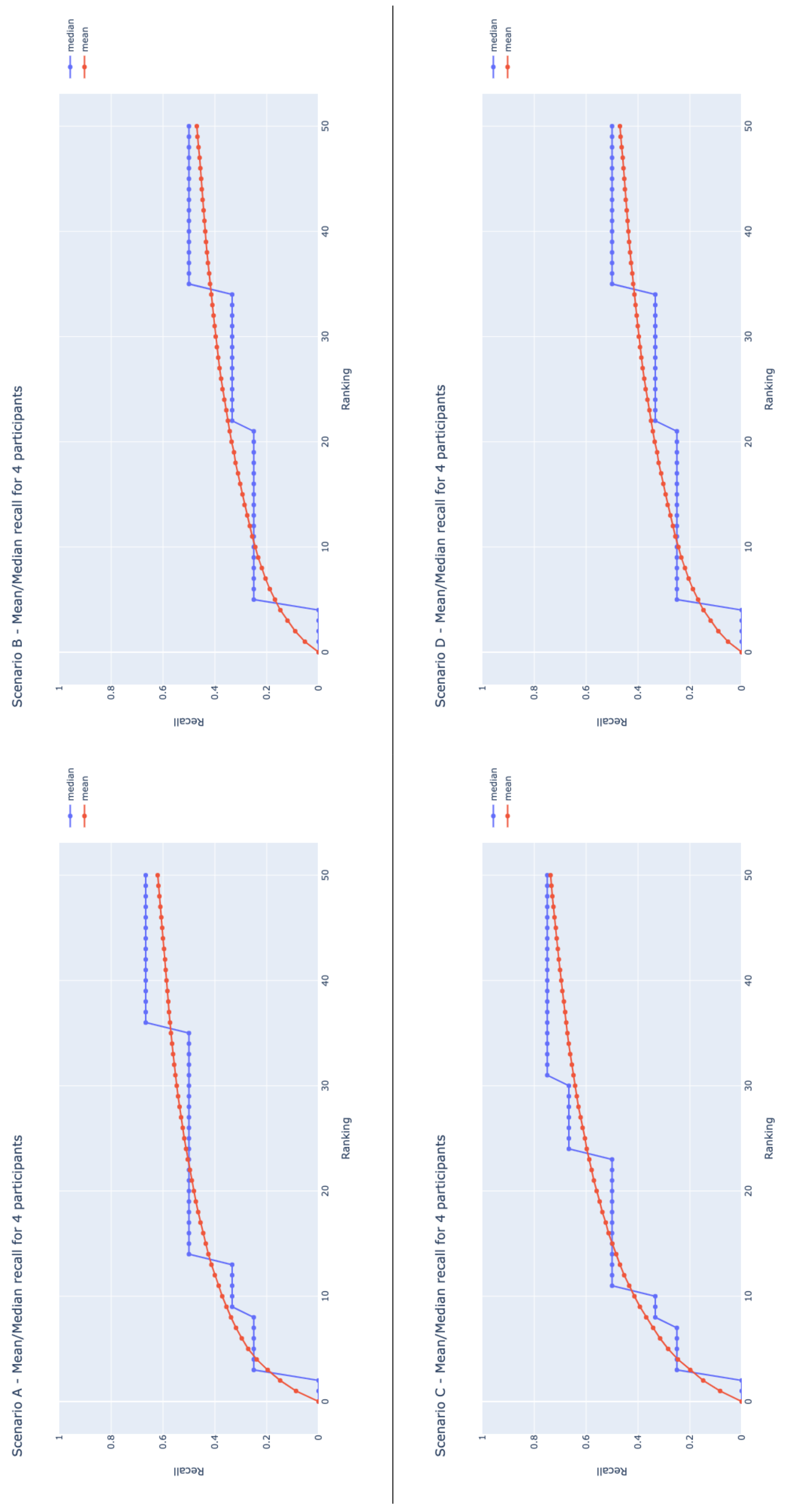


Figure C.2: Recall for 4 participants

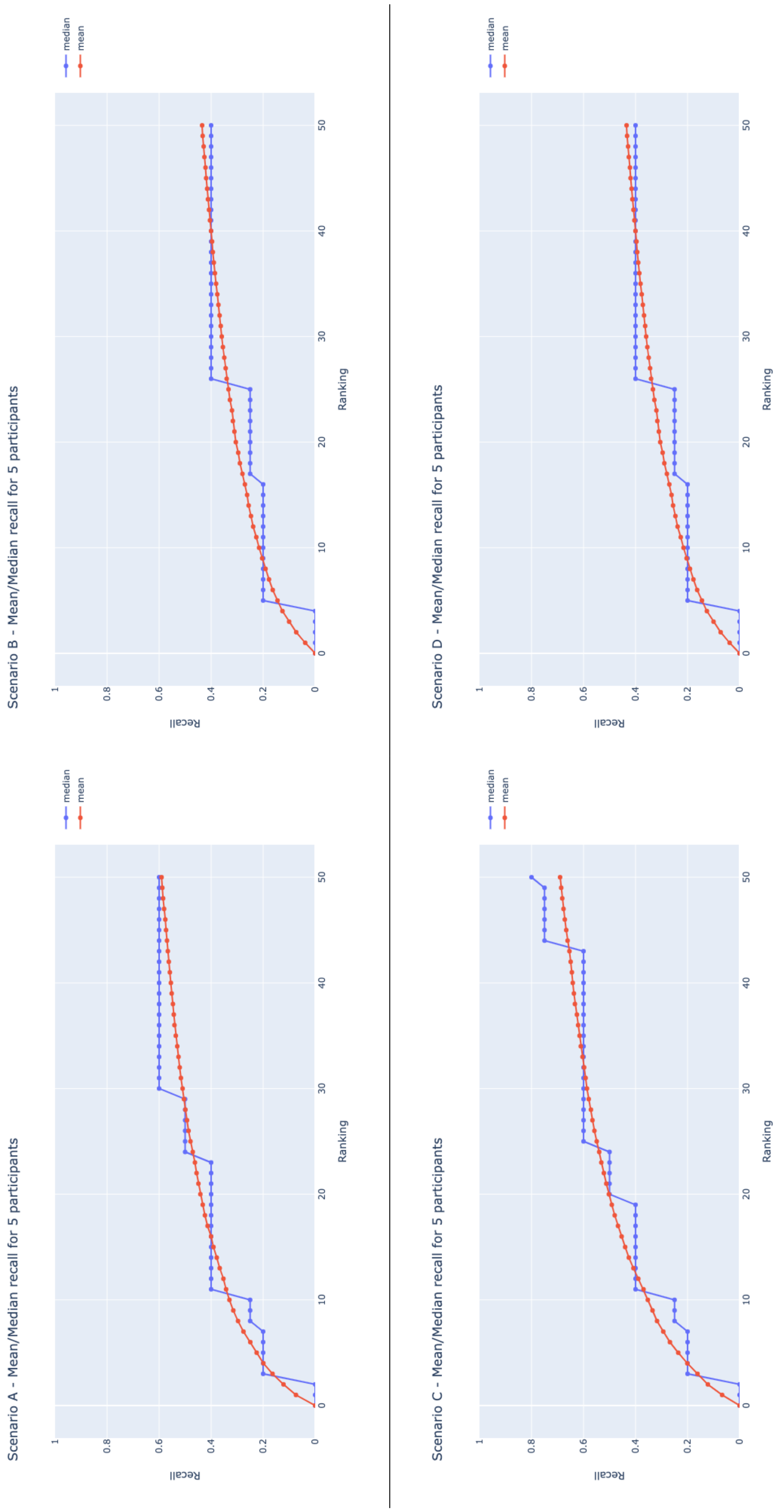


Figure C.3: Recall for 5 participants

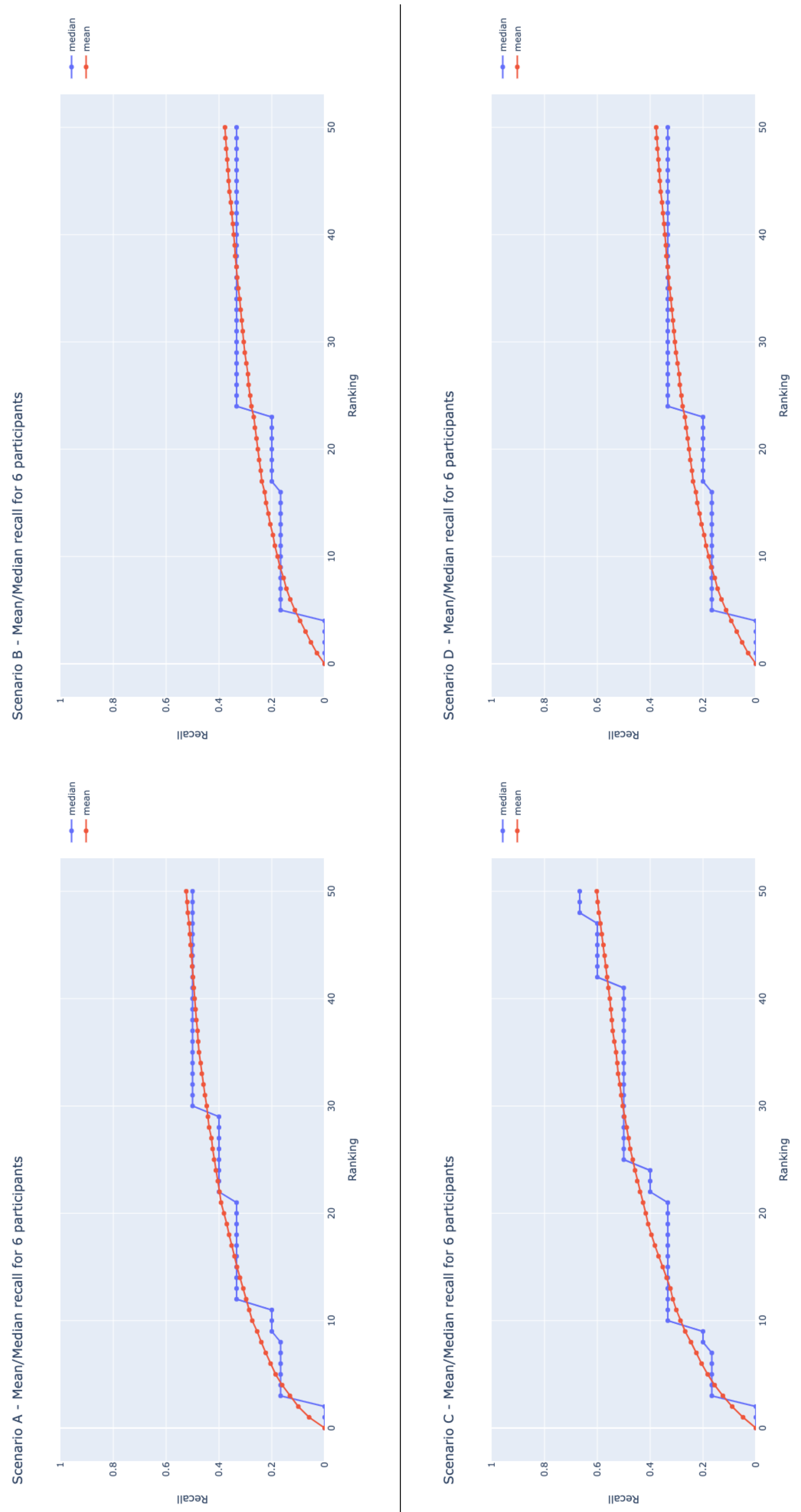


Figure C.4: Recall for 6 participants

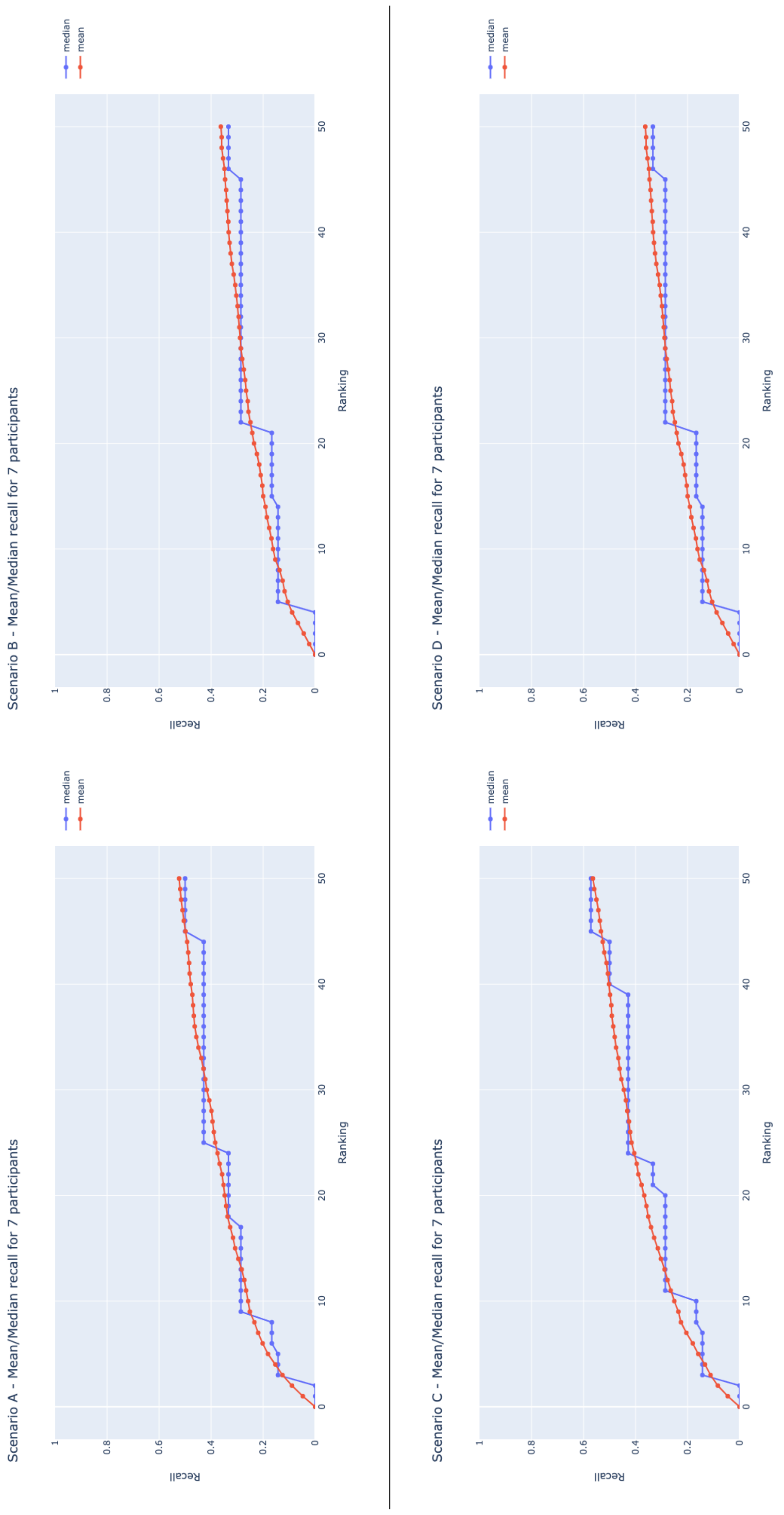


Figure C.5: Recall for 7 participants

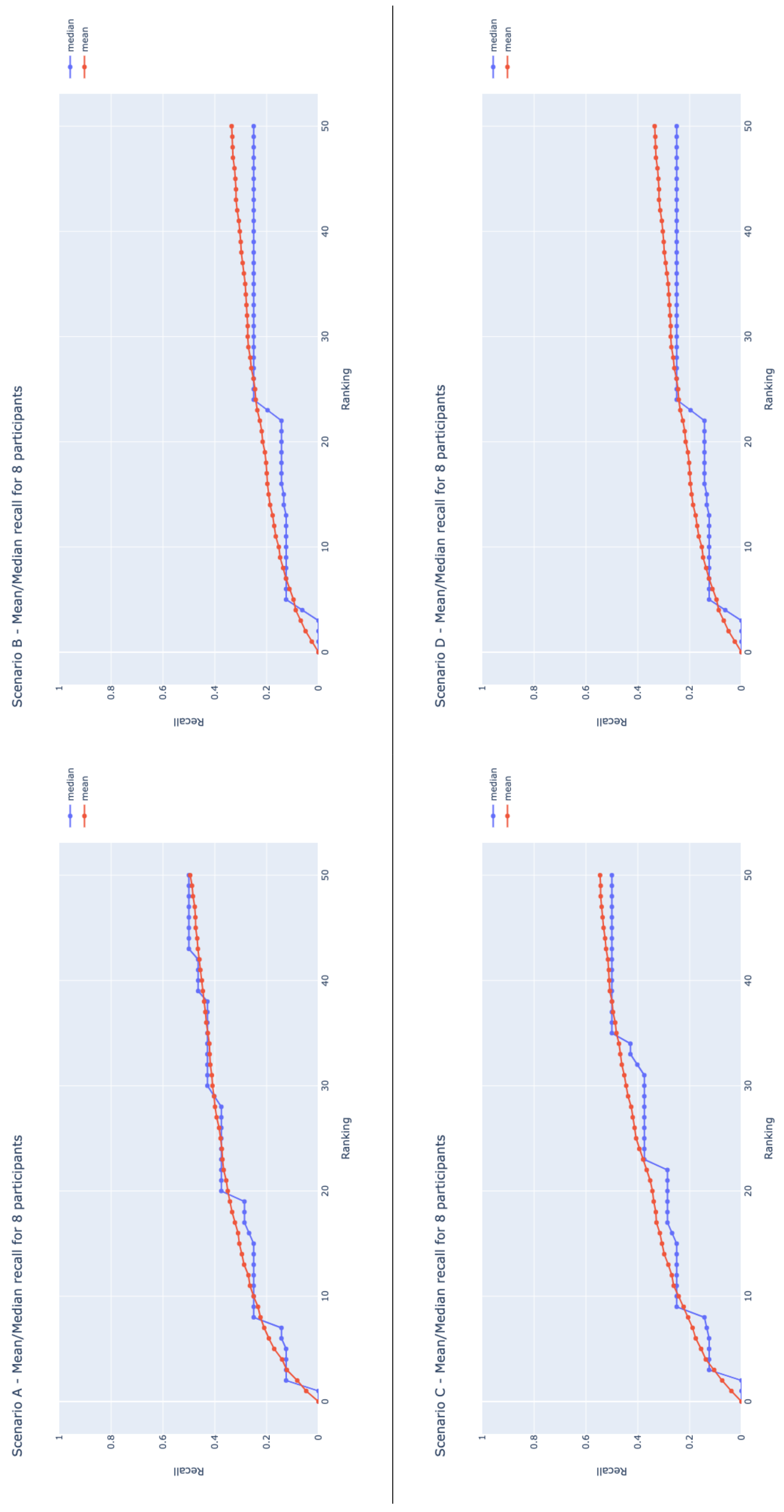


Figure C.6: Recall for 8 participants

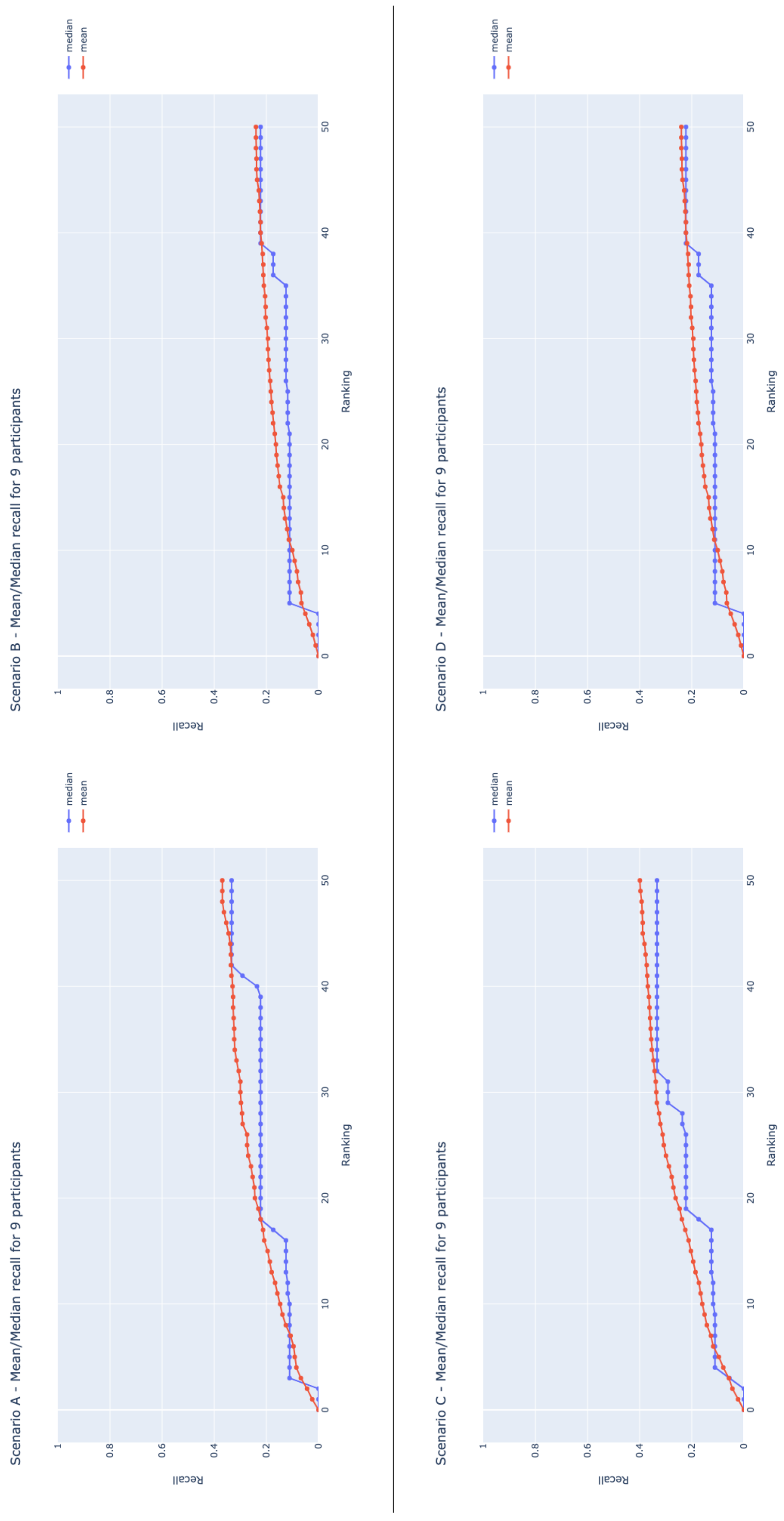


Figure C.7: Recall for 9 participants

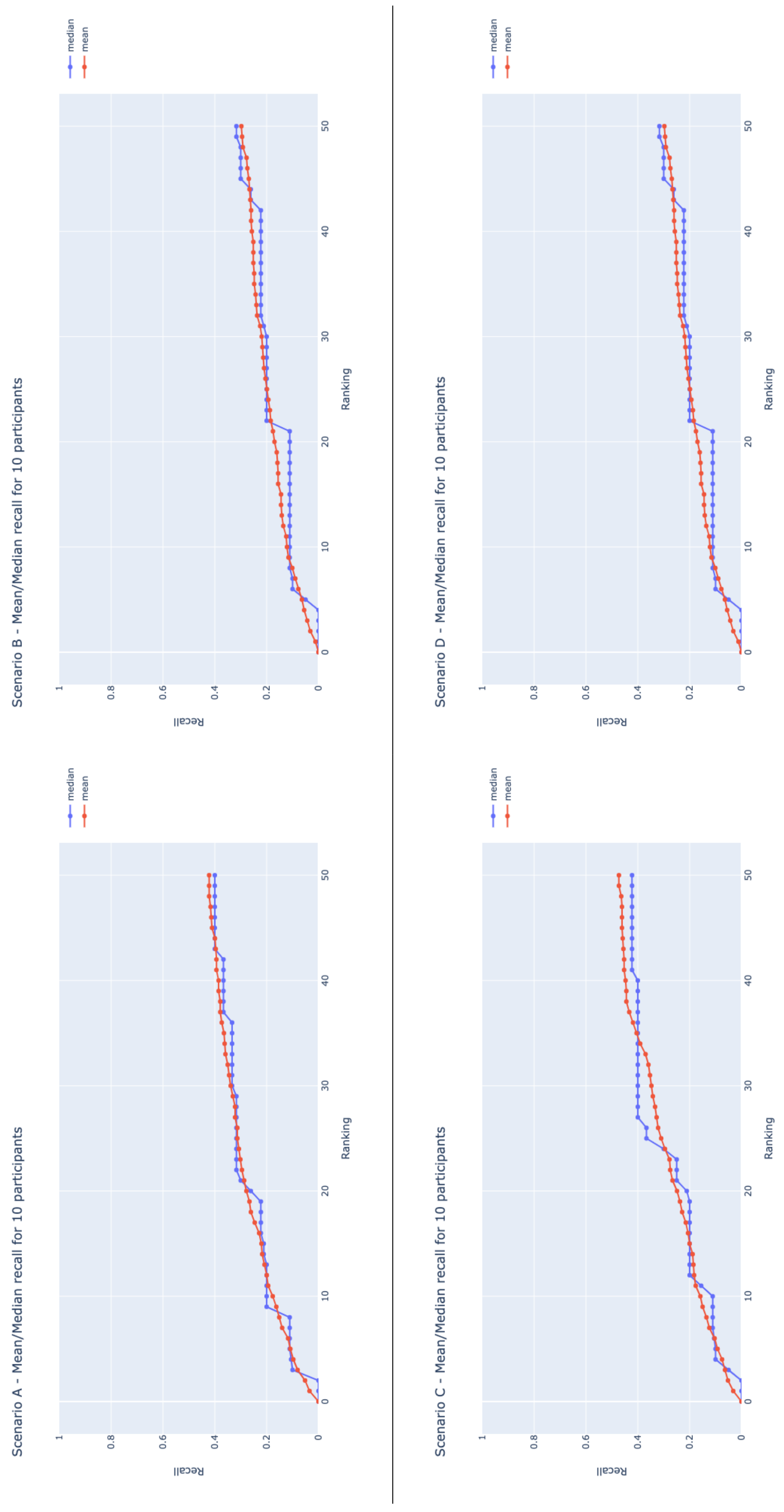


Figure C.8: Recall for 10 participants