The Robot Operating System: Package Reuse and Community Dynamics

Pablo Estefo[a], Jocelyn Simmonds[a,1], Romain Robbes[b], Johan Fabry[c]

[a]Department of Computer Science, University of Chile
[b]Faculty of Computer Science, Free University of Bozen-Bolzano
[c]Raincode Labs

Abstract

ROS, the Robot Operating System, offers a core set of software for operating robots that can be extended by creating or using existing packages, making it possible to write robotic software that can be reused on different hardware platforms. With thousands of packages available per stable distribution, encapsulating algorithms, sensor drivers, etc., it is the *de facto* middleware for robotics. Like any software ecosystem, ROS must evolve in order to keep meeting the requirements of its users. In practice, packages may end up being abandoned between releases: no one may be available to update a package, or newer packages offer similar functionality. As such, we wanted to identify and understand the evolution challenges faced by the ROS ecosystem. In this article, we report our findings after interviewing 19 ROS developers in depth, followed by a focus group (4 participants) and an online survey of 119 ROS community members. We specifically focused on the issues surrounding package reuse and how to contribute to existing packages. To conclude, we discuss the implications of our findings, and propose five recommendations for overcoming the identified issues, with the goal of improving the health of the ROS ecosystem.

Keywords: Robot Operating System, Package Management, Software Ecosystems

1. Introduction

Control programs for robots are complex pieces of software, usually both concurrent and reactive, making them hard to write and debug [1]. This type of software is also intrinsically probabilistic: sensors are not perfect, actuators suffer from wear-and-tear, etc. [2]. At the same time, robotics projects require

*Corresponding author. Address: Beauchef 851, Santiago, Chile.
Email addresses: pestefo@dcc.uchile.cl (Pablo Estefo), jsimmond@dcc.uchile.cl (Jocelyn Simmonds), rrobbes@unibz.it (Romain Robbes), johan@raincode.com (Johan Fabry)

deep expertise in diverse areas (e.g., mechatronics or computer vision), so developers are not necessarily software engineers. However, they do face software engineering (SE) challenges. For example, early work by Nesnas *et al.* [3] and Brogårdh [4] focus on the architecture of robotic applications: how to organize robotic software into reusable modules, as well as make it scalable, respectively. More recent work has focused on the challenges introduced by working in multi-robot environments [5] and cyber-physical systems [6]. Additional SE challenges arise when developing real time systems [7], as well as robots that must interact with humans [8, 9], where safety and teamwork are important issues.

Several middleware frameworks have been proposed to manage the many challenges faced in robotics, such as the Robot Operating System (ROS) [10], the Yet Another Robot Platform (YARP) [11] or the Open Robot Control Software (OROCOS) [12]. These middleware frameworks all offer an abstraction layer between the hardware and application layers, providing hardware manipulation primitives that hide the heterogeneity of the underlying hardware, as well as help manage the communication between robots. This approach has proven to be effective at accelerating robotic software development, and with more than 140 robots (humanoids, drones, cars, etc.) on the officially supported list, ROS is considered the *de facto* standard for robot programming.

As an open source project, ROS relies on the contributions from developers all over the world. ROS applications communicate using a publish-subscribe architecture, where the goal is to enable non-experts to quickly build software that includes sophisticated functionality such as path planning, object recognition and grasping. ROS also provides a package management system to simplify code reuse, so developers can contribute their own applications back to ROS in the form of packages. There are now 2,711 packages available on the ROS website[1]. The ROS community meets at the yearly ROSCon conference, with more than 450 attendees in 2017 [2]. Furthermore, the ROS Community Metrics Report lists 5,875 and 14,774 registered users respectively for *ROS Wiki* and *ROS Answers* (Q&A site) [13].

As such, ROS meets the definition of a software ecosystem – a collection of software projects which are developed and evolve together in the same environment [14] – where projects are ROS packages and the applications built on top of them. This means that we can study ROS from a software ecosystem viewpoint, such as the works of German *et al.* [15] and Decan *et al.* [16] have done for the statistical programming ecosystem centered around the R language. In particular, we want to study the package mechanism, which has become a significant hurdle in the development of ROS applications. For example, badly documented packages can be hard to reuse. Some packages become “orphans”[3], i.e., they are only available for a previous distribution of ROS and are not be-

[1]Number of packages by September 19th, 2018: 2018. `http://repositories.ros.org/status_page/melodic_default.html`. Accessed: 2018-03-13

[2]ROSCon 2017: 2017. `http://roscon.ros.org/2017`. Accessed: 2018-03-13

[3]Orphaned Package, ROS Wiki: 2017. `http://wiki.ros.org/OrphanedPackage`. Accessed: 2018-03-13

ing actively maintained. However, the problem is not always limited to the packages themselves: in the history of ROS, there have been planned breaking changes between distributions that made several packages obsolete[4]. As such, we propose the following research questions for this study:

RQ1: What difficulties do users encounter when reusing ROS packages?

RQ2: How do users contribute to the ROS ecosystem?

RQ3: What are the main contribution bottlenecks in the ROS ecosystem?

In order to shed more light on this issue and the other issues affecting ROS users, we conducted a series of 19 semi-structured interviews of ROS users, followed up by a focus group (4 participants) as well as a confirmatory on-line survey (119 participants). An analysis of the collected information uncovered the following challenges with respect to packages: 1) package reuse is not as easy as promised: bugs and lack of basic documentation hamper reuse, 2) a large amount of packages end up being abandoned by their developers, and 3) the lack of an active package maintainer affects the normal flow of contribution in the community. We also uncovered 4 challenges that affect contributors: 1) they do not have time to contribute, 2) they are not confident in the quality of their possible contributions, 3) they think that their contributions may be too specific for the general community, and 4) they are unaware of the contribution workflow. All of these challenges affect the resilience of the ROS ecosystem, both in the short- and long-term.

To address these challenges, we make five recommendations to the ROS community: 1) Identify and Predict Abandoned Packages, 2) Provide an Informative Package Repository, 3) Recommend contribution opportunities to qualified community members, 4) Limit breaking changes and 5) Motivate and encourage community contribution. The goal of these guidelines is to minimize the consequences of absent package maintainers. The guidelines were designed taking into account the contribution challenges that we identified. We also discuss ways in which these recommendations can be followed by the ROS community.

Structure of the paper. Section 2 discusses related work, and Section 3 presents background information on ROS and its community. Section 4 explains the methodology we followed to carry out our study. Section 5 discusses selected results from the interviews, focus group and online survey, focusing on package reuse and community dynamics. Section 6 discusses the contribution bottlenecks that we discovered and in Sect. 7, we present our recommendations for overcoming these bottlenecks. Section 8 presents the limitations of this study. Finally, we conclude this work in Sect. 9.

[4]ROS Hydro Migration guide, ROS Wiki: 2015. `http://wiki.ros.org/hydro/Migration`. Accessed: 2018-03-13

2. Related Work

We have divided the related work into 2 subsections. We first give an overview of empirical studies on how other communities develop software (see Sect. 2.1), and then of existing work on SE for robotics (see Sect. 2.2).

2.1. Empirical studies of SE in specific communities

Several studies have been performed to learn more about SE issues in specific contexts or for specific communities. For example, Murphy-Hill *et al.* [17] studied video game developers at Microsoft, finding that compared to a more traditional setting, creative imperatives made it more difficult to implement automated testing. Another study of video game developer by Washburn *et al.* [18], extracted lessons learned and best practices from an analysis of 155 "post-mortem" documents (i.e., what went right and what went wrong during development) that were published on `gamasutra.com`.

End users have also been studied. Given the large number of people that use spreadsheets, Hermans *et al.* [19] defined code smells for spreadsheets, implementing a tool to detect and present these code smells to users. They also studied code duplication in spreadsheets, especially focusing formulas that are copied as data instead of formulas; these copies may not be updated when one of them changes [20]. Stolee *et al.* [21] analyzed web mash-ups that were developed using Yahoo! Pipes, finding issues with reuse, evidenced by the large amount of duplication between existing programs. A similar study by Burlet *et al.* [22] focused on computer musicians using visual programming languages. They also found a very high amount of code duplication, along with differences in activity patterns, compared to traditional open-source software development.

Users of the Scratch programming language, which is popularly used to teach computational thinking, have also been studied. Aivaloglu and Hermans [23] carried out a controlled experiment to understand the impact of code smells. They found that long methods hindered the subject's understanding of a program, while code duplication made it harder for subjects to modify their programs. Code duplication in Scratch has also been studied by Robles *et al.* [24], by analyzing a large repository of Scratch programs; they found that code cloning was extensive in this repository.

Several of the references in this section focus on code duplication. A preliminary study of code duplication in the ROS ecosystem, focusing specifically on ROS launch files, indicated that this practice is also prevalent in ROS [25].

2.2. SE for Robotics

SE for Robotics studies the practice and challenges to software development, when robots are present. One journal specifically focuses on this topic: the Journal On Software Engineering for Robotics (JOSER) with seven volumes. In addition, the first edition of the RoSE[5] workshop was held at the International Conference on Software Engineering 2018.

We found three studies that are similar in focus to ours, which we discuss in this section. *HAROS* is a framework for quality assurance of ROS repositories, presented by Santos *et al.* [26]. The approach executes a static analysis rule checker on ROS repositories, in order to uncover faults and defects in ROS systems. They found that code is overly complex, that there is insufficient documentation and that the community is not concerned with standards compliance. A later study [27] shows that the community is concerned about the quality of ROS software, although they are not following their own standards.

Curran *et al.* [28] analyzed the impact and health of ROS packages in the ecosystem, instrumenting ROS core functionality in order to collect information about package execution. The impact that a package has on the ecosystem is calculated by counting direct and indirect dependencies, and package health is calculated by measuring how much time has elapsed between the latest update and activity on the issue tracker. The authors created a website to show this information to maintainers and contributors, so that maintenance efforts can be directed to packages with a high impact but low health.

At ROSCon 2017, Dittrich *et al.* [27] presented their preliminary findings about quality assurance practices in the ROS ecosystem. They interviewed ROS community members[6], finding that: 1) there is a lack of package maintainers, and consequently a problem of unmaintained packages; and 2) dependency errors are the second most frequent source of bugs in packages, and are often detected late in the development cycle. They also found that newcomers are not familiar with the quality practices defined by the ROS community.

3. The ROS ecosystem

In this section we give an overview of the ROS concepts necessary for this work, and we also describe the community behind ROS.

3.1. The Robot Operating System (ROS)

The goal driving the ROS project is to decouple robotic software from the hardware, making ROS applications robot-agnostic. Thus, from an architectural point of view, ROS is a middleware layer that resides between existing operating systems on specific robotic platforms, and user-created applications. This means that robots with equivalent hardware features can run the same applications after minor modifications. As such, ROS is available for a wide variety of robots: mobile robots, manipulators, autonomous vehicles, humanoid robots, unmanned aerial vehicles (UAV), among others.

[5]1st International Workshop on Robotics Software Engineering (RoSE'18): 2018. `https://sselab.de/lab9/public/wiki/RoSE18/index.php`. Accessed: 2018-03-13

[6]The number of interviewees was not provided.

Table 1: Release and end-of-life (EOL) dates for the last 7 ROS distributions, as well as the number of available packages (# Pack.) and authors/maintainers (# AM) for each distribution.

ROS Distribution	**Release date**	**EOL date**	**# Pack.**	**# AM**
Melodic Morenia	May 23, 2018	May, 2023	788	277
Lunar Loggerhead	May 23, 2017	May, 2019	856	294
Kinetic Kame	May 23, 2016	Apr., 2021	2509	699
Jade Turtle	May 23, 2015	May, 2017	1361	384
Indigo Igloo	Jul. 22, 2014	Apr., 2019	3210	837
Hydro Medusa	Sept. 4, 2013	May, 2015	2110	509
Groovy Galapagos	Dec. 31, 2012	Jul., 2014	2256	557
Fuerte Turtle	Apr. 23, 2012	–	2728	732

ROS applications are modeled as networks of *nodes*, which are processes that are in charge of particular tasks, such as controlling actuators, running navigation algorithms, processing images, publishing sensor data, etc. These nodes communicate through channels called *topics*. Following a publish-subscribe architecture, nodes publish *messages* through these channels, and nodes subscribe to the channels that have the data that they need. In other words, ROS applications are graphs of nodes connected by topics. This architecture is quite flexible, since new nodes can be added without updating the rest of the application.

ROS provides hardware abstraction and commonly-used functionality in robotics, which are available as *ROS Packages*. The *ROS Wiki*[7] defines a package as a piece of software that encapsulates useful functionality in a way that is easy to reuse (e.g., sensor drivers, software for planning and controlling robot motions, etc.). A package may contain ROS nodes, ROS-independent libraries, datasets, configuration files, etc..Thus, it can contain different types of files: source code (mainly C++ and Python), package definition files, build files, launch files, sets of parameter values, documentation files, and other ROS-related files, such as the physical description of a robot. In the rest of this article, when we refer to packages, we mean any package that is not part of the ROS core[8].

The ROS architecture and package system have led to ROS' success: ROS has become the de facto platform for robotic software. Table 1 shows the number of packages for the last 7 ROS distributions[9]. These numbers vary between distributions because some maintainers port their packages to a newer distribution (in the case of breaking changes), some maintainers abandon their packages, and new packages also appear. Up until Kinetic Kame, every distribution except Jade Turtle had at least 2,000 packages. We posit that, since Jade Turtle is between two long-term support (LTS) distributions, some maintainers may have decided to port Indigo Igloo packages directly to Kinetic Kame, skipping

[7]Packages - ROS Wiki: 2015. `http://wiki.ros.org/Packages`. Accessed: 2018-03-13

[8]ROS core stacks, GitHub : 2011. `https://github.com/ros`. Accessed: 2018-03-13

[9]Data obtained from the ROS website: 2018. `http://www.ros.org/browse/list.php`. Accessed: 2018-03-13

Jade Turtle altogether. Kinetic Kame is the currently recommended version, with both Lunar Loggerhead and Melodic Morenia under development, so the lower numbers for these last two distributions are expected.

3.2. The ROS Community

There is an active community supporting ROS. A group of users – ROS core and package developers, and tools maintainers – also play the role of Community Managers. This team consists of both volunteers and employees of the Open Source Robotics Foundation[10]. Some of their duties are: manage the ROS communication channels, lead the process of ROS Enhancement Proposals (REPs)[11] development, implementation and dissemination, and organize the yearly ROSCon conference[12], among others. We now describe the main communication channels used by this community. To begin with, *ROS Answers*[13] is a Q&A website that follows the same format as *Stack Overflow*. This is typically the main channel that users rely on when trying to solve problems. This channel currently has 22,247 users (22% increase since 2017), who have posed 42,360 questions, of which 69.09% have been answered[14].

Mailing lists are mostly used by package maintainers and community managers for announcements (releases, important fixes, etc). *ROS Discourse* is a forum site (launched Feb. 2016), where topics regarding the future of ROS are discussed. This website is mainly used for discussing more advanced topics, targeting experienced users, package maintainers and community managers, and is meant to replace the mailing lists. As of Oct. 2018[15], this website has over 31,000 users and approximately 14,300 discussion topics. Typical activity on this website, measured over a month (Sept. to Oct. 2018), involves approximately 880 posts and 470 active users.

Most packages are hosted on the *GitHub* platform. ROS documentation[16] recommends the use of the *Issues* section of the package repository for communication between developers and users, where questions and bugs can be reported and solved by directly interacting with the package owner. The number of packages available for each distribution, as well as the number of contributing developers, are available in Tab. 1.

Finally, many specific libraries, which in some cases are older than ROS itself, have their own sub-communities, which mostly interact through web forums. A sub-community is usually a subset of the larger ROS community that gravitates around certain specific library, framework or hardware set. Some examples of

[10]Open Robotics: 2017. `http://www.openrobotics.org/`. Accessed: 2018-03-13

[11]REP Purpose and Guidelines: 2017. `http://www.ros.org/reps/rep-0001.html`. Accessed: 2018-03-13

[12]ROSCon 2018: 2018. `http://roscon.ros.org/2018/`. Accessed: 2018-03-13

[13]ROS Answers: 2017. `http://answers.ros.org`. Accessed: 2018-03-13

[14]ROS Community Metrics Reports July 2018: 2018. `http://wiki.ros.org/Metrics`. Accessed: 2018-03-13

[15]About Discourse.ros.org - Site Statistics: 2018. `https://discourse.ros.org/about`. Accessed: 2018-03-13

[16]Tickets, ROS Wiki: 2017. `http://wiki.ros.org/Tickets`. Accessed: 2018-03-13

these are: the Point Cloud Library (PCL)[17] community or the PR2 robot[18] community. ROS users and contributors often interact with members of these sub-communities, who are commonly members of the ROS community.

4. Research Methodology

As discussed in Sect. 2.2, existing studies of the ROS ecosystem have focused on different aspects, like code quality, package dependencies and quality assurance practices. We began our study of the ROS ecosystem in 2016, interviewing 19 developers that use ROS for their robotic projects. The goal was to identify pain points in the ROS ecosystem, so as to focus our study. As such, we asked a broad range of questions about the interviewees' background and general opinion about ROS, the perceived learning curve, both positive and negative experiences using ROS, knowledge about the offered communication mechanisms, and finally, the software artifacts and developer roles involved in a typical robotics project. The interview questions are listed in Appendix A.

Most of the negative feedback given by the interviewees was about package abandonment and reuse, which is touted as one of the main advantages of the ROS ecosystem. These topics go hand in hand with community dynamics, since packages are built, maintained and used by community members. We carried out a small focus group (4 participants) in order to discuss the following topics in more depth: package abandonment and reuse, quality concerns with respect to package reuse, ways of participating in the ROS ecosystem, and contribution opportunities that are being missed.

The feedback gathered from both the interviews and focus group was used to identify the 3 research questions stated in Sect. 1: **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? We carried out an open online survey of ROS users in order to answer these questions. The survey questions are listed in Appendix B, the goal is to identify pain points in the ROS ecosystem that can be mitigated through community action.

In this section, we discuss two methodological aspects: how we picked the participants for each part of this study, and how we analyzed the collected data. The limitations of this study, as well as threats to validity, are discussed in Sect. .

4.1. Research Participants

We first interviewed developers and scientists working on robotics projects. Each interview was performed in person, with an average duration of one hour. The first author of this article, who carried out the interviews, is based in Chile and travelled to France for a research visit. For these reasons, we contacted the directors of the main robotics laboratories in Chile and France in order to

[17]Point Cloud Library (PCL): 2017. `http://pointclouds.org/`. Accessed: 2018-03-13

[18]PR2 robot: 2017. `http://willowgarage.com/pages/pr2/overview`. Accessed: 2018-03-13

recruit interviewees. We ended up with 19 participants, these both agreed to be interviewed and were available during the periods when the interviews were conducted (Chile: Mar. to Aug. 2016, France: Sept. to Nov. 2016). Seven of these belong to a robotics laboratory at the University of Chile, the rest to robotics laboratories located in France. None of those interviewed maintain a public ROS package.

In the case of the focus group, we recruited participants from a robotics laboratory at the University of Chile. We contacted people that belong to the Mechanical or Electrical engineering departments, so as to avoid people with a SE background. We chose 4 people that had some experience using ROS, and were willing and available to participate. Note that 2 of these participants were interviewed in the previous step of this study.

Finally, the survey was available to answer online through the Survey Monkey platform. Participants of the interview were invited to take the survey through an e-mail. An open invitation to participate was published in a post[19] in ROS Discourse and shared on Twitter and through ROS-related mailing lists (*e.g.* ROS Industrial). We got 119 responses, received up to August 21, 2017. The survey was voluntary and all answers were treated anonymously. The only requirement to participate was to have some experience using ROS. Most participants are affiliated to a university (58%), followed by institutions in the private sector (24%) and research centers (13%). Less than 3% claim not to be affiliated with an institution.

4.2. Data Analysis

We collected demographic information about the interviewees: the level of experience in robotics and ROS. The participants' background also varied, with participants working in vision, human-robot interaction, multi-robot environments, perception, autonomous vehicles, among others. The interviews were recorded and the audio was manually transcribed. We followed an open coding process of these transcriptions, analyzing 4,075 sentences and tagging 2,673 with a primary code and 1,195 with a secondary code, from a universe of 257 possible codes that emerged from the coding process. We wrote memos – summaries – about several topics that emerged from the data. The memo writing process started with the identification of the related codes, which allowed us to identify the parts of the interviews that were relevant to a specific memo. The first author performed the coding process and wrote the memos, with regular iterative feedback by the remaining authors.

Since we obtained the most negative feedback about *Package reusability*, *Package abandonment*, *Community bottlenecks* during the interview process, we focused on these topics during the focus group. This activity was guided by the first author of this article, lasted 3 hours and was held near the end of June

[19]Invitation post in ROS Discourse: 2017. `https://discourse.ros.org/t/participants-for-a-survey-on-collaboration-and-package-reuse-in-ros/`. Accessed: 2018-03-13

2017. The focus group was also recorded, and then manually transcribed by the first author. The comments made by each participant were grouped into one of the following subtopics: *Expectations about packages*, *Debugging*, *Package Configuration*, *Abandoned Packages*, *Community Interactions* and *Causes of missed contribution opportunities*, which are subtopics of the 3 interview topics we focused on. We wrote a memo for each subtopic, again this effort was lead by the first author, with regular iterative feedback by the remaining authors.

Finally, for the online survey, we derived 22 questions from our 3 research questions. These questions can be grouped into 5 parts. The first part asks for demographic information: type of institution (*e.g.*, University, Private Sector, etc.), country, background in robotics, level of ROS experience and background in SE. In the second part, we asked about their experience reusing ROS packages, their degree of dependence on packages, and their degree of familiarity with *Abandoned Packages* (presented in Section 6.2). The third part focused on package abandonment, asking about reasons for package reuse failure and artifacts that support package reuse. The fourth part asked about contribution activities: types of contributions made by the participant, obstacles that they encountered when contributing. We also asked participants to indicate support systems used (*e.g.*, *ROS Answers*, *ROS Wiki*, mailing lists, etc.), as well as reasons for not sharing questions about ROS with the community. The final part of the survey asks the reasons why a participant chose not to answer a question posed by a community member, e.g., on *ROS Answers* or *Stack Overflow*.

5. Results

In this section, we report the data collected in this study. Each interviewee has been assigned a numerical id. Findings are contrasted with results from the survey. We classified the interviewees according to their level of experience with ROS: *Beginner* (*Beg*), *Intermediate* (*Int*) or *Advanced* (*Adv*). *Beginner*s have up to 1 year of ROS experience and/or only use basic ROS features. An *Intermediate user* has 1 - 3 years of ROS experience, has explored more advanced ROS features (e.g., complex communication mechanisms), and can also read/write package files. Finally, an *Advanced user* has four or more years of ROS experience. These users also work with advanced ROS features and may have also written packages from scratch. *Advanced* users also have very specific questions about the packages they use, usually contacting package authors to solve issues. Survey participants were only classified according to their amount of experience with ROS, as we lack additional information. The distribution of participants is presented in Tab. 2. Note that almost a quarter of the survey participants reported no SE background, 34% had taken courses, and 42% claim to have a strong SE background.

5.1. Package Reuse Experience

When developers need a new feature, developers use a search engine to look for an existing package that implements the feature. The majority of our sur-

Table 2: 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 (100.00%)	119 (100.00%)

veyed participants have tried to reuse at least one community maintained package (95%) and depend on them (92%). Half of them have *some* or *major*[20] dependencies to these packages, so they are highly sensitive to variations in ecosystem health. The question was intentionally broad, with the goal of allowing users to specify dependencies as they perceive them.

Once a candidate package is found, the next step is to integrate it and test it. This includes downloading, compiling, installing, configuring, launching and monitoring the execution of the package (details in Sect. 5.2). Note that the integration phase is prone to failure: 71% of surveyed participants reported failures here (details in Sect. 5.3). When this happens, users look for help, trying to integrate the package using the knowledge gained from the ROS community (expanded in Sect. 5.4).

This search and integrate cycle is repeated until the expected behavior is obtained, or the developer realizes that it is not technically feasible to integrate the package. The following testimony by an advanced interviewee showcases the types of problems found reusing packages:

> *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.* — $I_{17,\mathrm{Adv}}$

5.2. Integrating a package

We now describe the package integration process, obtained by combining the interviewees' experience and the official ROS documentation. First, the user needs to *download* the package, using `apt-get` for Debian-based OS' or cloning the repository. The package then needs to be *installed*: this is done automatically when `apt-get` is used, an alternative is to use `catkin`[21], the default ROS building tool. Once installed, the package must be *configured* and *launched*: parameters, nodes, device identifiers and/or topic names must be set in order to properly launch a node/service. Finally, once the project is running, the user can monitor data flow, node status, etc., deciding whether the new package is behaving as expected, or if it needs to be reconfigured or removed.

[20]Both "a large amount" or "few critical" dependencies qualify as *major* dependencies.
[21]Catkin, ROS Wiki: 2015. `http://wiki.ros.org/catkin`. Accessed: 2018-03-13

5.3. Why do users fail when reusing packages?

We found that 74% of the survey participants that tried to reuse a 3rd party ROS package failed to do so (84 out of 113 participants, excluding the 6 participants that have never tried to reuse a package). We asked the survey participants to list reasons for which they have failed to reuse a package in the past. We populated the list with 9 reasons, extracted from the responses provided in the interview. This list seems to cover the majority of the cases that occur in practice, since only 4 survey respondents selected "Other". Five of the reasons where picked by at least a quarter of the survey participants:

1. *"The package was for an outdated ROS distribution"* (71%): This reason was frequently selected by *Beginners* (83%) and *Advanced users* (74%). In the case of *Beginners*, since they do not know the ecosystem well, they may not be aware that a package is outdated. In the case of *Advanced users*, the need for specific features may mean that there are no active packages that offer the required features. 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%): This reason was picked more by *Beginners* (61%) and *Intermediate users* (70%) than by *Advanced users* (44%). We expected this, since documentation is commonly more needed by less experienced users, while *Advanced users* can manage with other artifacts (*e.g.* launch files, robot and world geometric modelling files, etc.). Note that the third responsibility of a package maintainer, according to community practices[22], 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%): This issue affected *Advanced users* (59%) more than *Beginners* (33%) and *Intermediate users* (48%). This could be because *Advanced users* work on more complex projects and depend on more packages. In the case of specific features, there may be no obvious replacements for a buggy package. Coming across bugs in packages decreases confidence in the ecosystem [29].
4. *"I did not succeed in configuring the package for my use case (launch files, etc.)"* (34%): This reason affected 52% of *Intermediate users*, 33% of *Beginners* and 28% of *Advanced users*. Configuring launch files requires deep knowledge of the robotic domain underlying the package. Unless the user knows and/or there is a configuration guide, it is a tough task. A badly configured package may not work and may even affect some of the packages with which it interacts. We think that this cause was more frequent for *Intermediate users* because they start to use more packages, but are not as experienced as *Advanced users*.

[22]Maintenance Guide, ROS Wiki: 2017. `http://wiki.ros.org/MaintenanceGuide#Responsibilities_of_a_Maintainer`. Accessed: 2018-03-13

5. *"I could not install the package"* (38%): Core packages are normally easy to install using `apt-get`. However, other packages usually need to be manually installed, which means addressing dependency issues by hand. Although it affects less participants, we see this more frequently in users that rely on non-core packages (*Advanced users* (33%), *Intermediate user* (30%), *Beginners* (22%)). A documented installation process is fundamental when considering a new package.

The remaining 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%).

5.4. Looking for help

Our interviewees prefer to look online for help, contacting package developers or more experienced users. Another option is to consult knowledge artifacts created by the community, like wiki pages and tutorials. Other strategies are asking questions and providing/searching for answers in Q&A sites, and searching for code snippets. The 4 sources for online support prefered by the survey participants are: *ROS Answers* (85%), *ROS Wiki* (79%), *GitHub* (59%) and *Interaction with Maintainers* (29%). We now discuss each source.

5.4.1. ROS Answers

Searching. When running into an issue, the first step for all interviewees is to look for information using a search engine like Google. Relevant *ROS Answers* questions and answers can be usually found in the first 10 Google results. *ROS Answers* is also the first source of information for the majority of the survey respondents (85%), regardless of experience level. *Beginners* and *Intermediate users* look for guidance, clarifications, and/or software artifacts (e.g., code snippets, parameter values). *Advanced users* not only search for information, but also provide answers and act as moderators. Two of the 7 *Advanced users* interviewees mentioned that they do not use *ROS Answers* very often, since they face more particular or complex problems that would probably go unanswered on the platform. Here is the testimony of one of these users:

> *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.* — $I_{18,\mathrm{Adv}}$

Asking. 18 of the 19 interviewees indicated that they first try to fix package reuse issues by themselves using online resources. If the issue persists, they ask colleagues for help. The very last course of action is to post a new question on *ROS Answers.* Our survey shows that the three main reasons why users avoid asking new questions are: 1) 52% of *Beginners* think their questions are too specific, this percentage drops to 36% for more experienced[23] 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. Additional reasons for not asking questions include: presence of private/sensitive data that cannot be shared according to institutional policy, and long response times, such as in the following example:

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

Answering. Questions on *ROS Answers* are usually answered by experienced users and specialists in certain robotics domains. These users expect clearly stated questions, along with all the information and artifacts needed to give an answer. They also expect that the user that posed the question will be available to answer follow-up questions. By October 2nd 2018, 42,404 questions have been posted to *ROS Answers*, of which 69.3% have been answered. An additional 1,326 questions have been tagged with the `ROS` label on *Stack Overflow* (51.51% answered). The answered/asked ratio for *ROS Answers* shows that it as an effective channel for finding help. Note however that this ratio is slightly lower than that of popular communities on *Stack Overflow*: Python, Ruby-on-rails and the R programming language all border on an answer rate of 75%[24].

The following are the main reasons why survey participants do not answer questions on *ROS Answers*: 1) lack of time (64%); 2) answering a question properly requires further interaction with people, to clear up 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 ssue 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.

5.4.2. ROS Wiki

Searching. All interviewees agree that *ROS Wiki* is the source of documentation, and they consult it regardless of their level of experience with ROS. Surveyed respondents indicated that every package *must* document their features on the *ROS Wiki*, including current development status and any API specifications. They also expect to find installation and configuration guidelines, as well as a troubleshooting guide for common problems.

[23]Intermediate and advanced users.

[24]Python 74.58%, Ruby-on.rails 73.73%, R 74.56% answer rate in *Stack Overflow* as of October 2nd 2018

Creating and Editing pages. Each package should have a *ROS Wiki* page, which should be created and updated by the package maintainer[25]. None of our interviewees maintains a public ROS package, nor mentioned having edited existing *ROS Wiki* pages. According to the survey respondents, contributing documentation is rare: only 25% of the respondents have updated or added missing documentation, and only 13% have posted an experience report. Editing documentation is 10 percentage points below the next most popular form of contributing: submitting a new feature (36%).

5.4.3. GitHub

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

> *[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]* — $I_{3,\mathrm{Adv}}$

Users look for code examples that show how to use a package API, find recommended parameters, build a workaround to a problem. The issues that users encounter in this step are similar to those when searching in package documentation: examples could be for an outdated API and relevant examples could be missing. If a package has been abandoned, users cannot update faulty examples since there is no maintainer.

Interacting with Maintainers through the Issue Tracker. Advanced interviewees often avoid *ROS Answers* because they often face more complex or particular issues, and will often directly interact with the package maintainers through the package repository Issue Tracker.

5.4.4. Private Communication with Maintainers

Users will sometimes contact package maintainers directly. This allows them to get support without revealing sensitive information about their projects on public channels. The following quote is an example of this type of 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.* — $I_{14,\mathrm{Int}}$

5.5. What 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.* We now discuss them.

[25]ROS Package Documentation Guidelines: 2016. `http://wiki.ros.org/PackageDocumentation`. Accessed: 2018-03-13

Bug Reports. Bug reports are reported using the associated package repository Issue Tracker. This is the most common type of contribution, regardless of level of ROS experience, and is especially frequent in *Advanced* (77%) and *Intermediate users* (67%). In the short-term, bug reports benefit the package maintainer, who receives feedback about their package. In medium- and long-term, fixing a reported bug benefits all the package's users.

Q&A in ROS Answers. Activity on this Q&A platform is mostly guided by *Advanced users* (Answering: 68%, Asking: 66%) rather than less experienced users. More than a half of the *Intermediate users* (63%) ask new questions but only 26% declare having answered a question. From the *Beginners* who had contributed (50%), 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. Like bug reports, these contributions are handled through the Issue Tracker of the *GitHub* package repository. *Feature requests* are valuable feedback for the package maintainer, who can find out about unknown needs that their users have, or new ways that their packages are being used. *Feature submissions* implement new features that need to be reviewed by the package maintainer and eventually integrated into the package. This kind of contribution is mostly carried out by more experienced users. *Requesting* new features is also more in the realm of non-beginners, with 46% and 41% of the *Advanced* and *Intermediate users* participating in such a task. Half of *Advanced users* (51%) have also *submitted* new features, while only 19% of the *Intermediate users* have done so.

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

6. Bottlenecks in Contribution

Open Source communities rely on the interactions between contributors, who provide knowledge and artifacts, and users who consume them. In this section, we present 5 bottlenecks that affect the contribution dynamics of the ROS ecosystem. These emerged from the interviews and were confirmed and prioritized using the survey. We first describe them all, before focusing on the most critical to ROS' success as a robotics middleware: *Package Abandonment.*

6.1. Bottlenecks

B1. Lack Of Time. This is by far the most frequent restriction on contributing, regardless of the degree of expertise that a user has. 59% surveyed respondents indicated that it was one of the reasons why they failed to contribute. This bottleneck hinders all the types of contribution mentioned in the previous section. For example, users do not answer questions that they should be able to answer:

> *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.* — $I_{13,\mathrm{Adv}}$

This bottleneck cannot be directly addressed; rather, it is a strong constraint and any proposal should explicitly take into account users' lack of time.

B2. Package Abandonment. This occurs when it is unclear who the package maintainer is, or the maintainer is unresponsive. In this case, package development is frozen, since no one is in charge, and contributions cannot be received, evaluated, or integrated. This phenomenon is acknowledged by 90% of the survey participants, and for 63% of those who have experienced it, it occurs regularly. In the following quote, an *Advanced user* presents the case of a package for which build files needed to be updated to a newer build system (`catkin`) in order to make it compatible with newer versions of ROS.

> *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.* — $I_{4,\mathrm{Adv}}$

We discuss the issue of package abandonment in detail in Sect. 6.2, because of the substantial amount of evidence pointing to a broader and deeper impact on the ROS ecosystem. The next three bottlenecks describe situations where a potential contribution could have been shared but is not. Community Managers are responsible for making sure the community is aware about contribution channels, quality assurance guidelines and conventions, as well as trying to lower any barrier to entry that could make contributors hesitate to contribute. Like the lack of time bottleneck, these need to be taken into account when designing a solution, in order to minimize their impact.

B3. Lack of confidence in the value/quality of a contribution. Some *Beginners* and *Intermediate users* do not contribute because they are not sure about the correctness, quality or value of their contribution. 26% of the survey participants indicate that this issue is a bottleneck for them.

> *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.* — $I_{14,\mathrm{Int}}$

B4. The contribution could be too specific to my problem, domain, hardware or research problem. In general, this bottleneck was acknowledged by 31% of survey participants. *Intermediate* and *Advanced users* think that some piece of software or documentation may not be useful for others due to its specificity. The impact is that users avoid sharing bug reports, bug fixes and package customizations.

> *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.* — $I_{10,\mathrm{Int}}$

B5. Workflow is unknown or not clear. Some interviewees claim they do not share their contributions because the workflow is unknown or not clear. 14% of the survey participants indicated that they have experienced this situation.

> *Many developers don't follow REPs*[26]*. 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.* — $I_{4,\mathrm{Adv}}$

According to Steinmacher *et al.* [30], this issue has an important impact on the willingness to collaborate from community members and it represents an important barrier to entry for newcomers. The *ROS Wiki* has a section informing users about how to contribute[27]. It is not clear whether community members are aware of resources like these.

6.2. Package Abandonment

The impact of this contribution bottleneck on the ecosystem is the most complex. In part, this is because of the domain – robotics. The development of some packages requires deep knowledge of algorithms, mathematics, as well as other disciplines. Many packages are the side product of research work, *e.g.*, an experiment, a proof of concept implementation, etc. In other words, those working in this field are not full-time software developers, and so have limited resources to support their software. In the case of student projects, these are usually abandoned soon after the student has graduated. Another cause, orthogonal to the previous one, is that developers sometimes start building a package and then leave them unfinished in order to start working on another package (which may or may not replace the abandoned one).

[26]ROS Enhancement Proposal (REP) are the official documentation of architectural and design decisions for the ROS middleware: 2000. `http://www.ros.org/reps/rep-0000.html`. Accessed: 2018-03-13

[27]Get Involved, ROS Wiki: 2017. `http://wiki.ros.org/Get\%20Involved`. Accessed: 2018-03-13

This phenomenon was widely reported among survey participants: 90% of them acknowledged having encountered an abandoned package, where 63% indicate that finding an abandoned package is a common situation. *Intermediate* and *Advanced users* encounter this phenomenon with significantly higher frequency (from “very often” to “‘all the time”), presumably because they make use of more third-party packages, as well as more specialized packages. Interviewee $I_{17,\text{Adv}}$ pointed out that some popularly used packages from MoveIt framework[28] are in this situation, also highlighting the relationship between package abandonment and 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.* — $I_{17,\text{Adv}}$

6.2.1. Impact on Package Reuse and Contribution Dynamics

Intermediate and *Advanced users* rely significantly on packages outside the ROS core: more than 51% of them depend moderately to highly on packages maintained by the community. *Beginners* also depend on packages outside the ROS core, but in a slightly smaller measure (45%). As such, more than 92% of surveyed users depend on packages maintained by the community. Thus, we can argue that ROS users do rely on packages developed by the community. Contrasting these numbers with the high rate of users that have failed to reuse a package or that have acknowledged that they have had to deal with abandoned packages, we posit that package abandonment is a major issue.

For example, an *Intermediate* user that participated in the focus group, highlighted that the scope of debugging does not reach the ROS infrastructure: “*[the debugging process starts] assuming that the bug is hidden somewhere in your own code*” — $I_{22,\text{Int}}$. Thus, buggy abandoned packages means that users can no longer assume that the system outside the boundaries of their projects “just works”. The participants of our study agree that package reuse and the ROS community are the main strengths of this middleware, so the phenomenon of Package Abandonment threatens these strengths in three ways:

1. Abandoned packages weaken the ROS ecosystem reliability. Although reusing ROS packages is a common practice, several interviewees mentioned that searching for packages to reuse is quite 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*

[28]MoveIt, Motion Planning Framework: 2017. `http://moveit.ros.org/`. Accessed: 2018-03-13

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. — $I_{18,\mathrm{Adv}}$

As such, users can spend a significant amount of time trying to configure a package and understanding how to integrate it with existing code, only to find out that it has already been abandoned and that no one can answer questions about issues with the code.

2. We cannot foresee which packages are abandoned. When searching for packages to reuse, there is no way of filtering out abandoned packages, and users lose time before realized that a package is abandoned. Currently, there are no channels for users to report abandoned packages.

3. A missing maintainer cuts off contribution flows. Many maintenance tasks rely on users who provide feedback, *i.e.* bug reports, feature submission, etc. When the maintainer is missing, feedback and contributions cannot be considered. Bug reports and feature requests filed through the *GitHub* Issue Tracker are also ignored, and documentation is not added or updated. Contributions that require more effort from the contributor, such as bug fixes and feature submissions, are lost since no one is reviewing *Pull Requests*. Regarding support issues, *Advanced users* tend to interact directly with the maintainer in order to clarify more advanced questions. This is impossible in the case of abandoned packages. Finally, very specific questions on *ROS Answers* will remain unreviewed and/or unanswered. In summary, the absence of a maintainer creates a bottleneck in the evolution of a package, cutting off the contribution flow.

As ROS users declare that they lack the time to contribute, any sort of contribution is highly valuable. However, not receiving an answer is a strong barrier for contributors, which may demotivate them and make them refrain from contributing [31]. 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 [32]: it ultimately threatens the health of the ROS community and its growth.

6.2.2. How do users handle these threats today?

Users who need to reuse an abandoned package find other ways of overcoming their issues. It is in these such cases that users become skeptical of the "official" way of developing ROS applications and *scratch their itches* through custom workarounds. Our interviewees reported two common workarounds.

The first workaround is to find a solution outside of ROS. Users can wrap, or "*ROS-ify*", external libraries, applications or even an entire middleware in order to integrate them with ROS. This is done by creating a ROS node that encapsulates everything required to run the component being "*ROS-ified*". This was recognized by many interviewees as a common practice, which allows developers to quickly incorporate the desired feature or behavior in their project, albeit sacrificing quality. Interviewees $I_{17,\mathrm{Adv}}$ and $I_{18,\mathrm{Adv}}$ discussed this practice, indicating that they wrapped the OROCOS [12] middleware after several attempts to use the official `ros_control`[29] feature. While this is a testament to ROS's flexibility and extensibility, such a fragile assembly of heterogeneous applications can hardly constitute a long-term solution.

The second workaround is to fork the repository of the abandoned package, so as to adapt it without the supervision of an absent maintainer. Although this solves the problem for the package user, it is not a solution for the ecosystem. The forked package may benefit other users, but as long as its maintenance is not guaranteed it may also become an abandoned package. Someone forking a package to satisfy a local need may not necessarily be willing to maintain the package in the long-term.

These two workarounds represent shortcut solutions for a particular user of an abandoned package but do not solve the problem for the ecosystem. For both workarounds, the original abandoned package is still the point of reference. In other words, new users will still arrive at and try to use the original package, likely encountering the same issues over and over again. These users may solve the issues on their own, or fail to reuse the package. A popular abandoned package may thus be forked several times, by different users, who may duplicate their efforts by working individually, rather than in a collaborative way.

Thus, we need new collaboration strategies, ones that favor communication between package users and collaborators in order to replace the absent maintainer. All this with the goal of supporting the healthy evolution of abandoned packages and the ROS ecosystem.

7. Recommendations for overcoming contribution bottlenecks

The information presented in Sections 5 and 6 allow us to answer the research questions that guided this study. We now propose five guidelines to ease ROS package reuse that can lead to a healthier ecosystem. These guidelines take into account the bottlenecks that we described in the previous section.

Recommendation 1: Identify and Predict Abandoned Packages

Package reusability is one of the most valued strengths of ROS. However, according to our survey, 74% of users fail to reuse packages: in many cases this is because the packages have been abandoned by their maintainers. Users can waste a considerable amount of time and effort trying to install or configure such a package before realizing that it has been abandoned. Currently there are no methods to identify packages that have already been abandoned, nor to predict when packages are at risk of being abandoned. When users find packages that cannot be reused, and for which contributions will be ignored, the ecosystem becomes less reliable.

[29]ros_control - ROS Wiki: 2017. `http://wiki.ros.org/ros_control`. Accessed: 2018-03-13

There has been a reaction from the community, which has begun to manually list *Orphaned Packages*[30]. These packages are those who have not been released in the latest version of ROS. In other words, *Orphaned Packages* are a particular case of *Abandoned Packages*. Since this list is manually maintained by volunteers, it is not regularly updated nor is it complete: volunteers that lack time may not systematically keep track of the state of packages, or may be unfamiliar with the workflow to declare that a package is abandoned. According to the ROS' Maintenance Guidelines[31], the person in charge to update the state of the package is the maintainer. However, in the case of abandoned packages, the maintainer is already gone and will probably not return to update the status of the package.

The prevalence of abandoned packages in a context where humans lack time and may not know the workflow to notify the community of a package being abandoned leaves only one alternative: automation. An automated approach to identify packages that are abandoned does not suffer from these bottlenecks; approaches using techniques for mining software repositories may provide some support. Examples of them are the use of heuristics based on the history of development (in)activity of packages (*e.g.* number of commits, forks or git pull requests) [33, 34], activity of contributors in the bug tracker (or GitHub Issues) or mailing lists, the new contributors joining [35] or the release frequency.

Once abandoned packages are identified, the community can be notified so that they can take action. The first step is to warn possible users of these packages, so that they can minimize the amount of time that they dedicate to making the package work for them. Maintainers of packages that depend on abandoned packages should also be notified, so that they can start to make plans. For instance, package users may look for non-abandoned packages that offer similar functionality. They may also fork the repository of the abandoned package for minimal maintenance that lets them keep using the package. Although this action solve the problem locally if many users do it, it leads to duplicated efforts in the ecosystem. Such situation can be managed by reporting the existence of these forks even if the users are not willing to become official maintainers. In a second step, the community can make an open call for contributors to maintain the package or to address certain urgent maintenance tasks for those packages whose abandoned status seriously affects packages that depend on them.

Alternatively, and especially if there is no interest from the community to keep maintaining an abandoned package, the community can suggest options of packages that provide similar features in order to replace dependencies to abandoned packages with active ones.

A further development is to predict which packages are at risk of being abandoned. Prediction of possible package abandonment means that we try

[30]Orphaned Packages, ROS Wiki : 2017. `http://wiki.ros.org/OrphanedPackage`. Accessed: 2018-03-13

[31]Orphaned Packages, ROS Wiki : 2017. `http://wiki.ros.org/MaintenanceGuide#Role\_of\_a\_Maintainer`. Accessed: 2018-03-13

to estimate how probable it is that the development of a certain package may stop, or that contributions to this package will not be processed. Previous work [34, 36, 33, 37] agrees to consider a project as abandoned based on a threshold of 1 year of inactivity. Such a threshold makes any attempt of prediction can be achieved after one year. However, recently Coelho *et al.* proposed an approach for identifying abandoned projects in GitHub [35] that does not rely on that approach: an abandoned package may have little development activity. They classify projects through a machine learning model based on random forest classifiers. Although their sample is relatively small (127 projects) their approach obtained precision of 80% and recall of 96%. In addition, reusing their model, they provide a metric of abandonment prediction named *Level of Maintenance Activity (LMA)*. Thinking in an approach based on heuristics, there is vast amount of work related to the developer turnover in open source projects:

Constantinou *et al.* empirically studied developer retention in the RubyGem and NPM ecosystems [38]. They found that both weak commit intensity or 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 for failure [37]. 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.

Recommendation 2: Provide an Informative Package Repository

Popular and mature software ecosystems such as the LaTeX document engine, or programming languages like R and JavaScript, rely on an informative and complete catalog of available packages to reuse: CTAN[32], CRAN[33] and NPM[34], respectively. According to our survey, users look for the following information about packages that may be reused: package purpose and features, the installation process, the development status of the package (active or abandoned), who is the package maintainer, its dependencies, its API documentation, guidelines for configuring the package, information about availability of tests and/or ROS Bags. The ROS community guidelines indicate that packages should be hosted on a repository system that provides this information. Users would also like more information about available packages: troubleshooting experiences, performance benchmarks, recent activity on development and usage or alternative

[32]CTAN: Comprehensive TeX Archive Network: 2017. `https://www.ctan.org`. Accessed: 2018-03-13

[33]The Comprehensive R Archive Network: 2017. `https://cran.r-project.org`. Accessed: 2018-03-13

[34]NPM - NodeJS Package Manager: 2017. `https://www.npmjs.com`. Accessed: 2018-03-13

packages. Some of the above can be provided by harnessing the experiences of current users of a package, and complemented with popular related questions from Q&A sites. As a consequence, the user will be able to take a more informed decision when choosing which packages to depend on in their projects.

The ROS ecosystem has three main repositories with information of the available packages. The first one is the ROS Index Project[35], a community-maintained ROS package index. This effort is the closest to the informative repositories mentioned in the previous paragraph. On this website, users can find information about the indexed packages, such as: version, development status, website, documentation, name of the maintainer, link to API documentation. This information comes from the package manifest file (XML) manually entered by the maintainer. Although this index is non-official, It also uses the meta-data about the source, documentation and release git repositories from the official list of packages[36] which is manually completed and updated by package maintainers. This site, used to be an unofficial project which was officially integrated to the ROS' websites in October 2018[37].

The second source of information about packages is the *ROS Wiki*. Packages are also manually indexed on the *ROS Wiki*, by creating a dedicated page containing relevant information such as: package description, list of maintainers, ROS distribution support and API documentation. Although the guidelines recommend documenting a package before release, there are no standards with respect to minimum completeness and/or quality of this documentation. The ROS community also provides a third repository, which summarizes the data available in the main ROS repository. This website[38] lists the status of ROS packages, showing their maintainers, and current status of maintenance and availability for a given ROS distribution.

All of these repositories are manually updated, thus the accuracy of the information depends on the goodwill of the users and maintainers. These repositories are always at risk of being outdated. Even in the case of the repositories that are updated on a more regular basis, the information about abandoned packages will most likely not be up to date. Thus, some abandoned packages will still be classified as active, misleading users.

Improving the status of these repositories, while taking into account the bottlenecks (lack of time, knowledge of the contribution workflow, package abandonment) calls for the same solution as before: automation. For instance, an approach that detects abandoned packages or predicts that a package is likely to be abandoned could also update the repository to make that knowledge public to the community. The approach of Storey *et al.* about using of bot [39] for

[35]ROS Index: 2019. `https://index.ros.org`. Accessed: 2018-03-13

[36]Lists of packages for each ROS distribution: 2018. `https://github.com/ros/rosdistro`. Accessed: 2018-03-13

[37]Topic: "ROS2 Documentation", ROS Discourse: 2018. `https://discourse.ros.org/t/ros2-documentation/6475/2`. Accessed: 2018-03-13

[38]Status of ROS Packages per ROS release: 2017. `repositories.ros.org`. Accessed: 2018-03-13

assisting in testing, support and documentation can be explored. The quality and completeness of the package information was also a reported concern, by automating their measurement [40] would raise contribution opportunities for improving them. Such actions would help enrich current available information provided in the package repository and consequently easing user's package dependency decisions [41]. In addition, sporadic contributors would be aware of contribution opportunities and how to contact maintainers, which, according to Lee *et al.* are common barriers [32] for this kind of contributors. Moreover, exposing the activity of packages, announcing updates and enhancements attracts new users and contributors [42].

Recommendation 3: Recommend contribution opportunities to qualified community members

Recommender Systems in Software Engineering can help with providing information from large sources of data that, because of their size, heterogeneity and complexity of processing, cannot be done by a single person or a group of individuals [43]. Recommender Systems can be used to point out members of the community that are particularly more suitable to make certain contribution to the ecosystem [44]. They can identify experts to answer a popular unanswered question in *ROS Answers*, owners of certain expensive robots to replicate and confirm bugs associated to that hardware, or for instance, to address certain maintenance tasks from a highly used abandoned package. Selecting the adequate recommender system model requires experimentation over the particular dataset to work with [45]. Several expert recommender systems applied in Stack Overflow are based in Collaborative Filtering methods, although contrasting the size of the datasets of ROS Answers (30 thousand of questions answered) and Stack Overflow (11.7 millions of questions answered) such models are not directly reusable. We propose to use a hybrid approach, Content-Boosted Collaborative Filtering [46]. This mechanism complements the Collaborative Filtering (CF) approach with Content-Based (CB) methods. The CF approach would use the user activity behavior to model the willingness to address an unanswered question, thereafter managing the workload balance among volunteers. The CB methods would benefit from the rich vocabulary gathered from the terminology of the varied disciplines involved in robotics, and, in addition, from the names or identifiers of hardware (robots, actuators, sensors, IoT, etc.) and software artifacts (programming languages, libraries, package names, type of files, ROS concepts, etc.) that constitute a ROS project. As the ROS community is comparatively smaller than other ecosystems, it is more susceptible to issues of information scarcity (the cold start problem). In the case of *ROS Answers* for instance, we expect that the CB approach could compensate the lack of activity data of a new question, by classifying it into a cluster with questions of a similar latent semantics.

In addition to find a qualified member of the community for suggesting a contribution we also suggest to fragment a regular contribution (*e.g.* bug fix, bug report, update of documentation, new feature request, etc.) into small-sized tasks. Such "micro-tasks" would cost less time and effort reducing barriers for contributors. For example, a bug report and bug fix contributions can be fragmented into: verify of completeness and precision of the report, reproduce the bug, identify affected modules, propose a fix, reproduce the bug with the fix, verify compliance of coding standards of the fix, integrate the fix. A system, similar to an issue tracker, could keep track of the status of the contribution, inform about effort estimation, level of completeness of the contribution, and allow users to vote for the urgency of such contribution.

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

B1. Lack Of Time. Since time is a scarce resources, recommendation engines limits the time investment of users to the minimum necessary. Users do not have to waste time looking for someone with the required expertise to answer a question, if the system can recommend experts instead. Similarly, experts do not need to spend time looking for questions that they could answer; rather, the questions could be pre-selected for them. This limits their time investment to the minimum. A proactive recommender can also issue notifications to users, instead of being manually triggered by users. Similarly, a small-sized task is easier to address than a larger task and lowers the opportunity cost. This benefits particularly to peripheral contributors and One-Time Contributors [39] by lowering its barriers for contribution [32].

B2. Package Abandonment. Abandoned packages that are relevant for the ecosystem can be (partially) maintained by breaking maintenance tasks into micro-tasks: smaller tasks that are cheaper in terms of effort and/or time (*e.g.* bug confirmation, test coverage of certain component, bugfix verification, partial-feature implementation). A recommender system can offer a micro-task to qualified members of the community.

B3. Lack of confidence in the value or quality of the contribution. Experts will realize that their knowledge is valuable to the community when unanswered questions are recommended based on their experience and expertise.

B4. The contribution could be too specific to my problem, domain, hardware or research problem. Developers realize that the packages that they created to address particular research problem or equipment may be valuable to other members of the community, establishing a clear contribution opportunity.

In addition, such contribution recommendations are an opportunity to inform, remind and/or teach about Contribution Policies (how to contribute, quality guidelines, code standards, etc.) and the corresponding workflows to do so. This addresses the contribution bottleneck *B5. The workflow is unknown or not*

[39] A peripheral contributor who has had exactly one code contribution (i.e. a patch) accepted in that project" [32]

clear, further lowering barriers to contributions. In the long run, the effective implementation of this recommendation may increase the truck factor in packages and promote the emergence of new de facto contributors or maintainers for packages, necessary for a healthy growth of ROS [47].

Recommendation 4: Limit breaking changes

The ROS community has been through several cases of breaking changes. While these changes were demanded by the community, and the developers tried to make the transition as smooth as possible, these changes still had a significant impact on the community. An example of such a situation occurred between the ROS Groovy and ROS Hydro releases, where there was a large change to the build system, which switched from ROS Build to Catkin. This situation was reported by interviewee $I_{4,\mathrm{Adv}}$ (Section 6.1); this change meant that all abandoned packages were immediately outdated.

More recently, the ROS Kinetic release caused breaking changes in the API of the build system and some core functions in the middleware that had to be updated by maintainers. The result was that one third of the packages that were available in the previous release, ROS Jade, no longer worked on ROS Kinetic.

Another potentially large breaking change is occurring right now, with the introduction of ROS2, a new version of the ROS core written from scratch. ROS2 was motivated partially due to the success of ROS1 whose community demanded support of new use cases (*e.g.* teams of robots, small embedded systems, etc.) and new quality constraints (*e.g.* real-time systems, secure communication mechanisms) that were not available in ROS1. The ROS2 project website points out that “ROS 2 will be built as a parallel set of packages that can be installed alongside and inter-operate with ROS 1” [40]. Nonetheless, retro-compatibility is still a topic of discussion [41]:

> *[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.[...] Viewing ROS1 and ROS2 as an exclusive-or relationship is not a good approach.* — ROS Discourse moderator, July 3, 2018 [42]

Thus, during the transition period, it is likely that users will need to use packages from both ROS1 and ROS2, which will incur an increase of complexity as long as ROS 1 packages are not ported to ROS 2.

[40]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`. Accessed: 2018-03-13

[41]Topic: “Discussion on ROS to ROS2 transition plan”, ROS Discourse: 2018. `https://discourse.ros.org/t/discussion-on-ros-to-ros2-transition-plan`. Accessed: 2018-03-13

[42]Topic: “ROS1 or 2 for a newbie?”, ROS Discourse: 2018. `https://discourse.ros.org/t/ros1-or-2-for-a-newbie/`. Accessed: 2018-03-13

Such breaking changes may seriously affect all the packages that are already available; packages which are the reason why users prefer ROS for building their robotics applications in the first place.

These breaking changes emerge from the core ROS development team and close members, and the rest of the developers and maintainers need to be made aware of such deep changes in technology and workflows. The new technology workflows and their associated breaking changes raise the barrier to entry for maintainers that are already contributing to ROS. Packages that do not adapt to these changes are immediately outdated, and some of them may be abandoned by maintainers that have no time to learn and/or adapt their packages. This situation is particularly dangerous for packages that have already been abandoned, as it is unlikely that anyone will update them.

As a consequence, many packages are at risk of being affected by this type of change (e.g., one third during the transition between Jade and Kinetic). Maintainers will take time to react (in the best case) or may not react at all. A side-effect of this is that packages that depend on abandoned packages may be prevented from upgrading, if their dependencies cannot be upgraded. In fact, needing to use outdated packages was the most frequent reason of why our survey participants could not reuse a package (66%). Another possible symptom of this is the opposite scenario, where a user is stuck with an older version of ROS (perhaps because their dependence on an abandoned package prevents them from upgrading to a newer version), and thus cannot use a package that works only on newer versions; 16% of survey respondents failed to reuse a package for this reason.

Bogart *et al.* studied the strategies used by different communities – Eclipse, R/CRAN and Node.js/NPM ecosystems – to deal with breaking changes [48]. In Eclipse, the core community prioritizes backward compatibility. The community behind R/CRAN wants to ensure that installing and updating packages is easy for end users. For the Node.js/NPM community, maintainers should be able to install and publish packages in an easy and fast way. The Eclipse strategy seems to suit the ROS Ecosystem, considering the current bottlenecks for collaborators. In particular, some maintainers in the Eclipse ecosystem provide a frequently changing API for experts, and a simpler and more stable one for regular users. These maintainers must also document compatibility issues in detail, as well as guidelines for handling secondary effects of breaking changes, all in order to provide a smooth transition for users. In some sub-communities around certain R packages, maintainers have a more direct and continuous communication with users, specially for changes that may affect them. In three of the ecosystems studied by Bogart *et al.*, release and migration activities are supported by automated mechanisms that provide informative reports to package maintainers and users. The ROS Ecosystem would also benefit from the automation of such processes.

Recommendation 5: Motivate and encourage community contributions

There is a perception that questions posted on *ROS Answers* are either answered after a long delay, or not being answered at all. Our survey reports that 14% of respondents avoid asking in *ROS Answers* because "There is no point in doing so since people never answer". In the cases that questions are answered late, answers may then become obsolete, or are not needed anymore. In addition to discouraging participation, this situation represents a strong barrier to entry for newcomers [31]. *Recommendation 3* may help by inviting qualified members of the community to answer. However, this approach does not guarantee that the question will be answered. Here is where a complementary approach that considers motivation for contribution could take place.

The ROS community should evaluate the use of gamification techniques for motivating participation. This approach has been successful in many software development activities [49, 50]. *ROS Answers* follows the format of *Stack Overflow* in this aspect, giving points for answers, which increases users' *karma* (participation metric). Active users can then compete answering questions to get higher *karma*. Users can also gain badges for specific contribution achievements that are desired for an active and healthy community. For instance, to encourage to control the quality of questions, *ROS Answers* provides two badges: *City Duty Badge* for voting up/down 100 different questions of *City Patrol Badge* which is given for the first flagged post. The full list of badges, and how many times they have been awarded is available online[43]. These badges mostly reward Q&A viewing and voting, which is a good start. These gamification techniques (points, badges, etc.) have proven to be effective in *Stack Overflow* [51, 52, 53], therefore we can expect them to be effective in *ROS Answers* as well.

However, as seen in our study, we need to think about badges that encourage richer user participation. For example, ROS Answers users are not asking follow up questions, nor adding useful context information to existing questions, etc. We propose to further the use of these techniques extending them to other communication channels used by the ROS community. ROS or robotics-specific badges should also be considered as incentives for users and at the same time to enrich users' profiles. For example, in the case of a bug that affects a popular package, we could design team badges, where we make it explicit that as a community, we expect that users will collaborate in order to replicate this bug on different robots and ROS distributions.

There could be additional recognition in the community for outstanding contributors. For instance, public recognition in the website or at ROSCon, or coupon discounts for attending ROSCon could be attractive incentives [54, 55]. Additionally, the ROS community could concentrate contributions into events of short periods of time. Such events are common in other software develop-

[43]ROS Answers Badges: 2018. `https://answers.ros.org/badges/`. Accessed: 2018-03-13

ment communities, such as the GNOME [44] desktop environment *Hackfests* [45]. Similarly, special initiatives for contributions can be created, *e.g.* the Jenkins' "Adopt a Plugin" [46] initiative that encourages users to take charge of the maintenance of an abandoned plug-in. In all these cases, since possible contributors have to come across an announcement for contribution, we consider that these are passive approaches. On the other hand, the approaches presented in *Recommendation 1* and *3* are active approaches.

8. Limitations of this study

The first part of this study is a qualitative study, based on a restricted number of interviews, with participants from only two countries (France and Chile). This is due to the difficulty of getting access to interview participants. As such, the finding from this part of the study cannot be generalized. To remedy this, we then used a focus group to assess for which findings there was some level of agreement. Since the focus group was of a limited duration and involved a small number of people, we turned to another strategy to further triangulate our findings. Thus, we conducted an online survey over a broader segment of the ROS community, where we verified agreement with key statements from the interviews and focus group. The survey included respondents from a variety of countries and and a greater variety of profiles. Both the interview and the survey were iteratively prototyped, in order to make sure that the questions would be clearly understood by participants.

Our interview participants were all scientists. Other ROS users (e.g., practitioners in industry, students/teachers, hobbyists) may have different motivations, needs and interests. Therefore, it is possible that non-scientist may be affected by different issues, and thus that answers from non-scientist would open up different findings. We do stress, however, that a significant proportion of survey respondent (1/4) were from the private sector. In order to gauge this possible threat, we compared the distribution of answers between scientists and non-scientists. We find them to be very similar, which makes us think that large differences between the populations are unlikely. Thus, while non-scientists may face additional issues, they also struggle with reported issues such as Package Abandonment and Bottlenecks in Contribution, as was confirmed by the answers of non-scientists in the online survey.

Our discussion of the "lack of time" bottleneck is quite general. Some participants of the ROS ecosystem (e.g., members of the OSRF) are paid to contribute to ROS, may thus be less affected by the "lack of time" bottleneck. Our interviewees have the profile of unpaid volunteers, mostly working at universities and research centers. None of them mentioned being paid to specifically contribute

[44]The GNOME Project: 2018. `https://www.gnome.org`. Accessed: 2018-03-13

[45]Hackfests in the GNOME-related world: 2018. `https://wiki.gnome.org/Hackfests`. Accessed: 2018-03-13

[46]Adopt a Plugin - Jenkins - Jenkins wiki: 2017. `https://wiki.jenkins.io/display/JENKINS/Adopt+a+Plugin`. Accessed: 2018-03-13

to ROS. Regarding survey participants, we can not know whether they are paid to contribute to ROS specifically. Extrapolating from our interview responses, we would hypothesize that this is unlikely for respondents employed at universities and research centers. However, we can not hypothesize for respondents in the private sector. Therefore, although a very interesting direction, our data does not let us study the consequences of being a paid or an unpaid contributor in the contribution bottlenecks.

Internal bias is present in the analysis of the transcription, as the coding process was carried out by the first author. To increase reliability of the coding, the first author was given feedback on the coding process in meetings with two of the authors of this article. This was done by going through random sentences in the corpus and checking the assigned tags. Summarizing these transcripts into memos was done collaboratively by three of the authors of this article.

Additional interviews, focus groups, large-scale surveys and/or additional quantitative studies should be carried out to increase the confidence in the findings of this study. As some of the aspects are not specific to robotics per se, replication in domains other than robotics is also an interesting avenue of research.

9. Conclusions

The ROS ecosystem consists of a large number of packages that are available for developing robotics applications. The architecture of ROS applications as a publish-subscribe system was designed with reuse in mind. However, beyond its technical architecture, significant challenges affect ROS users when reusing packages. We started with an exploratory series of 19 interviews and a focus group with 4 robot scientists. Based on the feedback of these activities we defined a set of 3 research questions about package reuse and abandonment, as well as community dynamics. We then carried out an online survey in order to capture the experience of a wider variety of ROS users, where 119 people completed our survey.

We found that ROS users depend on packages developed by the community: more than 90% of the survey respondents do so. At first glance, this validates the claim that ROS encourages reuse. However, failing to reuse a package is also very common in this ecosystem, with more than 70% of the survey participants reporting that this has happened to them. One specific case is when a package has been abandoned, limiting the evolution of the ecosystem.

ROS is also an ecosystem with a large amount of open-source software. As such, the community relies on open-source contributions to thrive. We identified five bottlenecks to contributions to the ROS ecosystem:

1. Users simply lack the time to make contributions.
2. Abandoned packages (which can be caused by lack of time) cannot receive contributions.
3. Some users are not confident that their contributions are of high enough quality.

4. Some users think that their contributions are too specific to be more generally valuable.
5. The workflow for contributing to the ROS ecosystem is not always known.

Overall, package abandonment and contribution bottlenecks are complex phenomena that influence each other. Based on this study, we also formulated 5 guidelines that could improve the health of the ROS ecosystem. Some of the guidelines can also be supported by automation, which is vital in a context where lack of time is prevalent. The five guidelines are:

1. Identify and predict abandoned packages, in order to notify prospective and current users of these packages.
2. Maintain an informative package repository that includes the status of the packages.
3. Leverage recommender systems to proactively propose contribution opportunities of small-sized tasks to members of the community.
4. Limit breaking changes in the ecosystem, as abandoned packages will be left behind.
5. Encourage community contributions on social websites such as Q&A websites.

Several of these issues and bottlenecks are not unique to the ROS ecosystem. As such, we expect them to be applicable to other ecosystems as well.

10. Future Work

Our outlook on this work goes towards the evaluation of two of proposed recommendations: *R1: Identify and Predict Abandoned Packages* and *R3: Recommend contribution opportunities to qualified community members.*

The community needs to know which packages are abandoned or at serious risk of being abandoned. As maintenance efforts are always scarce, the focus should be set on packages which are critical in the dependency chains of the ecosystem: many packages may depend on them to work, or they are popular among users. We are currently working on methods for identifying these types of packages in the ROS ecosystem. The idea is to predict the likelihood of an active package being abandoned. An interesting research venue is to extend this evaluation in other software ecosystems. Such results will provide insights on technical and social aspects of different ecosystems regarding the package abandonment phenomenon.

One interesting approach to address the package abandonment issue is to distribute the maintenance workload among qualified members of the community. The maintenance workload could be chopped up into smaller tasks, *e.g.* reviewing a feature request or replicating a bug report. Existing recommender systems could be adapted to make recommendations about who could carry out these tasks. Finally, another interesting avenue of research is how to motivate community members into making contributions of different natures: the amount of time or effort required, degree of expertise required, etc.

References

[1] F. Martín, E. Soriano, J. M. Cañas, Quantitative analysis of security in distributed robotic frameworks, Robotics and Autonomous Systems 100 (2018) 95–107. doi:10.1016/J.ROBOT.2017.11.002.

[2] S. Thrun, W. Burgard, D. Fox, Probabilistic robotics, Vol. 45, 2005. doi:10.1145/504729.504754.

[3] I. A. D. Nesnas, A. Wright, M. Bajracharya, R. Simmons, T. A. Estlin, Claraty and challenges of developing interoperable robotic software, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003, pp. 2428–2435. doi:10.1109/IROS.2003.1249234.

[4] T. Brogårdh, Present and future robot control development—an industrial perspective, Annual Reviews in Control 31 (1) (2007) 69 – 79. doi:https://doi.org/10.1016/j.arcontrol.2007.01.002.

[5] S. Kernbach, D. Häbe, O. Kernbach, R. Thenius, G. Radspieler, T. Kimura, T. Schmickl, Adaptive collective decision-making in limited robot swarms without communication, The International Journal of Robotics Research 32 (1) (2013) 35–55. doi:10.1177/0278364912468636.

[6] P. J. Mosterman, J. Zander, Cyber-physical systems challenges: a needs analysis for collaborating embedded software systems, Software & Systems Modeling 15 (1) (2016) 5–16. doi:10.1007/s10270-015-0469-x.

[7] A. Liekna, E. Lavendelis, A. Nikitenko, Challenges in development of real time multi-robot system using behaviour based agents, in: Distributed Computing and Artificial Intelligence, 2013, pp. 587–595.

[8] M. Gombolay, X. J. Yang, B. Hayes, N. Seo, Z. Liu, S. Wadhwania, T. Yu, N. Shah, T. Golen, J. Shah, Robotic assistance in coordination of patient care, in: Robotics: Science and Systems XII, Robotics: Science and Systems Foundation, 2016. doi:10.15607/rss.2016.xii.026.

[9] B. Hayes, B. Scassellati, Effective robot teammate behaviors for supporting sequential manipulation tasks, in: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2015, pp. 6374–6380. doi:10.1109/IROS.2015.7354288.

[10] 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`

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

[12] 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.

[13] T. Foote, ROS Commuity Metrics Report, verified 21/08/2017 (July 2016).
URL `http://download.ros.org/downloads/metrics/metrics-report-2016-07.pdf`

[14] M. Lungu, M. Lanza, T. Gîrba, R. Robbes, The small project observatory: Visualizing software ecosystems, Science of Computer Programming 75 (4) (2010) 264–275.

[15] D. M. German, B. Adams, A. E. Hassan, The evolution of the R software ecosystem, Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR (2013) 243–252doi:10.1109/CSMR.2013.33.

[16] A. Decan, T. Mens, M. Claes, P. Grosjean, When github meets CRAN: an analysis of inter-repository package dependency problems, in: IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER - Volume 1, 2016, pp. 493–504. doi:10.1109/SANER.2016.12.

[17] E. Murphy-Hill, T. Zimmermann, N. Nagappan, Cowboys, ankle sprains, and keepers of quality: How is video game development different from software development?, in: Proceedings of the 36th International Conference on Software Engineering, ACM, 2014, pp. 1–11.

[18] M. Washburn Jr, P. Sathiyanarayanan, M. Nagappan, T. Zimmermann, C. Bird, What went right and what went wrong: an analysis of 155 postmortems from game development, in: Proceedings of the 38th International Conference on Software Engineering Companion, ACM, 2016, pp. 280–289.

[19] F. Hermans, M. Pinzger, A. van Deursen, Detecting and visualizing interworksheet smells in spreadsheets, in: Software Engineering (ICSE), 2012 34th International Conference on, IEEE, 2012, pp. 441–451.

[20] F. Hermans, B. Sedee, M. Pinzger, A. v. Deursen, Data clone detection and visualization in spreadsheets, in: Proceedings of the 2013 International Conference on Software Engineering, IEEE Press, 2013, pp. 292–301.

[21] K. T. Stolee, S. Elbaum, A. Sarma, Discovering how end-user programmers and their communities use public repositories: A study on yahoo! pipes, Information and Software Technology 55 (7) (2013) 1289–1303.

[22] G. Burlet, A. Hindle, An empirical study of end-user programmers in the computer music community, in: Proceedings of the 12th Working Conference on Mining Software Repositories, IEEE Press, 2015, pp. 292–302.

[23] F. Hermans, E. Aivaloglou, Do code smells hamper novice programming? a controlled experiment on scratch programs, in: 24th International Conference on Program Comprehension (ICPC), IEEE, 2016, pp. 1–10.

[24] G. Robles, J. Moreno-León, E. Aivaloglou, F. Hermans, Software clones in scratch projects: On the presence of copy-and-paste in computational thinking learning, in: Software Clones (IWSC), 2017 IEEE 11th International Workshop on, IEEE, 2017, pp. 1–7.

[25] 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.

[26] A. Santos, A. Cunha, N. Macedo, C. Lourenço, A framework for quality assessment of ROS repositories, in: IEEE International Conference on Intelligent Robots and Systems, Vol. 2016-Novem, 2016, pp. 4491–4496. doi:10.1109/IROS.2016.7759661.

[27] Y. Dittrich, G. van der Hoorn, A. Wasowski, How ROS cares for Quality, ROSCon 2017 (2017).

[28] W. Curran, T. Thornton, B. Arvey, W. D. Smart, Evaluating impact in the ROS ecosystem, in: Proceedings - IEEE International Conference on Robotics and Automation, no. June in 2015, 2015, pp. 6213–6219. doi:10.1109/ICRA.2015.7140071.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

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

[34] 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.

[35] J. Coelho, M. T. Valente, L. L. Silva, E. Shihab, Identifying Unmaintained Projects in GitHub, in: Proceedings of ESEM 2018.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] G. A. A. Prana, C. Treude, F. Thung, T. Atapattu, D. Lo, Categorizing the Content of GitHub README FilesarXiv:1802.06997.

[41] 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.

[42] 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.

[43] 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.

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

[45] X. Wang, C. Huang, L. Yao, A survey on expert recommendation in community question answering, JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 33 (1) (2018) 1–29. arXiv:arXiv:1807.05540v1, doi:10.1007/s11390-015-0000-0.

[46] P. Melville, R. J. Mooney, R. Nagarajan, Content-boosted collaborative filtering for improved recommendations, Proceedings of the 18th National Conference on Artificial Intelligence (AAAI) (July) (2002) 187–192. arXiv:86, doi:10.1.1.16.4936.

[47] 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.

[48] 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.

[49] T. Dal Sasso, A. Mocci, M. Lanza, E. Mastrodicasa, How to gamify software engineering, in: Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on, IEEE, 2017, pp. 261–271.

[50] 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.

[51] 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.

[52] 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.

[53] 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.

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

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

Appendix A. Interview Questions

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?

Appendix 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.