It Takes a Village to Build a Robot: An Empirical Study of The ROS Ecosystem

Sophia Kolak[*], Afsoon Afzal[†], Claire Le Goues[†], Michael Hilton[†], and Christopher Steven Timperley[†]
[*]Columbia University, New York, NY
[†]Carnegie Mellon University, Pittsburgh, PA
Email: sdk2147@columbia.edu, afsoona@cs.cmu.edu, clegoues@cs.cmu.edu, mhilton@cmu.edu, ctimperley@cmu.edu

***Abstract*—Over the past eleven years, the Robot Operating System (ROS), has grown from a small research project into the most popular framework for robotics development. Composed of packages released on the *Rosdistro* package manager, ROS aims to simplify development by providing reusable libraries, tools and conventions for building a robot. Still, developing a complete robot is a difficult task that involves bridging many technical disciplines. Experts who create computer vision packages, for instance, may need to rely on software designed by mechanical engineers to implement motor control. As building a robot requires domain expertise in software, mechanical, and electrical engineering, as well as artificial intelligence and robotics, ROS faces knowledge based barriers to collaboration.**

In this paper, we examine how the necessity of domain specific knowledge impacts the open source collaboration model. We create a comprehensive corpus of package metadata and dependencies over three years in the ROS ecosystem, analyze how collaboration is structured, and study the dependency network evolution. We find that the most widely used ROS packages belong to a small cluster of foundational working groups (FWGs), each organized around a different domain in robotics. We show that the FWGs are growing at a slower rate than the rest of the ecosystem, in terms of their membership and number of packages, yet the number of dependencies on FWGs is increasing at a faster rate. In addition, we mined all ROS packages on GitHub, and showed that 82% rely exclusively on functionality provided by FWGs. Finally, we investigate these highly influential groups and describe the unique model of collaboration they support in ROS.

***Index Terms*—robot operating system, software ecosystem, software evolution, robotics software, collaboration**

I. Introduction

The Robot Operating System (ROS) is a framework for building robotics software, designed with the promise of making development easier through modular design and code reuse.[1]. It is widely used by robotics developers, and contains 5,798 packages across its 12 distributions.[2]

The stated goal of the ROS project is "to encourage *collaborative* robotics software development" [1]. Before ROS, it was common for developers to write highly specific robotics software for one project, and then start from scratch as soon as they moved to a new job [2]. ROS's innovation was its promise of modularity—it sought to end wasteful practice by letting developers borrow and share the parts of their robots that could be reused.

[1]http://wiki.ros.org/ROS/Introduction
[2]https://index.ros.org/stats/

Developing a fully functional robotic system is a difficult task that "no single individual, laboratory, or institution can hope to do on their own" [1]. It requires expertise in many different fields such as mechanical engineering, computer science, electrical engineering, or even math and physics. For example, experts who create computer vision packages may need to rely on software designed by mechanical engineers to implement motor control. As a result, the ROS framework allows developers without domain expertise in many areas of robotics to build on top of each others' knowledge, and put together a fully functional robot.

The ROS framework has indeed been successful in gathering people with different backgrounds to compose the ecosystem [3]. However, while this diversity is a strength for ROS, it also presents a challenge as there are multiple sub-communities focused on these different expertise areas.

Very few studies have looked at the ROS ecosystem. Pichler et al. [4] investigate the quality of ROS packages, and the impact of quality on popularity of the package in the ecosystem. They show that the quality of a project does not correlate with its usage. With a qualitative study, Alami et al. [5] found that the implementation and execution of QA practices in the ROS community are influenced by social and cultural factors. However, none of these studies investigate the structure of the ROS ecosystem and the format of collaboration deployed in this ecosystem.

In addition to being understudied, the ROS ecosystem is currently at the pivotal point of transitioning from its first version (ROS) to the second major version (ROS2). By studying ROS now, this work can help identify the underlying structures necessary for the growth and maintenance of ROS, as well as point out bottlenecks to avoid during the design and implementation of the ecosystem for ROS2.

In this paper, we quantitatively examine the growth of the ROS ecosystem and package dependency structure over three years and on varying scales by collecting data on all ROS packages released in the Kinetic distribution, and creating a dependency graph of the ecosystem by parsing package manifests. Using this data, we study the nature of collaboration in the ROS ecosystem, and answer the following research questions: **RQ1)** *How is the ecosystem growing?* We show that the ROS ecosystem has been successfully attracting new applications and developers at an exponential rate, and the number of released packages has grown at a steadier, linear rate over the past three years; **RQ2)** *How is the ROS ecosystem structured?* We find an extraordinary level of structure and cooperation within the ROS ecosystem, facilitated by its expertise-focused foundational working groups; and **RQ3)** *How do the foundational working groups contribute to the ecosystem?* We show that these working groups are the most influential groups in the ecosystem, as they account for more than 80% of all dependencies, and 82% of ROS applications *exclusively* rely on the packages they release. However, their membership has grown little in recent years, which may have implications for the long-term health of the ecosystem.

In this paper, we make the following contributions:

- We create a database of packages within the ROS ecosystem (Kinetic distribution) and their dependencies, with the ability to create snapshots of the ecosystem over time.
- We examine all (released and unreleased) ROS packages on GitHub along with their dependencies, and discover over 200,000 Client applications that extend the ROS framework.
- We analyze the level of growth in the ROS ecosystem, and compare it to PyPI and Julia; two other software ecosystems similar in age and/or size.
- We manually classify organizations contributing to the ROS ecosystem, identify the structure of collaboration in the ROS ecosystem, and show that foundational working groups are the most influential organizations that account for the vast majority of dependencies in the ecosystem.

The full data from this paper is publicly available at https://doi.org/10.5281/zenodo.3997720.

II. Background

In this section, we introduce the reader to the ROS ecosystem and package structure. The Robot Operating System (ROS) [6] is a flexible framework for writing robotics software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotics platforms. ROS is generally designed following publisher-subscriber architecture, which allows it to be modular at a fine-grained scale.[3] For example, a ROS process (i.e., node) controls a LADAR sensor, and another node controls the wheel motors. These nodes, which can be from different packages, can communicate with each other through message passing.

ROS is a relatively young framework (first released in 2009), and is currently used by thousands of people around the world.[4] It follows an annual release model that is both similar to and linked to Ubuntu, and as of May 2020, there have been 12 official, released ROS distributions (e.g., Lunar, Kinetic, and Indigo). ROS is designed with the purpose of encouraging *collaborative* robotics software development by allowing robotics developers to build upon each other's

[3]ROS also provides other methods of communication among processes such as peer-to-peer request-reply service calls.

[4]ROS has more than 34,000 registered users on *ROSanswers*, the main Q&A platform for ROS users: http://download.ros.org/downloads/metrics/metrics-report-2019-07.pdf

```
<package format="2">
  <name>robot_arm</name>
  <version>1.2.3</version>
  <description>controls robot arm</description>
  <maintainer email="janedoe@osrf.org">Jane
      Doe</maintainer>
  <license>BSD</license>
  <buildtool_depend>catkin</buildtool_depend>
  <exec_depend version_gte="0.2.19">
       python_qt_binding
  </exec_depend>
</package>
```

Fig. 1. An example `package.xml` file for a sample package called `robot_arm`.

work [1]. Here, we provide a high-level overview of the ROS package structure and package distribution.

A. Package structure

All ROS packages follow a specific structure, and contain an XML file with meta information about the package. This manifest file, called `package.xml`, records basic information about the package and its dependencies. Figure 1 illustrates a sample `package.xml` file, the package name, version, description, maintainer, and license as well as its dependencies. Multiple packages in ROS can also be organized into a single metapackage, historically known as a *stack*.[5]

The declared dependencies in a `package.xml` file must specify the name of the package, and the type of dependency, and may optionally enforce a minimum, maximum, or equivalent version of the package to depend on. For instance, the `robot_arm` package in Figure 1 has a buildtool dependency on the ROS package `catkin`, and an execution dependency on the ROS package `python_qt_binding`. This declaration does not specify a version of `catkin`; it does specify a minimum version for its dependency on the `python_qt_binding` package. If no version is specified, ROS's main package installer, *rosinstall*, downloads and installs the latest released version of a dependency. In practice, most dependencies are simply specified by the name of the package, meaning that the actual source code a package relies on is ambiguous.

B. Package distributions

The ROS distributions and the stacks they contain are managed by *rosdistro*, a tool that allows users to access the full dependency tree and information of all packages and repositories in the ROS ecosystem. Every ROS distribution includes a `distribution.yaml` file, which follows the format specified by rosdistro, and records information on all packages in that distribution. Figure 2 shows a sample `distribution.yaml` file for the Kinetic distribution with a single package `opencv3`.

[5]http://wiki.ros.org/action/show/rosbuild/Stacks

```
release_platforms:
  ubuntu:
  - xenial
repositories:
  opencv3:
    release:
      tags:
        release:
            release/Kinetic/{package}/{version}
      url: https://github.com/ros-gbp/opencv3-
          release.git
      version: 3.3.1-5
    status: maintained
type: distribution
version: 2
```

Fig. 2. Excerpt from a `distribution.yaml` file, showing how a sample package (`opencv3`) is documented by the rosdistro.

Within a `distribution.yaml` file, each included package or stack has its own metadata entry, including as much or as little information as the maintainer decides to include. For instance, some package entries contain source, release, and documentation URLs, with corresponding version and status information for all three instances of the code. Other packages in the YAML file contain only a documentation URL.

`distribution.yaml` files also do not maintain any information about older versions of packages. When a new version of a package in the distribution gets released, its entry in the file is overwritten. This means that formerly released versions of ROS packages are not explicitly recorded.

The `distribution.yaml` files are maintained on the ROS GitHub repository.[6] Between the first commit to `kinetic/distribution.yaml` on March 4, 2016 and our data collection on June 24, 2019, 8,957 commits altered this file. As of June 24, 2019, `distribution.yaml` contained information on 2,692 packages.

III. Methodology

The goal of this study is to investigate the ROS ecosystem to gain a better understanding of how collaboration is conducted at the package level in ROS. We address the following high-level research questions:

- **RQ1:** *How is the ecosystem growing?*
- **RQ2:** *How is the ROS ecosystem structured?*
- **RQ3:** *How do the foundational working groups contribute to the ecosystem?*

Following the methodology of [7] [8], and [9], we mined ROS repositories and analyzed the data to answer our research questions. We build two related corpora: (1) ROS Distribution data (ROS Distro), which contains information on framework packages released in the rosdistro for Kinetic, the largest and most widely-used distribution to date[7] (Section III-A), and (2) ROS Client Applications (ROS Client), which draws data from all ROS packages found on GitHub, generally consisting of ROS Client applications built against the framework. This dataset allows us to draw inferences about how distribution code is used in real robotics projects.

In addition, we used the data provided by the *Libraries.io* [10] platform to compare the growth of the ROS ecosystem against other software ecosystems, described in Section III-C.

A. ROS Distribution

We constructed a catalog of all versions of all packages that have ever been released in the rosdistro for the Kinetic distribution, as well as the date and time of each release. At the time of our data collection on June 24th 2019, 11,866 versions of 2,647 unique packages had been released in the rosdistro over the entire history of the Kinetic distribution. On average, each package has 4.48 released versions.

We processed all ∼60,000 commits to the rosdistro as of June 24th 2019, seeking commits that changed the *kinetic/distribution.yaml* file in particular. We then extracted the commit hash, date-time, and package metadata for each commit which indicated the addition of a new package or the update of a package version. Since new versions of packages overwrite the older `distribution.yaml` file, this data was the first place to explicitly capture which versions of various packages exist and precisely when they were added.

We use package dependency as a proxy for code reuse (and, loosely, community-level collaboration). Since we are interested in analyzing collaboration in the ROS community, we collect dependency relationships between packages. As mentioned in Section II-A, ROS packages specify dependencies on other packages in their `package.xml` file. We mined this manifest file for every release of each package in our dataset, collected their dependencies, and created a directed graph, where `package-versions` are nodes, and dependencies are edges. For example, `foo-1.0.1` specifies a dependency to `bar-2.0.3`, and as a result a directed edge connects these two `package-version` nodes. We adopt the terminology of dependency and reverse dependency used in the literature to differentiate between incoming and outgoing edges of the dependency graph [11].

Even though it is possible to specify a version for dependencies, most packages do not; only 1.8% of dependencies in the distribution specified a particular version of the package they relied on. As a result, we infer dependency version based on package release time. That is, we assumed that if package foo, version 1.0 was created on date D, and specified a dependency on package bar, foo likely relied on the most recently released version of bar before time D. This reconstructs the version of a dependency that `rosinstall` had access to at the time of the package's release.

Our collected data covers the entirety of the ROS Kinetic distribution. For the purposes of our analysis (Section IV), we create and analyze data snapshots every six months between June 2016 and June 2019. Snapshots for different dates are

[6]https://github.com/ros/rosdistro

[7]https://discourse.ros.org/t/ros1-2018-version-survey-the-results-are-in/4547

reconstructed by querying the data set and can be created for any point in the distribution's history. We release both the data and the source code to create these snapshots at https://doi.org/10.5281/zenodo.3997720.

B. ROS Client Applications

As described in Section II-B, packages in the rosdistro are released to be used by other packages built against the ROS framework. Therefore, there are many ROS projects and packages that are developed for personal or professional use, and are never released as part of the rosdistro.

To analyze the ROS ecosystem with a broader perspective, and calculate popularity and dependability of ROS packages, we mined GitHub and identified all repositories containing a `package.xml` file that specifies a dependency on `catkin`, the universal build system for ROS.

The search for ROS packages on GitHub resulted in 210,656 repositories. Each repository may contain multiple ROS packages. Overall, we found 230,488 ROS packages on GitHub. Since the packages in the ROS Distribution dataset are also hosted on GitHub, they are included in the ROS Client dataset. However, they only constitute 1% of the dataset and therefore can be overlooked. We also mined and analyzed the `package.xml` file of every package, and collected their dependencies in the same manner described in Section III-A. However, we collected this information only on the latest version of the packages since they do not have official releases. At the end, we recorded 2,228,860 dependencies for all ROS packages on GitHub.

C. Libraries.io

Libraries.io [10] is a platform that indexes data on dozens of popular software package managers,[8] monitors package releases, and maps the dependency relationships between packages. This platform is commonly used by researchers to study software ecosystems [11], [12]. Using the API provided by the Libraries.io platform (version 1.6.0), we study the growth of two software ecosystems, PyPI and Julia, as baselines against the ROS ecosystem. We select the PyPI ecosystem since it has been established as mature and stable [9], and Julia since it is similar in size to the ROS ecosystem.

IV. Results and Analysis

In this section, we present the results of our study in response to our research questions.

A. RQ1: How is the ecosystem growing?

To answer this research question, we measure growth of the ROS ecosystem in terms of its size (i.e., number of client and distro packages) and developer membership (i.e., different Github users that have written ROS code) over time, and compare to other software ecosystems. Given the advanced domain expertise required to contribute high-quality, reusable robot software (e.g., a local planning algorithm), ROS may have more difficulty in growing its size and membership compared to other ecosystems of similar size. Understanding

[8]At the time of writing, Libraries.io provides data on 4,987,611 packages from 37 package managers, which does not include ROS.

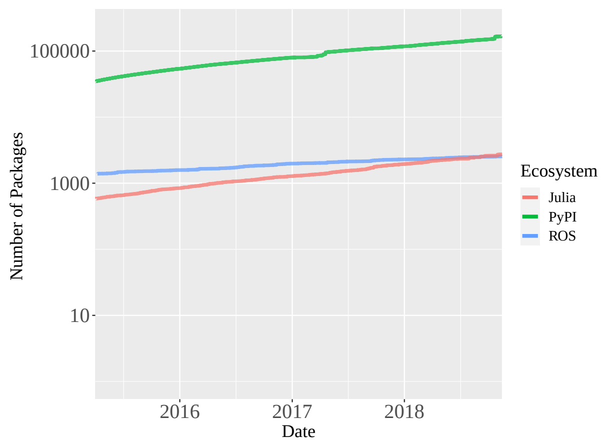

Fig. 3. Growth of the ROS (Kinetic), PyPI, and Julia ecosystems, in terms of number of packages, plotted on a log scale. For each of these curves, the most explanatory regression was linear, as opposed to exponential.

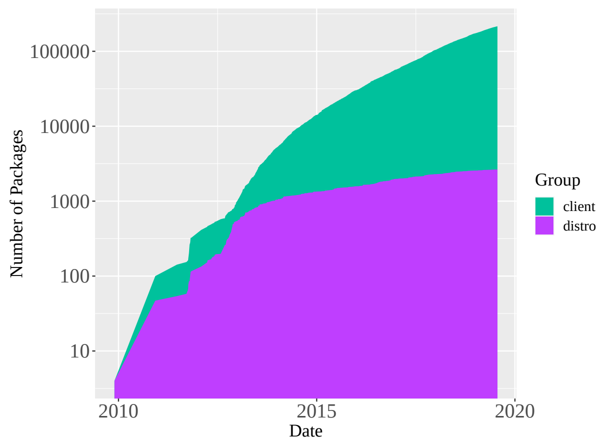

Fig. 4. The total number of ROS client and distro packages over time, plotted on a log scale.

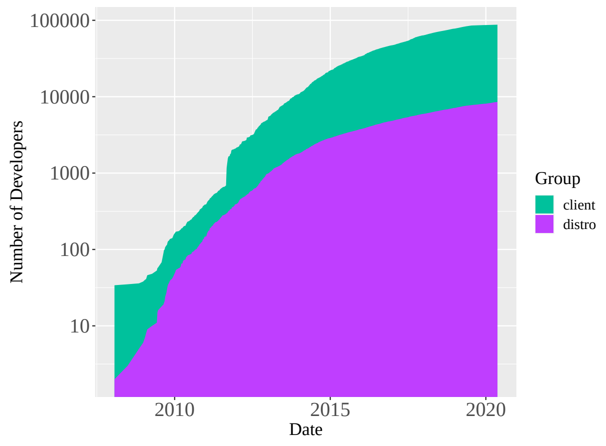

Fig. 5. Total number of GitHub users who have made at least one commit to ROS Distro and ROS Client packages over time, plotted on a log scale.

this growth is vital for assessing and maintaining the long-term health of the ecosystem, as unchecked growth or stagnation can lead to diminished package quality, poor tool support, and lagging management systems [11].

To understand how the ROS ecosystem is growing, we first determine the growth in the number of packages in the ROS distro over time relative to two other software ecosystems, PyPI and Julia, using the methodology described in Section III-C. Our results, shown in Figure 3, indicate that growth in all three ecosystems has been linear since 2015. This result indicates that the rate of growth of the ROS ecosystem is healthy and similar to other ecosystems, albeit slower.

Another important metric of growth is the number of software projects that make use of ROS and its associated ecosystem. To approximate the popularity of ROS in client applications over time, we use the GitHub API to determine the creation date of each of the associated repositories for the packages within the client dataset. Our results, shown in Figure 4, show that the total number of ROS client packages is growing exponentially, unlike the linear growth of the distro. As of June 24, 2019, there are over 230,488 ROS packages on GitHub. This result is indicative of a healthy ecosystem that is continuing to attract new users and yield new applications.

Given the steady growth of both the ROS distro and client applications, we examined whether ROS is continuing to attract new developers at a similar rate. To approximate the total number of ROS developers over time, we used the GitHub API to find the date at which a contributor (identified by Github username) made their first commit to a repository within the client dataset. By summing the number of users who have contributed to ROS over time, we construct an estimate of the rate at which the ecosystem is attracting new developers. Figure 5 shows the membership for both the ROS distro and client applications over time. We observe that over 87,564 developers have contributed to ROS client projects on GitHub, and that number is growing exponentially. The number of developers contributing to the rosdistro appears to be increasing at a linear rate, and stands at 8,538 developers as of May 2020, an order of magnitude fewer than the number of all ROS developers on GitHub. To determine if the number of developers is growing proportionally with the number of packages, we took the ratio of packages to developers over time. We found that this ratio was between .35 (1:3) and .63 (2:3) for the distro, over the years from December 2013 to June 2019. This result indicates that the ROS ecosystem has been successful in attracting both new users and contributors over time.

Key Insight: The ROS ecosystem continues to attract new contributors and end users who use ROS for an exponentially increasing variety of applications. However, despite the increased popularity of ROS over time, the number of packages made available by its package management system is growing at a much slower, linear rate, suggesting that users are reusing existing packages rather than creating new packages and expanding the capabilities of ROS.

B. *RQ2: How is the ROS ecosystem structured?*

In this research question, we seek to gain a better understanding of the *social structures* through which individuals and groups collaborate and contribute to the ROS ecosystem.

To do so, we first used the GitHub API to distinguish between packages within the ROS distro that are owned by users and those that are owned by groups. In total, we identified 111 unique users and 186 unique groups that host ROS packages for the Kinetic distro. We subsequently performed a manual classification of the 186 groups and identified six distinct types of group, described below.

Foundational Working Groups: Groups that are organized around a shared area of expertise to provide the fundamental building blocks that are required to construct most robots (e.g., navigation, perception, drivers). They include both the **Core Stacks** that provide essential packages for the ROS framework (i.e., "the plumbing"), and **Working Groups** that offer fundamental packages.

Academic Groups: Labs and research groups affiliated with universities or other academic institutions.

Companies: GitHub organizations that are tied to a particular company or product.

Competition Teams: Groups created to build a robot for a competition (e.g., RoboCup[9] and the DARPA Robotics Challenge).[10]

Interest Groups: Groups that are not directly affiliated with Open Robotics, the maintainers of ROS, which typically are focused on a particular project rather than an area of expertise (e.g., planning).

Singletons: Packages hosted either by a single user, or by a GitHub organization with only one contributor.

Having identified the types of contributor within the distro, we study how their total size (i.e., number of packages) and influence over the ecosystem (i.e., reverse dependencies) has evolved over time, shown by Figures 6 and 7 respectively. Note that we exclude core stacks from these figures to prevent skewing the results since all ROS packages and applications must rely on the core stacks. Companies have continued to represent the largest type of organization within the ecosystem in terms of their number of packages since 2010. Working groups are the second largest type of organization, although

[9]https://www.robocup.org/

[10]https://www.darpa.mil/program/darpa-robotics-challenge

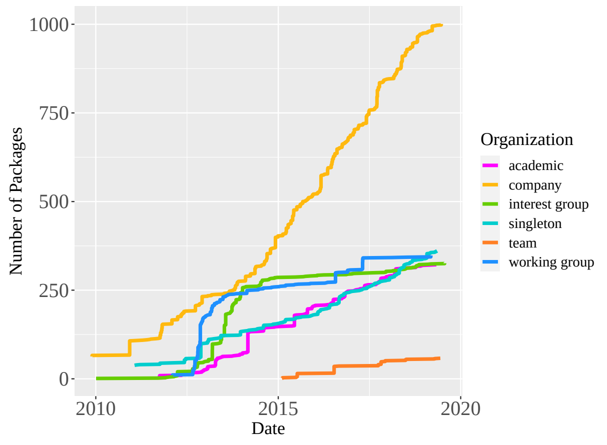

Fig. 6. The total number of packages that belong to a given type of contributor over time.

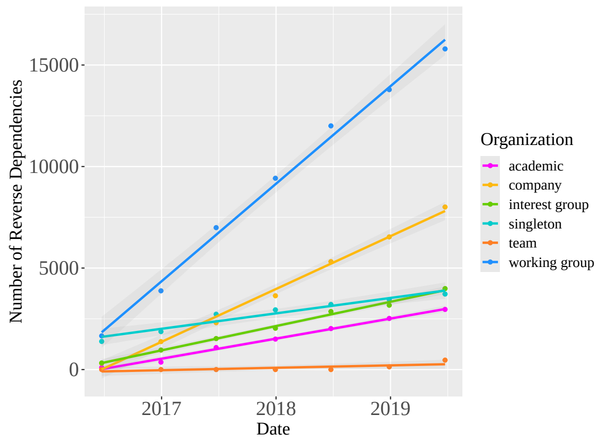

Fig. 7. The total number of reverse dependencies on packages that belong to a given type of contributor over time.

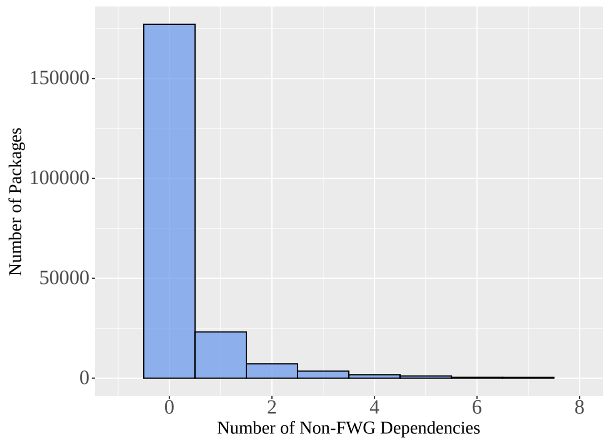

Fig. 8. The number of client packages on GitHub that have dependencies on n packages that are provided by a group or individual other than a foundational working group. Approximately 82% of client packages have zero dependencies on non-FWG packages.

their growth has been stagnant since 2017. Despite being

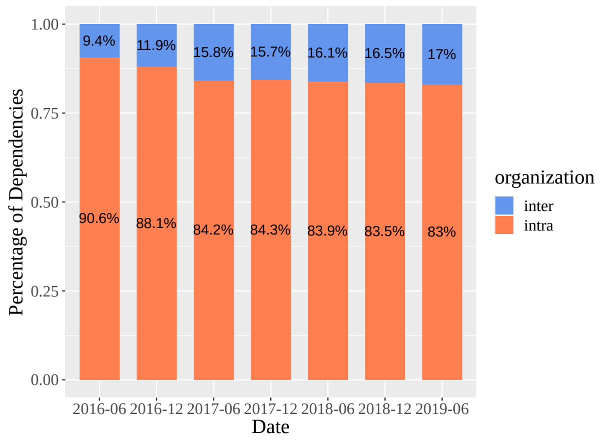

Fig. 9. A breakdown of the source of dependencies on packages other than those produced by foundational working groups. Illustrates the percentage of such dependencies that are on packages from the same organization (intra), versus those from other organizations (inter).

the second-largest type of organization, working groups are acquiring dependent packages (reverse dependencies) at nearly twice the rate (1.8X) as the next-fastest-growing group. For the entire history of the ROS ecosystem, working groups have had more reverse dependencies than the sum total of all other entities (core stacks excluded).

Upon closer inspection of client applications on GitHub, we find that 82% of those applications rely exclusively on packages from the foundational working groups (FWGs), and 99% of Client applications have five or fewer non-FWG dependencies, as shown in Figure 8. Although it is common for software ecosystems to rely heavily on a core set of packages, [7], [13], [14], it is surprising that so many ROS Client applications exclusively rely on the FWGs. These groups are established organizations focused on providing the essential functionality of a robot (e.g., navigation, perception, hardware drivers). By *exclusively* relying on packages produced by FWGs, Clients are neglecting all other types of packages contributed to the ecosystem.

As the ecosystem grows, we would expect that a smaller percentage of packages exclusively rely on core groups, as higher-level libraries and APIs extract their functionality. Figure 11 shows that the opposite is occurring: over time, a greater fraction of packages in the distro are only relying on the FWGs. This means that packages produced by the other organizations (academic, company, interest group, etc.) are continually getting less use.

While FWGs make up approximately 80% of reverse dependencies, there is still collaboration in the ecosystem that does not involve them. To better understand collaboration between other groups in the ecosystem, we examined dependencies on packages other than those produced by FWGs. Specifically, we classified dependencies as either *intra-organizational*, if the packages are hosted by the same GitHub user or organization, or *inter-organizational* if not. Figure 9 shows how the balance of inter and intra-organizational dependencies has evolved over time. We find that the vast majority of non-FWG dependencies are on packages produced by the same organization, indicating limited collaboration between the "satellite" groups that make up the rest of the ecosystem. However, we do also observe that the proportion of inter-organizational dependencies is increasing albeit slowly.

Key Insight: Individuals and organizations within the ROS ecosystem revolve around a small number of highly influential FWGs, which account for over 80% of reverse dependencies. Relatively little collaboration takes place between the distinct individuals and organizations, and most packages depend exclusively on functionality provided by the FWGs.

C. RQ3: How do the foundational working groups contribute to the ecosystem?

Having identified the importance of ROS's foundational working groups, in this question we take a closer look at their responsibilities, health, and relationship to the rest of the ecosystem.

Figure 10 lists each of these foundational working groups and describes their responsibilities. Collectively, the packages provided by these foundational working groups provide the essential building blocks that are required to construct a robot. With the exception of ROS, which composes the core stacks and provides the "plumbing" for ROS itself, each group is organized around a particular area of expertise (e.g., perception). This means of structuring collaboration shows how ROS has tackled the considerable difficulties of writing high-quality, general-purpose robotics software by means of divide and conquer: The foundational working groups allow the relatively limited population of developers that possess the specialist knowledge and skills in a particular subarea of robotics (e.g., planning) to more easily find one another and collaborate, rather than attempting to tackle that subproblem alone. Historically, most these groups trace their roots back to ROS's "Special Interest Groups", designed to bring together developers with a shared interest in a particular topic.[11]

Key Insight: The ROS ecosystem exhibits an extraordinary level of structure and cooperation, facilitated by its expertise-focused foundational working groups, that is rarely seen in other software ecosystems where multiple competing implementations are the norm; ROS developers have chosen to work together towards a common goal rather than compete.

Given the considerable influence of FWGs, we examine the extent to which each of these group has been able to attract new contributors and official members over time. Figure 12 shows the number of official members for each of the corresponding GitHub organizations of each FWG over time. We observe that new membership has slowed over the past few years, and only one new developer has been added to a FWG since the start of 2020. Other groups, such as *ros-simulation*, have not recruited new members since September 2013. The plateauing membership of FWGs may have long-term health implications for the ecosystem as a whole. Presently, the largest FWG has 25 official members, meaning that a small number of people are responsible for maintaining a large portion of the most critical software in ROS.

To account for development in the FWGs by users that are not official maintainers, we looked at the the number of active contributors for each of the FWGs over time, as measured by the number of GitHub users that made at least one commit during a given three-month period, shown in Figure 13. We observe a spike in the number of active contributors at around the end of 2012, which coincides with the release of the ROS Groovy distribution. Since that point, the number of active contributors has fluctuated but remains quasistatic and does not suggest continued growth: The greatest number of active contributors in a given three-month period occurred in the final quarter of 2013.

The influence of FWGs over the ecosystem as a whole suggests that they are stable and functioning well, but as ROS attracts more developers, FWGs risk becoming a bottleneck that leads to stagnation within the ecosystem. Furthermore, having so much of the ecosystem rely on a small number of individuals is generally bad for ecosystem health [17], and is exasperated over long periods of time as key developers are more likely to stop contributing [18].

Key Insight: The number of people maintaining and contributing to ROS's foundational working groups has increased little over the last few years. Given the considerable influence of foundational working groups, this result may ultimately limit the growth of the ecosystem.

V. Discussion

Our analysis of the ROS ecosystem shows that, despite the considerable challenges of building robotics software (e.g., the need for significant expertise in several domains), ROS has been highly successful in attracting developers and end users, and its growth over time is comparable to software ecosystems of a similar age. The results of our study show that the unique structure and nature of collaboration within the ROS ecosystem, embodied by its foundational working groups, is at the root of this success. In this section, we further explore this unique form of collaboration and its potential implications for the future of the ecosystem.

Whereas in most software ecosystems, different groups compete to offer multiple implementations of the same functionality, the ROS ecosystem demonstrates an extraordinary level of cooperation on a single set of implementations through

[11] https://wiki.ros.org/sig

Working Group	Pkgs.	Client	Distro	Members	Responsibilities
ros (a.k.a., Core Stacks)	159	1,543,515	65,847	25	build system, client libraries, geometry, file parsers
ros-drivers	97	21,136	933	19	hardware device drivers
ros-perception	52	110,578	4,010	12	object detection, image processing, computer vision
ros-controls	29	31,626	2,693	8	generic robot controllers, control frameworks
ros-simulation	7	24,990	771	9	ROS interfaces for Gazebo [15] and Stage [16] simulators
ros-planning	70	71,864	4,022	18	navigation, local and global planning, motion planning
ros-visualization	58	20,870	3,067	15	tools for visual debugging
ros-geographic-info	10	1,124	249	6	common interface for mapping and coordinate conversions
ros-infrastructure	1	107	13	3	hosts the ROS distro, build farm, and official website

Fig. 10. An overview of the working groups that provide the foundations of ROS, and a description of their associated responsibilities. **Pkgs.** is the number of packages provided by a working group. **Client** is the number of reverse dependencies from client packages. **Distro** is the number of reverse dependencies from distro packages. **Members** is the number of public GitHub organization members for that working group.

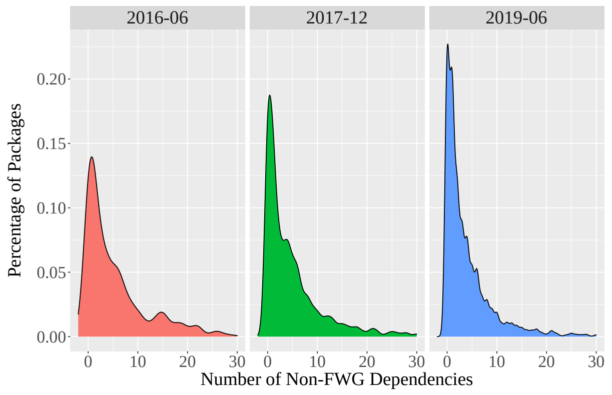

Fig. 11. A density plot showing the fraction of ROS distro packages that rely *only* on the FWGs over time. Observe that an increasing majority of packages do not depend on any non-FWG packages.

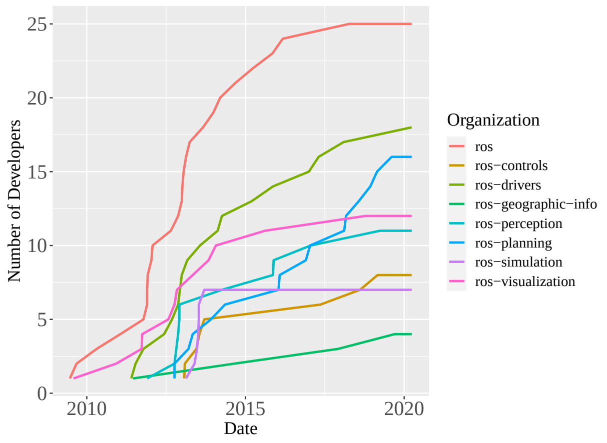

Fig. 12. The number of official GitHub organization members for each of the foundational working groups over time.

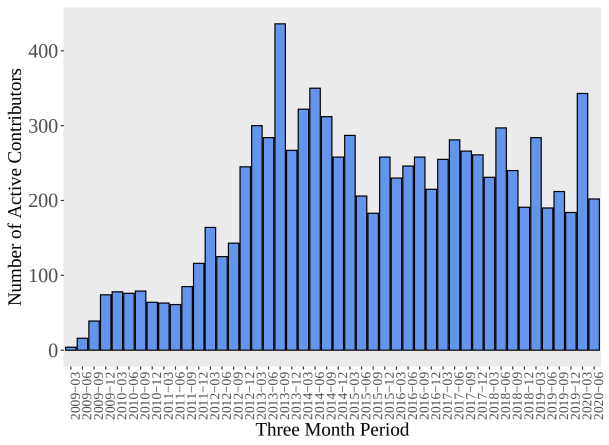

Fig. 13. The number of GitHub users who have made at least one commit to a repository belonging to a foundational working group during a given three-month period.

its foundational working groups. Each of these groups is composed of a relatively small number of experts in a particular subdomain of robotics, and produces software that is used widely throughout the ecosystem. Indeed, ROS was designed from the ground up to encourage *collaborative robotics software development*: For instance, experts in a particular area of robotics (e.g., perception) can collaborate to produce high-quality software that can be reused by others as the building blocks of their own systems [1].

The unique, highly cooperative nature of the ROS ecosystem may help to explain its success during its early stages: by working together to tackle the considerable challenges of writing robust, general-purpose robotics software, developers may avoid duplicating difficult work and instead focus on building something that works and can be used by others sooner. In turn, this may lead to more users and organizations adopting that code, which ensures the survival of the ecosystem.

However, the factors behind the early success of ROS may unintentionally lead to long-term health problems for its ecosystem. In this study, we observe that FWGs account for the majority of reverse dependencies in the ecosystem, and over 80% of ROS code on GitHub does not use functionality beyond that provided by FWGs. Packages developed by organizations or individuals other than those in Figure 10 are rarely successful in convincing others to rely on their packages. When these other packages do gain reverse dependencies, they are almost always from other packages within the same organization. Explaining the reasons behind such behavior is beyond the scope of this study, however, releasing packages with very specific functionality, low reliability, and limited documentation could be among the reasons.

The problems caused by a heavy reliance on FWGs may be exacerbated by the limited number of maintainers and active contributors to those groups, and their slow growth in membership over time [19], [20]. That few developers are responsible for maintaining the most crucial packages makes the ecosystem susceptible to knowledge loss and stagnation. The slow recruitment could be attributed to the required domain expertise, or it may simply be a result of the difficulties of adding new maintainers to open source organizations due to the inherent overhead of communication and coordination.

The ROS framework is at a pivotal point of transitioning to its second major version, ROS2. This transition affords the opportunity to review the positive and negative aspects of the ROS ecosystem, and adopt policies and ideas that ensure it continues to grow as it enters the next stage of its development. To facilitate continued innovation and avoid stagnation, the ROS ecosystem needs to solve the challenge of encouraging greater collaboration, reuse, and code sharing outside of its FWGs. For instance, an improved package distribution can simplify the process of releasing packages to the ecosystem, and make the ecosystem more resistant towards release mistakes. A better search mechanism in the package manager can make it easier for the developers to find and reuse packages most suitable for their needs. Ultimately, introducing tools and methods that simplify the release process while providing some sort of assurance of the quality of the packages, and making it easier to search for packages in the ecosystem can encourage more collaboration among ROS developers outside of the FWGs, which is beneficial to the health of the ecosystem.

VI. Related Work

In a recent study most closely related to ours, Pichler et al. study the interdependencies between ROS packages on GitHub, BitBucket, and the rosdistro, and how quality propagates through the dependency network [4]. Our study is different in (1) the number of ROS packages collected from GitHub (more than 230,000 compared to less than 7,500), (2) our handling dependency versions and collecting of historical data, which supports creation of ecosystem snapshots of the ecosystem for any point of time, and, critically (3) the focus of study. Our study focuses on the ecosystem structure, collaboration, code reuse, and ecosystem health, while Pichler et al. focus on the quality of the ecosystem.

In an empirical study consisting of interviews and a survey with ROS developers, Estefo et al. investigated the difficulties that ROS users encounter when reusing ROS packages, main contribution bottlenecks in ROS ecosystem [21]. They report that 74% of the survey participants that tried to reuse a third-party ROS package failed to do so for various reasons including packages developed for an outdated ROS distribution, lack of documentation, and difficulty configuring for a particular use case. In a separate, prior study, Estefo et al. study code duplication in ROS packages [22]. Specifically, they focus their attention on the duplication of launch files and XML configuration files that are used as part of the application launch process. They show that 25% of all packages that contain more than one launch file contain code clones.

Alami et al. conduct a qualitative study to better understand quality assurance practices within the ROS community [5]. They find that the implementation and execution of QA practices in the ROS community are influenced by social and cultural factors and are constrained by sustainability and complexity. In a talk, the ROSIN group reported that by conducting intensive interviews with members of the ROS community, and analyzing 177 reported ROS bugs, they found that one-third of the ROS bugs are dependency errors [23].

Santos et al. performed various static analyses to measure code and process metrics for the GitHub repositories associated with 180 C++-based ROS packages [24]. They find that applications and drivers tend to have more developers, commits, and raised issues than library code.

Curran et al. develop a series of tools for ROS that can be used to measure the impact and health of individual nodes, packages, and contributors within the ROS ecosystem, and to dynamically determine the nodes and packages from the ecosystem that are being actively used by the community [25]. Repository health is computed using 13 metrics that cover aspects of the repository such as its size, number of subscribers and watchers, and the state of its issue tracker.

Prior studies have also investigated the health and survival of open source projects in general [9], [26], [27]. Samoladas et al. show that open source projects that existed more than 10 years ago continue to evolve, and every new programmer added in a project increases project's survivability by 15.8% [26]. Coelho et al. explore the reasons that popular open-source projects fail by conducting a survey of the maintainers of 104 popular-but-now-deprecated GitHub projects [27]. Valiev et al. find that packages with few reverse dependencies are more likely to become dormant over time, and to ultimately become abandoned [9].

Decan et al. compare three programming language ecosystems (Python's PyPI, R's CRAN, and JavaScript's NPM), and identified the differences in their structure [28]. They found most PyPI packages to be isolated in the dependency graph because they only depend on the rather extensive standard library of Python, similar to our finding about ROS ecosystem. In a separate prior work, Decan et al. carry out a quantitative empirical analysis of the similarities and differences between the evolution of seven packaging ecosystems (ROS not included) [11]. Similar to our findings about ROS, they observe that the dependency networks of software ecosystems tend to grow over time, only a small proportion of packages account for most of the reverse dependencies.

Our study and analysis of the ROS ecosystem is similar in nature to many prior studies performed on other software ecosystems [7]–[9], [13], [14], [29]. Valiev et al. used historical data from 46,547 projects in the PyPI[12] ecosystem, and showed that the number of project ties and the relative position in the dependency network have significant impact on sustained project activity [9].

Wittern et al. studied the evolution of the NPM[13] ecosystem between 2011 and 2015 [7]. Their study includes 185,005 core packages and 114,995 client applications collected from GitHub. They show that the number of packages having one or more dependencies increased from 23.4% in 2011 to 81.3% in 2015. In addition, in 2015, 72.5% of packages had no reverse dependency, while only 4.9% of packages had 6 or more dependents.

German et al. conduct a comparative study of core and user packages in the CRAN R ecosystem, a popular statistical computing project [8]. The authors studied 207 core packages and 2,733 user-contributed packages. They find that user-contributed packages tend to be smaller and have less documentation than core packages. They find that packages typically have few dependencies, and that user-contributed packages depend on more core packages than other user-contributed packages. Similarly Plakidas et al. study the evolution of R ecosystem, and analyze 8,941 packages on CRAN [14]. They find that 54% of packages on CRAN have dependencies and 22.8% have reverse dependencies.

VII. Threats To Validity

Internal: Did we skew the accuracy of our results with how we collected and analyzed information? To avoid skewing our results, we collected all the ROS projects on GitHub that we could find. However, we did not include packages with only a documentation link, and no code. There were also a total of 106 projects which we could not clone due to naming errors, so we excluded those from our dataset. Additionally, some projects in the ROS client dataset may have been clones or student projects—inflating the number of packages relying exclusively on the FWGs. Lastly, since ROS is one of few open-source robotics frameworks, we compared its evolution against Julia and PyPI, although ROS is much more domain specific. The libraries.io datset we used for this comparison was collected externally. Therefore, any errors in their data collection on PyPI and Julia would also be reflected in our results.

External: Do our results generalize? To enable our results to generalize as much as possible, we look at ROS projects on GitHub, so that our sample is as varied as possible. All of our projects are open source, so we cannot make claims about how our results may generalize to proprietary projects.

Replicability: Can others replicate our results? To support others replicating our results, we make our data publicly available: https://doi.org/10.5281/zenodo.3997720.

VIII. Conclusion

In this work, we studied how the ROS ecosystem has grown over time, and how its members collaborate to build robot software. Our results show that ROS has been successful in tackling the inherent challenges of building robotics software, and is continuing to attract new applications, developers, and end users. However, we also observe that number of packages released on the ROS package manager is growing at a much slower rate.

We studied the structure of the ROS ecosystem by identifying its types of contributor, and analyzing the dependencies between packages. We found that a small number of groups, which we term foundational working groups, account for 80% of all dependencies in the ROS ecosystem, and the majority (82%) of ROS applications *exclusively* rely on packages released by these groups.

We further examined collaboration within these foundational working groups. We observe that these groups are highly structured and facilitate a unique form of collaboration and cooperation between between domain experts working on particular robotics subproblems. While this extraordinary level of cooperation may account for some of the early success of ROS, we also observe that membership of these groups has remained relatively stagnant in recent years, which may have long-term health implications for the ecosystem. We briefly outline and discuss how ROS can tackle these challenges and continue to be successful as it transitions to its next major release, ROS2.

[12]https://pypi.org/

[13]https://www.npmjs.com

Acknowledgment

This research was partially supported by AFRL (#FA8750-15-2-0075), DARPA (#FA8750-16-2-0042), and the NSF (#CCF-1750116 and #CCF-1929249); the authors are grateful for their support. Any opinions, findings, or recommendations expressed are those of the authors and do not necessarily reflect those of the US Government.

References

[1] "About ROS," https://www.ros.org/about-ros/, accessed: 2020-01-16.

[2] "The origin story of ROS, the linux of robotics," https://spectrum.ieee.org/automaton/robotics/robotics-software/the-origin-story-of-ros-the-linux-of-robotics, accessed: 2020-01-16.

[3] "Results 2020 user survey," https://discourse.ros.org/t/results-2020-user-survey/13494, accessed: 2020-05-28.

[4] M. Pichler, B. Dieber, and M. Pinzger, "Can i depend on you? mapping the dependency and quality landscape of ros packages," in *International Conference on Robotic Computing*. IEEE, 2019, pp. 78–85.

[5] A. Alami, Y. Dittrich, and A. Wasowski, "Influencers of quality assurance in an open source community," in *International Workshop on Cooperative and Human Aspects of Software Engineering*, ser. CHASE '18, May 2018, pp. 61–68.

[6] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[7] E. Wittern, P. Suter, and S. Rajagopalan, "A look at the dynamics of the javascript package ecosystem," in *13th Working Conference on Mining Software Repositories*, ser. MSR '16, May 2016, pp. 351–361.

[8] D. M. German, B. Adams, and A. E. Hassan, "The evolution of the r software ecosystem," in *European Conference on Software Maintenance and Reengineering*, March 2013, pp. 243–252.

[9] M. Valiev, B. Vasilescu, and J. Herbsleb, "Ecosystem-level determinants of sustained activity in open-source projects: A case study of the pypi ecosystem," in *Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE '18, 2018, p. 644–655.

[10] A. Nesbitt and B. Nickolls, "Libraries.io open source repository and dependency metadata," 2017.
[11] A. Decan, T. Mens, and P. Grosjean, "An empirical comparison of dependency network evolution in seven software packaging ecosystems," *Empirical Software Engineering*, vol. 24, no. 1, pp. 381–416, Feb 2019.
[12] A. Decan, T. Mens, and E. Constantinou, "On the impact of security vulnerabilities in the npm package dependency network," in *International Conference on Mining Software Repositories*, ser. MSR '18, 2018, pp. 181–191.
[13] J. Kabbedijk and S. Jansen, "Steering insight: An exploration of the ruby software ecosystem," in *Software Business*, B. Regnell, I. van de Weerd, and O. De Troyer, Eds. Springer Berlin Heidelberg, 2011, pp. 44–55.
[14] K. Plakidas, S. Stevanetic, D. Schall, T. B. Ionescu, and U. Zdun, "How do software ecosystems evolve? a quantitative assessment of the r ecosystem." in *International Systems and Software Product Line Conference*, ser. SPLC '16, 2016, p. 89–98.
[15] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *International Conference on Intelligent Robots and Systems*, ser. IROS '04, vol. 3, 2004, pp. 2149–2154.
[16] R. Vaughan, "Massively multi-robot simulation in Stage," *Swarm Intelligence*, vol. 2, no. 2-4, pp. 189–208, 2008.
[17] V. Cosentino, J. L. C. Izquierdo, and J. Cabot, "Assessing the bus factor of git repositories," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2015, pp. 499–503.
[18] A. Capiluppi, M. Morisio, and J. F. Ramil, "Structural evolution of an open source system: a case study," in *Proceedings. 12th IEEE International Workshop on Program Comprehension, 2004.*, 2004, pp. 172–182.
[19] M. Nassif and M. P. Robillard, "Revisiting turnover-induced knowledge loss in software projects," in *International Conference on Software Maintenance and Evolution*, ser. ICSME'17. IEEE, 2017, pp. 261–272.
[20] P. C. Rigby, Y. C. Zhu, S. M. Donadelli, and A. Mockus, "Quantifying and mitigating turnover-induced knowledge loss: case studies of chrome and a project at avaya," in *International Conference on Software Engineering*, ser. ICSE'16. IEEE, 2016, pp. 1006–1016.
[21] P. Estefo, J. Simmonds, R. Robbes, and J. Fabry, "The robot operating system: Package reuse and community dynamics," *Journal of Systems and Software*, vol. 151, pp. 226 – 242, 2019.
[22] P. Estefo, R. Robbes, and J. Fabry, "Code duplication in ros launchfiles," in *International Conference of the Chilean Computer Science Society*, ser. SCCC '15, Nov 2015, pp. 1–6.
[23] Y. Dittrich, G. van der Hoorn, and A. Wasowski, "How ROS cares for quality," 2017, ROSCon. [Online]. Available: https://roscon.ros.org/2017/
[24] A. Santos, A. Cunha, N. Macedo, and C. Lourenço, "A framework for quality assessment of ros repositories," in *International Conference on Intelligent Robots and Systems*, ser. IROS '16, Oct 2016, pp. 4491–4496.
[25] W. Curran, T. Thornton, B. Arvey, and W. D. Smart, "Evaluating impact in the ros ecosystem," *International Conference on Robotics and Automation*, pp. 6213–6219, 2015.
[26] I. Samoladas, L. Angelis, and I. Stamelos, "Survival analysis on the duration of open source projects," *Information and Software Technology*, vol. 52, no. 9, p. 902–922, Sep. 2010.
[27] J. Coelho and M. T. Valente, "Why modern open source projects fail," in *Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE '17, 2017, p. 186–196.
[28] A. Decan, T. Mens, and M. Claes, "On the topology of package dependency networks: A comparison of three programming language ecosystems," in *European Conference on Software Architecture Workshops*, ser. ECSAW '16, 2016.
[29] E. Constantinou and T. Mens, "Socio-technical evolution of the ruby ecosystem in github," in *International Conference on Software Analysis, Evolution and Reengineering*, ser. SANER '17, Feb 2017, pp. 34–44.