"They Can Only Ever Guide:" How an Open Source Software Community Uses Roadmaps to Coordinate Effort

DANIEL KLUG, CHRISTOPHER BOGART, and JAMES D. HERBSLEB, Carnegie Mellon University, USA

Unlike in commercial software development, open source software (OSS) projects do not generally have managers with direct control over how developers spend their time, yet for projects with large, diverse sets of contributors, the need exists to focus and steer development in a particular direction in a coordinated way. This is especially important for "infrastructure" projects, such as critical libraries and programming languages that many other people depend on. Some projects have taken the approach of borrowing planning tools that originated in commercial development, despite the fact that these techniques were designed for very different contexts, e.g. strong top-down control and profit motives. Little research has been done to understand how these practices are adapted to a new context. In this paper, we examine the Rust project's use of roadmaps: how has an important OSS infrastructure project adapted an inherently top-down tool to the freewheeling world of OSS? We find that because Rust's roadmaps are built in part by summarizing what motivated developers most prefer to work on, they are in some ways more a description of the motivated labor available than they are a directive that the community move in a particular direction. They allow the community to avoid wasting time on unpopular proposals by revealing that there will be little help in building them, and encouraging work on popular features by making visible the amount of consensus in those features. Roadmaps generate a collective focus without limiting the full scope of what developers work on: roadmap issues consume proportionally more effort than other issues, but constitute a minority of the work done (i.e issues and pull requests made) by both central and peripheral participants. They also create transparency among and beyond the community into what central contributors' plans are, and allow more rational decision-making by providing a way for evidence about community needs to be linked to decision-making.

CCS Concepts: • **Human-centered computing** → **Open source software**; • **Social and professional topics** → *Sustainability*.

Additional Key Words and Phrases: collaboration; common pool resources; open source; Rust language

ACM Reference Format:
Daniel Klug, Christopher Bogart, and James D. Herbsleb. 2021. "They Can Only Ever Guide:" How an Open Source Software Community Uses Roadmaps to Coordinate Effort. *Proc. ACM Hum.-Comput. Interact.* 5, CSCW1, Article 158 (April 2021), 28 pages. https://doi.org/10.1145/3449232

1 INTRODUCTION

Open source software (OSS) has come to fulfill an infrastructure role in the economy. Eghbal [26] highlights OSS projects such as MySQL and Ruby that both OSS and industrial projects depend on heavily, but are themselves non-profit OSS projects. To fulfill an infrastructural role, there needs to be careful coordination among maintainers and users of the infrastructure [68], who are doing the work on behalf of different companies or foundations, or perhaps as volunteers. Good coordination

Authors' address: Daniel Klug, dklug@cs.cmu.edu; Christopher Bogart, cbogart@cs.cmu.edu; James D. Herbsleb, herbsleb@cs.cmu.edu, Carnegie Mellon University, USA, 5000 Forbes Ave, Pittsburgh, Pennsylvania, 15213.

is especially important for infrastructure projects, since by definition the project is an essential underpinning of many other projects: poorly-considered changes can damage these stakeholders more than they would if the project was merely an incidental dependency of other projects, that they could simply swap out for an alternative. Coordination of work in self-organizing systems[27] poses a difficult and important problem in CSCW.

How can software infrastructure projects ensure that they will not only be maintained in the future, but will preserve values that their users depend on? Unlike in commercial software development, in OSS "developer community" [78] software projects there is no manager who has direct control over which features or attributes developers choose to spend their time on, yet these projects still need to somehow coordinate, stabilize, and make visible their development priorities. Open source projects do have governance, but governance models do not generally dictate what features will be added and when. For example, even in the highly orchestrated work in the Linux kernel, there are multiple coordination processes, driven by the open source norm that contributors self-select their tasks [75].

Much preexisting work in CSCW has focused on the tensions between infrastructure contributors' work on infrastructure and their own priorities, often driven by the primary work they do that the infrastructure is intended to support. For example in scientific software written by academic collaborators, short-term paper deadlines can lead people to focus on needed new features over long-term maintainability [68]; on the other hand infrastructure development can offer contributors new opportunities leading them to realign their own priorities [11], perhaps helping build consensus. Researchers have identified a broad spectrum of ways that OSS communities can organize themselves to coordinate development and avoid tragedy-of-the-commons problems [50], but in some cases preexisting social networks among contributors drive much of the work done [46]. Some OSS projects have taken the approach of borrowing planning tools that originated in commercial development, milestones and issue tracking (e.g. Scala [1]), beta testing (e.g. PostgreSQL [2]) or roadmaps (e.g. Rust), despite the fact that these techniques were originally conceived for very different contexts, i.e. strong top-down control and profit motives, in which executives and managers make final decisions about goals and timelines, and rank-and-file developers are responsible for carrying out these plans. Developers in open source communities, in contrast, are often free to choose their own tasks, so this bottom-up power may have an impact on how planning tools work in the open-source world.

Investigating how diverse OSS projects attempt to shape collaboration in a stable visible way requires considering the bottom-up forces at work: developers' motivation whether and how to contribute, users' motivation to choose, support, or influence development, and factors that make one project survive while another fails [98], as well as the top-down techniques leadership employs in projects despite the relative lack of power that OSS leaders have over their communities [75].

In this research we investigate how consensus around a community's direction is constructed, maintained, and evaluated. We approach this by considering how roadmaps as an originally top-down technique from industry are adapted and reconfigured to work for an OSS project. Roadmaps can be understood as a layout of existing plans to make future decisions. They are usually a visualization of further steps [97] intended to be open to later revision [79]. We do not investigate what effect the choice of roadmapping had over some other method of coordination the community could have chosen, or the process of deciding on the use of roadmaps in the first place; but rather how they carried out the particular method they did choose, and its immediate effects during

[1]https://github.com/scala/scala/milestones
[2]https://www.postgresql.org/developer/beta/

one iteration. We look at an OSS roadmap's creation, how it is applied, and how the community evaluates its impact, by addressing the following research questions:

RQ1. What functions does a roadmap serve in an open source community?

RQ2. How does an open source community use a roadmap to fulfill those functions?

Our results show that although a roadmap appears superficially to be an edict from project leaders specifying where resources should be applied, it in fact reflects a consensus among active developers about where they wish to apply their efforts. Its power derives not just from the core developers' ability to accept or reject changes, but because it reassures a would-be contributor that productive developers are already motivated to collaborate with them, if they stick to roadmap-related topics. The roadmap-building process helps these developers reach consensus, and community members use the roadmap throughout the year as a rhetorical resource to cut off digressions and to signal intention to cooperate with community goals.

2 BACKGROUND

These research questions address an apparent mismatch between the idea of volunteers coming together to do work that motivates them, and roadmaps as plans that on their surface appear to be telling people what to do. Prior research has only partly explained how open source collaborators set directions, and literature on roadmapping in corporate settings appears to reveal little about how roadmapping applies to volunteer projects. In this section we describe prior research in both these areas.

2.1 The Problem of Coordinating Developer Effort in Open Source Software

In recent years, the use of OSS has become pervasive [35] among software developers resulting in great economic value of OSS [20, 34] which is, however, largely invisible to the public. Although OSS is often critical infrastructure [26], it is managed very differently from traditional infrastructure. Its users can freely distribute, access, adapt, modify and redistribute source code for their own and for community use. Analyses of OSS projects from various social and organizational perspectives have shown that managing such a project requires taking into account developers' distinct motivations for contributing [5, 15, 38], benefits and rewards of contributing [13, 44, 54], preferred levels of involvement [4, 62], building and managing social capital [66, 80], networking [60, 76, 77], and differing communication and interaction strategies [6, 19, 33].

The varying motivations and characteristics raise the question of how OSS communities *coordinate* to agree to and work towards common goals. We define "coordination" as many individuals deciding how to work together effectively; that is, how to choose tasks that amount to collective progress in a mutually agreeable direction as opposed to working at cross-purposes. OSS contributors and maintainers often work in a distributed and decentralized way, with very little hierarchy or institutional structure [22, 99], and are more likely to engage in projects and tasks based on personal interests [5]. Coordinating and organizing work in OSS projects therefore involves matching the demand for effort (desired features and known bugs that will take time and specialized skills to fix) with supply of effort (volunteers and paid developers who have their own motivations and priorities).

2.1.1 *Supply of and Demand for Development Work.*

Like any software, OSS requires maintenance "to correct faults, improve performance or other attributes, or adapt to a changed environment" [48]. Unfulfilled demand for maintenance may render regular software obsolete. But for infrastructure, the ramifications of insufficient maintenance are magnified because other projects and their users rely on the infrastructure; thus the demand for development effort is greater, coming from a large dependent pool of projects and users. Prior research shows the demand for maintenance work, such as issue fixes, testing, and documentation may depend on many factors: for example, the size of the user base for a particular feature [56], or extent of upstream or interdependent projects [12]. Research on managing OSS requirements [73, 103] shows how demand is discovered, analyzed, prioritized, and validated within discussions and issue requests. Popular projects need help triaging user-reported issues [2, 104]. Infrastructures typically also need coordinators [65] who ensure that individual projects have features needed for an infrastructure-wide release.

Skilled volunteers are motivated by factors such as their strength of identification with the community [38], internal (e.g., self-use) and external (e.g., reputation) motivations [36, 45], a desire to learn [102, 105], or long-term "hobbyist" status, in which developers become more deeply involved and play a critical role in long-term viability [74]. Developers hired by industry also play an increasing role in OSS development[28]. Firms are more likely to pay developers to participate as a way of sharing the cost of innovation, creating demand for their complementary products or services, establishing their technology as de facto standard, or attracting improvements or complements for their products [100]. However, industry support for OSS projects carries some risk of discouraging volunteers. But this can be mitigated by transparency in decision-making[101] and negotiation of governance, membership, ownership, and control over production[58].

2.1.2 Matches and Mismatches in Effort Supply and Demand. Participants in OSS infrastructure are generally free to contribute anywhere. These individual decisions bring about an emergent allocation of effort across projects. But besides the decision-making of individual participants, it is unclear what mechanism influences participants to apply effort where there is the greatest need. In contrast, it is clear that in commercial firms participating in markets, the forces of supply and demand determine price, a strong signal guiding the allocation of resources [9]. Economists Dalle & David [21] were puzzled about "how, in the absence of directly discernible market links between the producing entities and 'customers,' the output mix of the OSS sector of the software industry is determined. Yet, to date, the question does not appear to have attracted any significant research attention". We were unable to find research that addresses this issue in the years since then. The study of requirements management points out the difficulties of discovering, articulating, and implementing needed features even when development effort is plentiful [103]. The lack of development effort has been documented in the highly publicized *Heartbleed* bug [23], but we are not aware of systematic studies of under-supply or how to recognize and address it. In total, the research seems to support the conclusion that there currently is no general mechanism closing the gap between demand for and supply of effort, except for the perceptions and decision-making of individual developers. Yet infrastructure and effort mismatch are difficult for participants to see[68].

2.1.3 Organizing and Allocating Work in Open Source Software Projects. OSS project leaders face tradeoffs between openness and fostering a productive collaboration. Decision-making behind closed doors can cause conflict that discourages volunteers, since they may feel their preferences are not being considered [40]. But too much visibility into disagreements among leadership can also lead to uncertainty among volunteers that decisions may not be firm and their contributions may not end up being used [82].

With often only partial control and limited means of enforcement, OSS project leaders may rely on social factors such as their technical reputation and community traditions to promote a vision of the project's direction [55, 64]. Publishing schedules and roadmaps can help get developers to identify with and take responsibility for community goals [64]. Leaders may develop formal policies and guidelines for collaboration to give structure to developers' work [40], and may assert the authority to reject additions in a given software release [55].

Prior research has identified implicit ways that core members influence newcomers and peripheral members to adopt cultural norms and practices. Hemetsberger and Reinhardt [37] describe a number of mechanisms that core members of open-source projects such as KDE use to enculturate peripheral members: for example that project's manifesto[3] may discourage non-like-minded contributors, and KDE's leaders enforce norms through mailing list discussions and code review processes. Crowston and Shamshurin [18] showed that core members of successful Apache incubator projects were more communicative than in unsuccessful projects, and were more likely to use pronouns in a way that suggested inclusiveness of the peripheral community. Gallivan [32], however, argues that rigorous control, standardization, and measurability ("McDonaldization") helps open source projects achieve common objectives in virtual, distributed environments where trust relationships are difficult to form; in particular despite potentially many mutual trust relationships in open-source communities, control is a one-directional relationship from core to periphery.

2.2 Roadmaps in Commercial and OSS Development

Roadmaps are plans for use of resources over time, often created in iterative and reflective processes [61] and intended to be open for changes [79]. The goal is to lay out existing plans, future decisions, and visualize further steps [79, 97] that may be revised based on project results [41]. In commercial contexts, developer resources and needs are coordinated explicitly by management, and roadmaps are a tool to create, implement, and manage software in alignment with company strategies, product life-cycles, and audiences [24, 30, 96].

In Software Product Management (SPM), roadmaps are a communicative tool for knowledge sharing [81], consensus-reaching, and individual interpretation of goals by people involved in development processes [47]. For example, product roadmaps present features to manage product stages [49, 96], select and assign requirements [25], and connect teams to ensure the success of a product within a larger time frame [30, 96]. To create roadmaps, information about audiences, their characteristics, and needs is usually collected beforehand [7]. As a communicative tool, roadmaps describe what will be (or should be) achieved in which way in a project, and how it will meet business objectives [57].

Many OSS projects generate roadmap documents, including large OSS communities such as React [67], Facebook Libra [84], Scala [85], and QT [95] as well as industry-produced OSS such as AWS CloudFormation [14] and industrial coalitions like Open Service Broker [3]. These roadmaps appear to have varying roles in the communities. Some seem to have multiple versions as if they are being maintained and revisited, while others are one-time descriptions of envisioned future features. However it is difficult for a casual observer to tell what importance these roadmap documents play. In this research we choose the Rust Language community as particular example to examine its use of roadmaps.

3 CASE STUDY: ROADMAPS IN THE RUST LANGUAGE PROJECT

Based on our theoretical propositions, we selected the Rust programming language as a single-case study. It is appropriate both because of its popularity and its openness. Its popularity as infrastructure means that there are many users who may pressure participants to make and implement good choices about features and priorities. Its openness means that a rich variety of data about the Rust compiler community's working and decision-making processes is available from blogs, forums, and GitHub repositories. Thus we have the opportunity to study in great detail a community

[3]https://manifesto.kde.org/

making and implementing consequential choices together. This constitutes what Yin [106] calls a "revelatory case" as it provides "an opportunity to observe and analyze a phenomenon previously inaccessible to social science inquiry."

The Rust programming language has been growing into the role of a popular and important part of the software infrastructure [59]: many individuals, subteams, and outside organizations have a stake in its future. Rust is promoted as being suitable for infrastructural code where performance and reliability are important, such as web browser engines or in hardware devices with limited resources; for that reason it is used by numerous big tech companies [70], such as Facebook and Mozilla. The Rust community is organized in teams [69] and work groups. It has a large and active social community, with a variety of blogs, chat rooms, forums, GitHub discussion threads, and in-person conferences and meetings worldwide.

The Rust community adopted, then evolved, a roadmapping process, adding to the purposes that the roadmap serves over time. After the release of version 1.0 in 2015 the Rust core team initiated a process to organize and prioritize future work and to define future goals in all areas of Rust, citing a need to *sequence feature additions* to avoid later rework, and *prioritize features that would solve many problems or benefit many users* [51]. In 2016, the Rust team refined their *RFC* (request for comments) process; RFCs are documents proposing significant changes to the project [93]. An overarching roadmap process was added to define initiatives as *rallying points* with concrete goals, fixed time frames, and clear commitments from individuals. This process involves building consensus in the community on project-wide goals, then proposing these goals for community discussion through an RFC, and finally advertising and publishing the agreed upon goals as a yearly roadmap.

The Rust core team [69] released the first Rust roadmap in February 2017 [94]. To create the roadmap, the core team gathered priorities through a Rust community survey [92] and a commercial user survey with companies using Rust [91]. For the 2018 roadmap, in addition to the annual survey [83], the core team asked the Rust community to blog and post ideas for Rust in the next year [87]. The Rust community submitted 100 blog posts with suggestions for the roadmap. The core team then collected and incorporated the suggestions into an RFC for discussion and review [71], and released the roadmap in March 2018 [88]. The 2019 roadmap followed a similar process [86]: building on 73 community blog posts [72], a survey [90], and the RFC discussion, the core team created the roadmap and released it in March 2019 [89]. Unlike previous years, the 2019 roadmap was explicitly organized around Rust's team structure, and made explicit mention of those teams having their own roadmaps.

The process thus has evolved over four years to more thoughtfully sequence development, to prioritize the worst problems and the most users, to elicit both broad (survey) and deep (narrative blog post) input from the community, to devolve some planning to the separate teams in the form of team-specific roadmaps, and, finally, to ensure that chosen initiatives are are not only needed, but actually supported by people willing to commit to working on them.

3.1 The 2018 Rust roadmap

The 2018 roadmap, available at https://github.com/rust-lang/rfcs/blob/master/text/2314-roadmap-2018.md, lays out four major goals: shipping a 'Rust 2018' edition of the language, creating more documentation support for intermediate-level Rust programmers, encouraging global spread of Rust by adding internationalization support and links with local Rust groups, and finally, strengthening the compiler's work teams and their leadership. The document goes on to identify several concrete things that need to be done to support those areas.

The 2018 compiler release that is the Roadmap's first goal focuses on support for four identified use cases for the language: network services, WebAssembly (i.e. use in web browsers), command line applications, and use in embedded devices.

The document also specifies a rough schedule for the year, starting with design work in February and March 2019, focusing on RFCs , "buckling down" in April through July, focusing on development work, "Fun!" in August through November, focused on forward-looking, exploratory features, and "Reflection" in December.

The document ends with a brief discussion of 'rationale, drawbacks, and alternatives'.

3.2 Other Rust documents

The Rust project publishes a great many documents defining their product, their community, and its governance. Documents that are somewhat standard for open source projects, available at the project's GitHub site at https://github.com/rust-lang/rust, include a "README.md" telling users what Rust is and how to install it, copyright and license files positioning the work legally, "CONTRIBUTING.md" and "CODE_OF_CONDUCT.md" files laying out high level community norms for how developers can contribute and how they are expected to interact, and "RELEASES.md" describing the change history of the project at a high level. Beyond that the project provides a wealth of deeper information, including "The Rust Programming Language"[4] that teaches the language itself, the "Guide to Rustc Development"[5] teaching how the compiler works and going into great depth about contribution norms and governance.

As of September 2019, beyond the compiler project itself, the Rust community had 147 other GitHub repositories under its organizational umbrella, including the collection of RFCs and the discussions around them https://github.com/rust-lang/rfcs (the annual roadmaps are found among these RFCs); the other repositories hold auxiliary tools, bots, websites, and documents.

4 METHODOLOGY

Understanding how communities work is often a complex research matter that requires large data collection. Our research benefits from the Rust community being very open and communicative; they produce lots of publicly accessible artifacts that document community and software related activities. Therefore, a high volume of data is available to researchers about how the community builds, maintains, and evaluates consensus about its direction.

4.1 Data Collection

To analyze what functions a roadmap serves to the Rust community and how they use it to fulfill those functions, we collected the following publicly available data produced by the Rust community.

4.1.1 Yearly Rust Roadmaps. We focused on the community-wide 2018 Rust roadmap and collected the official roadmap document [88]. Because the Rust community introduced its first roadmap for 2017, analyzing the 2018 roadmap allows to look at the past and the following years' roadmap to include the community's own reflection on how the roadmap was used. We collected 97 of the 100 blog posts [71] (3 were no longer retrievable) submitted by Rust community members during the process of creating the 2018 roadmap, written in response to the core team's call for goals and directions for Rust in 2018.

4.1.2 Direct records of Rust compiler project work. The Rust community uses the RFC process to find consensus on proposed substantial changes to the language, standard libraries, and also to

[4]https://doc.rust-lang.org/book/index.html
[5]https://rustc-dev-guide.rust-lang.org/

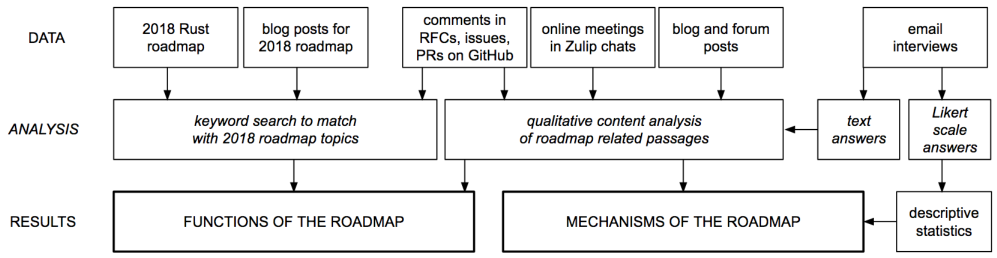
DATA
ANALYSIS
RESULTS
2018 Rust roadmap
blog posts for 2018 roadmap
comments in RFCs, issues, PRs on GitHub
online meetings in Zulip chats
blog and forum posts
email interviews
keyword search to match with 2018 roadmap topics
qualitative content analysis of roadmap related passages
text answers
Likert scale answers
FUNCTIONS OF THE ROADMAP
MECHANISMS OF THE ROADMAP
descriptive statistics

Fig. 1. We gathered software engineering artifacts, GitHub comments, blog posts, and email interviews. We analyzed software engineering artifacts and a set of pre-roadmap blog posts for roadmap-relevant content. We analyzed GitHub comments, chats, blog posts, and interview text through qualitative coding, and statistically analyzed Likert-scale answers in the email interviews. We describe the functions and mechanisms of the roadmap by drawing on all three types of analysis.

community standards. Issues and PRs (pull requests: i.e. proposed specific edits to code) are often linked to RFCs and show where the actual coding work of all contributors happens and to what Rust contributors allocate their time and effort. Comments in these RFCs, issues, and PRs involve discussions among contributors and teams. We scraped all code and discussion contents of GitHub repositories associated with the Rust compiler project from Jan 1, 2018 to Dec 31, 2018, the time frame for the 2018 roadmap. This data allowed us to analyze how much of which kind of work (coding work and discussion work) by which people (core or peripheral people) adhered to the topics called for in the roadmap.

4.1.3 Records of argumentation and discussion. To understand how participants used the roadmap as a resource for argumentation during the year to affect decisions and priorities, we collected excerpts from across several communication channels used by the Rust community (Table 1) in which people explicitly mentioned the roadmap (i.e. explicit mentions of the word "roadmap" or "road map"):

- **Compiler project work** We extracted roadmap mentions from the corpus of RFC, issue, and PR discussions described above, excluding any mentions in the roadmap's own RFC#2314 (https://github.com/rust-lang/rfcs/pull/2314).
- **Posts in Rust blogs and forums** Some participants in the Rust project, as in many OSS communities, maintain personal and official community blogs to post about updates, goals, ideas, or critical thoughts. To gather samples of participants explicitly using the existence and content of the roadmap as a resource in argumentation about the project direction, we searched for roadmap mentions in posts of main publicly accessible Rust blogs (*Rust Blog* (https://blog.rust-lang.org), *Inside Rust Blog* (https://blog.rust-lang.org/inside-rust), *Read Rust* (https://readrust.net), *This Week in Rust* (https://this-week-in-rust.org)) and the *Rust Internals* forum (https://internals.rust-lang.org) from Jan 1, 2018 to Apr 23rd, 2019. This time period was extended past the end of the year specifically to include posts advocating for content for the 2019 roadmap, since they might contain reflections about the 2018 roadmap and its content. The 2019 call for roadmap blog posts explicitly asked Rust contributors to reflect on Rust in 2018 [86].
- **Online team meetings:** As an OSS community, Rust contributors characteristically are distributed all over the world which is why meetings are mainly held online. The Rust compiler team holds weekly meetings on the collaborative chat software Zulip (https://rust-lang.zulipchat.com) to update, manage, monitor, and plan work, in working groups and

Table 1. total number of collected data and excerpts of each data that mention "roadmap"

	RFC, issue, and PR comments on GitHub	Blog and forum posts	Blog posts reflecting on roadmap	messages in Zulip chat threads	***total***
data collected	135,234	3,394	73	58,901	*197,602*
mentions of "roadmap"	59	110	28	144	***341***

throughout the larger community. Zulip conversations are semi-public: members need to create a free account and log in to participate, thus setting a low barrier to read or contribute to the discussions. Anticipating that team members and contributors might use these online meetings to discuss matters related to roadmaps and roadmap processes, we searched for roadmap mentions in Rust team meetings held on Zulip starting from Jan 1, 2018 and extending a few months beyond the end of 2018 to Apr 23rd, 2019, so as to also include reflection on the 2018 roadmap that happened in early 2019.

In the textual data collected from GitHub comments, online meetings, and blog post we identified a total of 118 participants by name and username who made at least one comment or multiple comments regarding roadmaps. We anonymized participants (P001, P002, ..., P118) chronologically by appearance in the different data sources. Five participants were core team members, 28 were members of other teams, 85 were non-team members, and five were identified as working group members (see Fig. 2).

4.1.4 Email Interviews. In addition to our data mining, we conducted short emailed structured interviews with Rust contributors to contextualize some of our findings about the two research questions. We generated a sample of community members stratified by level of community involvement. To find highly involved members, we collected a list of all Rust team members and all blog post authors for the 2018 road map (99 people at the time of the sampling). For the less-involved members we chose a random sample of the same size, out of all other committers to the compiler project who listed emails on their Github profiles. After later data cleaning (people with multiple or invalid emails), we ended up with a list of 190 candidates. We mailed the interview to those candidates, and 39 people responded (20.5% response rate). 24 of those identified themselves as belonging to a Rust team, and 15 said they did not (see Fig. 2). As the email interviews were conducted anonymously, we could not match participants with our existing list of participants in Rust forums. Therefore, interview participants were anonymized and numbered separately (PS001, PS002, ..., PS039). The interview questions asked Rust contributors about their experience with and opinions on all Rust roadmaps of any year. The questions are shown in Appendix A.

4.2 Data Description and Analysis

Our case study includes data collected from GitHub to reconstruct the allocation of effort in code work, textual data from several Rust community sources to analyze the communicative aspects of creating and using roadmap documents and discussing work effort related to roadmap topics, and answers from structured email interviews with Rust community members to triangulate results obtained from the collected textual data. To analyze how the Rust community creates, uses, and evaluates roadmaps, we decided to follow a mixed-method approach as quantitative or qualitative methods by themselves could not sufficiently address our research questions [16]. We simultaneously used quantitative and qualitative data collection methods and followed a convergent approach to separately analyze the data sets and then combine results in the interpretation. Following this methodological approach, our goal was to generate a complete and deep understanding [17] of how roadmaps are used to discuss and allocate effort.

We used a quantitative technique to estimate the proportion of work done during the year that was relevant to the community-wide 2018 roadmap. We developed a roadmap topic heuristic for determining whether a given piece of text was relevant to topics mentioned in the roadmap. The purpose of the heuristic was to give us an objective way of saying whether a unit of discussion or coding was part of the roadmap or not, and secondarily, which part of the roadmap it pertained to. The heuristic starts with some hand-written regular expressions built around topics we found in the 2018 roadmap, and identifies text by applying those regular expressions, and also by making inferences about topics of "related" items, for example inferring that an issue that claims to track an RFC probably addresses the same topic as the RFC does. Its output is a list of all issues, pull requests, RFCs, and commits, tagged as "in roadmap" or "not in roadmap". The algorithm is described in detail in Appendix B. We applied this heuristic to create two datasets:

- To identify where ideas in the roadmap came from, we applied this heuristic to the 97 retrievable blog posts that answered to the 2018 Rust call for roadmap blogs, generating a mapping between 2018 roadmap topics and the blogs which the core team drew on in preparing the roadmap. We also identified whether each post was written by a member of a Rust team.
- To estimate the influence of the roadmap on work done throughout 2018, we applied the heuristic to all Rust project issues and Rust project PRs, creating a data set consisting of one record per PR or issue, tagged with: a (possibly empty) set of roadmap topics, the context of discussion (issue, or PR), the type of contributor (Rust team member or not), and two measures of work effort: discussion work and coding work. Discussion work was operationalized as the number of characters of English text in PR and issue discussion threads (after removing code snippets); coding work was operationalized as lines of code added or removed in the Rust project commits associated with PRs.

These datasets distinguish individual participants as "team" vs "non-team": we defined these by scraping the membership of all Rust teams (Figure 2) from the project's governance page [6].

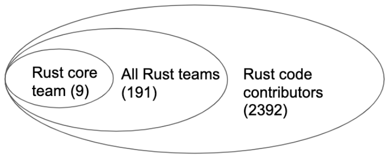

Fig. 2. We classify Rust community members as "team" (191 people) or "non-team" (other participants, whether contributing code or other effort), depending on whether they were listed on some team in https://www.rust-lang.org/governance on January 3, 2019. Although organizational literature often refers to "core" and "peripheral" members, to avoid confusion we use the word "core" for the 9-person team the Rust governance page identified as the "core team", "team" to refer to the 191 members of teams (including core), and "non-team" for the larger community periphery.

As a supplemental check on sources of ideas in the roadmap, we manually inspected the ten commits to the 2018 Rust roadmap document in the GitHub Rust RFC project and summarized the changes, looking for introductions of new topics (none were found). This was a small, relatively

[6]https://www.rust-lang.org/governance, in January 2019 as retrieved from https://web.archive.org/web/20190103220022/https://www.rust-lang.org/governance

Table 2. Examples for applying codes to excerpts and sorting them into categories

Excerpt	Code	Category
*“a key step in any successful WG is going to be forming a **roadmap**”*	point out need for a roadmap	creating a roadmap
"it’s not the kind of change that’s targeted for the roadmap this year”	rejecting an RFC	using roadmaps to decline allocating effort

informal effort since a cursory check showed that little substantial change to the RFC had been made during the discussion. Complementing this quantitative technique, we also created a dataset of hand-coded roadmap mentions from project work, team meetings, and blogs. Table 1 shows the amount of data collected from each source. We extracted 341 excepts that mentioned "roadmap" or "road map" from the collected data, tracking for each excerpt its author and source.

In our case study of textual data collected from GitHub comments, online meetings, and blog posts, we followed a qualitative content analysis approach [42, 52] to characterize what people said about roadmaps in the excerpts of these sampled Rust online artifacts.

We decided to use qualitative content analysis for our case study because the method is rooted in social research but is not linked to any particular science or concepts [43]. This makes it a very useful approach to study documents and artifacts across various data sources [8]. Content analysis is profitable for mixed-method research as it comprises quantitative and qualitative methodology and qualitative content analysis in particular allows the researcher to extract manifest and latent information from different textual data [10].

We used a data-driven open coding approach across all collected excerpts from the text-based data sources (GitHub comments, online meetings, blog posts) [52]. We performed inductive coding and created preliminary codes to construct a coding scheme while processing through the qualitative data. Open codes from all data sources were then combined into larger categories. In total, we generated 91 codes (see Table 2 for code examples), that were then sorted into eight categories (see Table 3).

Throughout the open coding process, the research team ensured a common shared agreement of generated and applied codes. The coding of each varying textual data set (GitHub comments, online meetings, blog posts) was based on the consistent use of codes by one researcher and the subsequent review of generated and applied codes by a second researcher. In this process, little disagreement was found. In such cases, the two researchers met to review, discuss, and refine the disagreed upon codes in relation to the data source and to which research question the coded excerpt relates most. Through this discussion and refinement, all disagreements were solved and codes were mutually validated. This way of ensuring validity in qualitative research through agreement is an established approach in the CSCW community [53] and matches our inductive coding approach for a qualitative case study across varying textual data sources.

In addition to analyzing blog posts and online meetings, the structured email interviews served to collect additional data to triangulate results we observed [29]. Interview questions asked about how roadmaps influence decision-making, how helpful roadmaps are for the community, and how roadmaps match personal work priorities (see Appendix A). We analyzed the numeric responses, shown in Table 4. To identify themes in the textual responses, one researcher grouped responses to each question into categories, and another researcher reviewed and challenged the categorizations.

5 RESULTS

In the following two subsections we answer the research questions: what does the roadmap accomplish for the Rust community (RQ1), and how does it do so (RQ2).

Table 3. Number of excerpts and number of codes applied per category

Category	Num. excerpts	Num. codes applied
Creating a roadmap	134	17
Using roadmap to decline allocating effort	33	13
Pointing effort to roadmap topics	26	12
Executing a roadmap	81	28
Asking about a roadmap	11	4
Linking to roadmap documents	28	2
Praising the use of a roadmap	13	6
Criticizing the use of a roadmap	15	9
total	***341***	***91***

Table 4. Summary of responses to email interview. Q1-3 asked for textual explanations accompanied by a Likert-style question on a five-point scale, where 3 would be a neutral answer, and 5 means the roadmap is high in influence on respondent's activities, helpfulness to them, and in alignment with the respondent's priorities. * = team and non-team differ (t-test, p<0.05). Questions are given in Appendix A

Question	Likert answers (mean)			Text answers (count)	
	overall	Team	Non-Team	Team	Non-Team
Q1 influence (1-5 scale)	2.8	3.2	2.3*	10	6
Q2 helpful (1-5 scale)	4.1	4.2	4.0	11	7
Q3 priorities (1-5 scale)	3.5	3.7	3.1*	11	3
Q4 improve (text)	-	-	-	15	4
Q5 years (numeric)	3.7	3.8	3.5	-	-
Q6 team (yes/no)	-	24 yes	15 no	-	-

5.1 Functions of the Roadmap

Building and using the roadmap appeared to serve neither the extreme of forcing team members' agenda on a wider community, nor letting the broader user community choose a direction. Rather it allowed team members and others to identify areas of consensus around project goals, and keep focus on those goals through the year.

5.1.1 Reaching consensus of purpose among team members. The Rust team put out a call at the beginning of 2018 asking the community to submit *"blogposts reflecting on Rust in 2017 and proposing goals and directions for Rust in 2018"*. An analysis of those posts and the eventual 2018 roadmap suggests that the Rust team indeed succeeded at soliciting input from people outside their team structure: only 18 of the 97 retrievable blog posts collected were authored by people listed as team members or alumni.

However, the blog posts responding to the solicitation did not seem to be a major source of novel ideas from outside the central community; the resulting roadmap document was a synthesis of shared ideas from many sources. Most (23 of 30) of the roadmap topics we could find in the blog posts were mentioned by both team and non team blog posts. Only three topics were mentioned only by team members, and four only by non team members. No single blog post contained more than 12 of the topics, suggesting that the roadmap really is a synthesis of many perspectives, not

Table 5. This table quantifies of two types of effort (discussion and code contribution) applied by the Rust community, broken down by roadmap-relatedness and type of effort. The "total" figures show most discussion and coding was about non-roadmap items; however the Bytes per issue and lines per PR figures show that there was more effort per item about roadmap items.

	Total issue text	÷	# issues	=	Bytes per issue	Total lines of code	÷	# PRs	=	lines per PR
Roadmap	31.6 MB	÷	2899	=	10915.0	246K	÷	680	=	362.2
Non Roadmap	78.8 MB	÷	9092	=	8662.2	923 K	÷	3320	=	277.9

simply a codification of an existing consensus. Nor did the RFC-style process for accepting the roadmap after the core team had created it elicit completely new ideas from the community; rather discussion consisted mostly of clarification and acceptance. The roadmap changed little from when the core team proposed it on Jan 29, 2018, and its adoption on March 5th. Discussion (51 general comments and 20 comments linked to lines in the document) led to little change during that time. Besides typos, formatting, and clarifications, the main substantive change was a rewording to more strongly emphasize compiler performance. In short, the process did not appear to generate innovative new directions, but rather a consolidation of ideas that already had support but had not previously been gathered together.

5.1.2 Focusing work during the year. Analysis of effort expended by the Rust community during 2018 demonstrates that the 2018 roadmap was neither followed religiously nor ignored completely. Rather it represented a community focus, in the sense that its initiatives attracted proportionally more coding and discussion per issue than issues not on the roadmap. Table 5 quantifies two types of effort applied by the Rust community, broken down by type of effort (contributing to discussion in GitHub threads, or writing code).

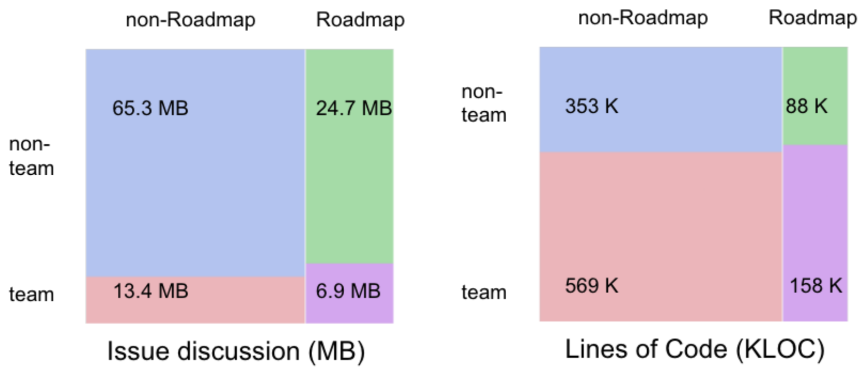

Fig. 3. Volume of discussion (left) and coding (right), broken down by team (n=191)/non-team (n=2392) members, and by roadmap/ non-roadmap issues. Left figure measures discussion in megabytes; right figure measures lines of code in pull requests in thousands of lines of code. Non-roadmap matters dominated in volume, for both discussion and code. Non-team members did most discussion; team members did some what more coding overall. Note that team members do more work per person, but there are vastly more non-team members; and that roadmap issues involve more work per item than non-roadmap issues (see Table 5).

Roadmap matters constitute a minority of the work, but receive outsize attention. In the Rust compiler project's issue and PR threads, the Rust community generated 121,457 comments across

11,991 different discussion threads during 2018 discussing proposed and ongoing development on the Rust compiler. The hottest threads (i.e. Rust project issues or pull requests with the most bytes of discussion) were more likely to be roadmap topics – 6 of the top 10 largest issue threads were roadmap topics, but overall only 2899 out of 11991 (24%) of issues related directly to the roadmap, as measured by our heuristic ($\chi^2 = 6.989$, p=.0082). In other words, roadmap topics were a community focus, but the long tail of smaller efforts actually constituted most of the discussion. Discussion of roadmap-related issues constituted on average 27.8% more text per issue than non-roadmap issues. Roadmap issues included more text than non-roadmap issues (p=.0092, 2-tailed t-test of log-transformed byte counts of issue discussions).

Although 21.1% of the lines of code added and deleted were roadmap-related, the same focus relationship applied; only about 17.0% of PRs worked on were associated with the roadmap, but these roadmap PRs were more substantial changes, averaging 30.4% more lines of code per PR (p<.0001, 2-tailed t-test of log-transformed lines of code per pull request).

Thus although the majority of issues discussed and code changes proposed are not envisioned in the roadmap, the ones that are in the roadmap consume proportionally more effort per issue, especially from frequent contributors. The roadmap appears to serve as a focus of attention while still allowing for a great deal of work outside its boundaries. Not everything the community agrees on requires consensus-building or needs to be in the roadmap; some priorities, such as bug fixing, are obvious.

When asked whether they followed the roadmap personally, twelve of the interviewees (PS002, PS006, PS007, PS008, PS016, PS018, PS020, PS021, PS023, PS027, PS036, PS039) replied that Rust roadmaps set a common direction for the community. Some emphasized common focus (*"I think they give a clear focus point for the year, what the community wants to work on next, (...) see if we accomplished our goals and what our next ones can or should be"* –PS008, email interview), while others emphasized an open, non-prescriptive attitude (*"Ostensibly, they should not be called roadmaps, but they are helpful in the sense that they set *general* priorities. Of course, a lot of other things outside the roadmap will be worked on as you cannot command volunteers to do otherwise"* –PS016, email interview). Another said: *"Roadmaps are independent of the actual work that we can invest, so they can only ever guide"* (PS021, email interview).

5.1.3 *Prioritizing work for the core.*

Team members pay more heed to roadmap priorities than non-team members do. Although the roadmap is pitched as a description of general community priorities, there is evidence that some people, both team and non-team, perceive the roadmap as especially relevant to the activities of team developers, and less important or binding for non-team participants. Four of the 16 people who answered our interview question about how roadmaps influenced their decisions about what to work on indicated that the roadmap applied most to highly-involved people. One respondent, who claimed a fairly low (2/5) influence from the roadmap, said: *"I started contributing for my own learning and experience, roadmaps didn't influence me to start contributing but do influence what I contribute now that I'm more involved"* (PS023, email interview). Another, who claimed high influence (5/5) from the roadmap, said, *"I'm on the core team and work on subteams so the roadmap is directly related to the work I do"* (PS039, email interview).

The amounts of text and code generated by participants support the idea that team members were more likely to pay attention to roadmap issues: 87 out of the 108 team members who contributed code in 2018 (81%) added a comment to at least one roadmap-related issue, while only 39% of non-team contributors did so (1065 out of 2757); this difference in proportions was significant ($\chi^2 = 75.98$, p<.00001). Still, the bulk of the work they did, regardless of role, was on non-roadmap matters. 34.1% of the text team members wrote in issue comments was in roadmap-related issues

(6.9MB out of 20.3MB in Figure 3), and 21.7% of the lines of code they wrote were in roadmap-related PRs. Contributors not in teams had a similar proportion of roadmap work, with 27.4% of issue comment text and 20.0% of code lines written being roadmap-relevant. It seems that teams' proportionally greater preference to work on roadmap issues at the individual issue level does not result in a vastly greater proportion of roadmap work done, by volume; this might be explained, for example, by team members "touching" many issues in which they do not do the bulk of the work.

Although some developers have very particular issues that they prefer to work on, others, especially team members, took cues from the roadmap when setting their own priorities. In the interviews, people gave equivocal answers to the question of whether Rust roadmaps influence their decision of what work to contribute: the average choice was 2.8 on a 1-5 scale, slightly closer to "not at all" than the scale's midpoint of 3. People who said they were on a team rated this higher (3.2) than non-team respondents (2.3) (t-test, p<.01). Four of the people who elaborated on this question said that they felt the roadmap was mostly relevant to important issues addressed by team developers. Two specifically indicated that the presence of a feature in the roadmap gave developers confidence to work on that feature, knowing that some change they wanted to work on would be taken up by others in the community. One said they *"only contribute drive-by* [i.e. as a one-off edit without much community engagement] *when an itch needs scratching; roadmaps do influence where I see a chance of scratching actually result in usable changes to the language"* (PS001, email interview). In short, the roadmap provides encouragement to work on certain issues, for certain people, but most developers do not feel constrained to work on roadmap initiatives.

Influence between individual priorities and the roadmap ran both ways among interviewees. People rated agreement with the roadmap's priorities slightly positively: an average of 3.5 on the 1-5 scale, with team members significantly higher at 3.7 than non-team members at 3.1 (t-test, p<.05). Out of 14 who chose to elaborate, causality ran both ways: two said their priorities matched the roadmap's because they helped write it, and three said they just happened to agree with its priorities; on the other hand five said they pursued roadmap initiatives because they didn't have their own priorities, and three said they *disagreed* with the priorities but valued the importance of having a shared goal more than getting their own way. One person said the roadmap priorities were too vague to resolve the disagreements that were relevant in their working team.

5.1.4 Creating external visibility. Some saw the roadmap as also serving to communicate the intentions of the Rust community to those outside the community, to make the community's trajectory more predictable. When first proposing the roadmap process, the author of the proposal listed among its goals *"Advertise our goals as a published roadmap."* and *"Celebrate our achievements with an informative publicity-bomb"* [1].

In our interviews, four of the 14 people (PS001, PS005, PS006, PS038) who answered our question about why roadmaps are valuable indicated that they helped the project communicate its vision and intentions outside the project. One said the roadmap helps users plan by giving them *"(...) a sense of which unstable features are OK to use in a project that's planning to switch to stable in a reasonable time frame"* (PS001, email interview). Another respondent found them helpful as a way to judge their own plans to use the language: *"I consider Rust to still be a young language that is not yet finalized, depending on the direction it goes it could be a deal-breaker for me"* (PS038, email interview).

5.1.5 Building a sense of group identity. In online team meetings on Zulip, the largest number of roadmap mentions concerned creating roadmaps. The majority of mentions (63%, 91/144) were by a single participant, P008, a core team member who championed both the roadmap and the formation and strengthening of Rust's team structure in 2018. P008's rhetorical use of the roadmap included emphasizing the need to start a separate roadmap, e.g. for a subproject, and suggesting and collecting roadmap topics for existing roadmaps. P008 emphasized benefits of having roadmaps, such as successful collaborative work (*"a key step in any successful WG is going to be forming a **roadmap**"* –P008, core team member, online meeting), structuring work processes (*"I think encouraging people to outline a roadmap with specific steps is a good idea"* –P008, core team member, online meeting), or reaching bigger and shared goals. They argued, for example, that creating roadmaps is worth the effort put into it (*"it's worth taking the time to make the roadmap"* –P008, core team member, online meeting) and that work time is needed to create roadmaps.

A few non team members also mentioned a need for roadmaps to organize work effort (*"We need to open issues first, and to have some kind of roadmap"* –P040, non team member, online meeting) but were overall less committed to making decisions of how to create and manage roadmaps (*"not sure if we want to wait and collect all the appropriate tool/subteam roadmaps and publish one collectively?"* –P038, non team member, online meeting). In online meetings, non team members rather make comments to show mostly strong support for roadmap creation in reaction to suggestions made by core team members (*"I think a roadmap is definitely a good idea, something to get working groups working towards a goal could be helpful in keeping them active"* –P045, non team member, online meeting) or praise the effort made by team members to create and apply roadmaps (*"I applaud all this, can't agree more on everything :)"* –P048, non team member, online meeting).

Team members understood roadmaps as a useful planning tool for ongoing and future work and to manage working groups and attract more contributors by presenting work areas and goals. Roadmaps functioned to manifest topics working groups should focus on over a certain time which is why team members gently pushed towards creating roadmaps, for example by suggesting a new group begin with a very lightweight alternative to the complex community-wide process: (*"I'm not imagining very long 'roadmaps', just some bullets"* –P008, core team member, online meeting). The team members' effort to have contributors and working groups start roadmaps illustrates the need and the goal to organize and manifest work in written form and how especially the core team tries to manage larger and general goals for the distributed Rust community.

5.1.6 Summary. The Rust community's team members began with a diverse set of priorities as individuals: the roadmap process was a way for team members to decide on a consensus focus of attention, and commit to applying themselves to those things during the year. It gave them a way to define themselves more strongly as a group by knowing that they had a shared purpose, and there is some evidence that it gave peripheral participants a way to assert their identity with a group, and for in-group members to gently channel outside contributions away from distracting alternate paths. Although the process explicitly listened to input from outside the community of Rust team membership, it did not in practice bring significant new ideas from outsiders into the conversation.

5.2 Mechanisms of the Roadmap

A roadmap written and never referred to again might simply gather dust and bear no relation to subsequent activity. The Rust community however appears to take the roadmap seriously after it is written. Individuals used it to gauge whether their own ideas are likely to be supported by others, to strengthen formation of teams, to discuss and argue with each other to encourage or discourage proposed efforts, and to reflect on progress.

5.2.1 Assembling work groups. Although the roadmap, during its creation phase, helps the whole community build consensus about its overall goals, developers also use it to find each other and form collaborations to do more particular tasks.

In blog posts, team and non team members alike mentioned personal or project roadmaps as a way to inform each other about work activities and promote plans of action. For example, they referred to detailed goals in project roadmaps (*"There's a bit more detail on the project roadmap"* –P091, non team member, blog post) or pointed out roadmap goals for work groups (*"Embedded is one of the four target domains in the Rust 2018 Roadmap (...)"* –P084, non team member, blog post). In one issue comment, a contributor motivated others to contribute ideas to the roadmap call for blog posts to influence the Rust roadmap (*"Please write a Rust 2019 blog post and express this concern. I think if enough of us do that, we can influence the roadmap"* –P021, team member, issue comment).

Core team members in early 2018 pushed for creation of formal working groups for domains that were defined as focus in the roadmap. In blog posts, team members emphasized that work effort would be aimed at domain working groups (*"the primary focus of this year's project is (...) the domain working groups that we kicked off with our 2018 Roadmap"* –P076, core team member, blog post) and team leaders advertised to the community to allocate their resources to domain working groups. Blog posts at the time announced new working groups for a domain or argued for reorganizing existing working groups to better meet roadmap goals (*"The dev-tools team should be reorganised to continue to scale and to support the goals in this roadmap"* –P077, team member, blog post).

Conversely, although the roadmaps are not promoted as being a complete list of things to work on, they also serve to pre-warn developers that some things they might work on would not likely attract much support or collaboration. In some RFC, issue, and PR comments, team members used the roadmap to refer to the overall direction Rust should take. Even without definite future goals, the mere existence of a roadmap process served to reject proposals not matching potential goals. This included explanations such as, it is not the right time, not the right trend (*"While the details of roadmap is still in play, (...) this seems like a clear expansion with insufficiently strong motivation"* –P008, core team member, RFC comment), or not the right perspective (*"I don't think that major rework of enums currently aligns well with our current priorities or those priorities we are likely to set in the upcoming roadmap"* –P008, core team member, RFC comment).

5.2.2 Discouraging non-roadmap RFCs and basis for rejecting proposals. Team membership appears to affect how people talk about the roadmap. Roadmap mentions by team members in RFC, issue, and PR comments intended to point contributors to roadmap topics and away from the RFC proposal (*"I'd like to draw attention to our 2018 roadmap"* –P012, core team member, RFC comment). However, team members often still valued developers' ideas and motivated future work. For example, they presented the prospect that a feature could make it on the upcoming roadmap (*"could be an interesting thing to consider for next year's roadmap"* –P002, team member, RFC comment).

The roadmap gave a justification for team members and especially for core team members to dismiss proposals that did not fit well with the community's vision for Rust, or that would take too much significant effort away from current efforts. In comments on GitHub, the roadmap was mostly mentioned as an argument in discussions for team members to decline proposed RFCs when they did not seem to fit roadmap goals (*"it's not the kind of change that's targeted for the roadmap this year"* –P002, team member, RFC comment). This argumentative strategy seems to go against the perception of the roadmap as a mere guideline, instead posing roadmap goals as delimiting boundaries to which work and effort should be allocated. Only some comments gave additional explanations for declining such RFCs in relation to the roadmap. For example, the roadmap was treated as a strict work plan when proposals are a possible threat to achieving roadmap goals (*"I am pretty worried if we delay now we will have a hard time delivering on our roadmap for the year"* –P007, team member, issue comment). Team members also used the roadmap to reinforce something perceived to be a true but insufficient reason to end RFC or issue discussions, for example, when a proposal did not generate enough community interest (*"There hasn't been a lot of activity on this RFC (...) it also doesn't particularly fit the roadmap"* –P008, core team member, RFC comment). They also defined adequacy of RFC discussions against the roadmap goals (*"I also don't think this RFC is of high enough priority to the Rust roadmap to devote a lot of attention to reaching consensus"* –P018, core team member, RFC comment). In other words, features that did not match the roadmap were not worth the effort to find consensus within the community.

Although non-team members rarely used the roadmap to argue against features, one contributor mentioned the roadmap to speak out against an issue (*"Finally, 'abstract type's are not close on the roadmap"* –P011, non team member, RFC comment). Beyond its role then in consolidating a consensus when it was created, the roadmap also is used as an argumentative resource for encouraging work on shared goals, and discouraging work (and even extended discussion of work) that risks becoming a distraction.

5.2.3 Reason to promote particular issues and PRs. We found in issue and PR comments non team members mostly mentioned the roadmap by referring to, supporting, or emphasizing roadmap goals in issue discussions or when asking about clarification or the status of roadmap goals. They often argued in favor of features that were on or related to the roadmap (*"Using build systems other than/in addition to Cargo is explicitly a goal in the 2018 roadmap"* –P028, non team member, issue comment). They often mentioned the roadmap as a strong reference to argue for working on or implementing features, sometimes even with reference to previous roadmap topics (*"Cargo being able to integrate into larger build systems was I think on the 2017 roadmap"* –P009, non team member, RFC comment). In discussing work effort in issues and PRs, non team members also pointed roadmap goals out to others (*"Note for those who haven't seen yet: macros 2.0 is apparently slated to be _stable_ later this year, according to the proposed roadmap"* –P021, team member, issue comment).

5.2.4 Shared basis for later reflection. The Rust roadmap process promises a retrospective reflection at the end of each year [1]. As part of that, the Rust core team asked people to reflect on 2018's roadmap when posing ideas for the 2019 Roadmap. The reflections within these posts mostly evaluated progress on the roadmap's particular initiatives. For example, posters praised progress on WebAssembly (*"2018 has been a really cool year for WASM and Rust"* –P116, team member, blog post reflecting) or on futures and async/await (*"A lot of progress was made on Futures async/await in 2018"* –P110, team member, blog post reflecting). People also criticized lack of progress in unfinished tooling (*"Tooling was a large part of the goal for Rust 2018. If one gets lucky, tooling around editor and IDE support can "just work", but many times it doesn't."* –P071, non team member, blog post reflecting) or missing libraries. Other posts commented on the features themselves, claiming that changes made had no actual benefit for the users or were mistimed.

Reflections about the process itself were relatively rare. Developers mentioned that community collaborative work processes had not yet improved as planned and that the community still needed to better manage exhaustion and time spent on topics in general (*"many of the key contributors to rustc (...) were put under an enormous amount of pressure to get their changes shipped by the deadline"* –P086, non team member, blog post reflecting). Moving into 2019 as the efforts to reflecting on 2018 waned, blog posts mentioning roadmaps mostly highlighted work group achievements, such as developments in the Rust package manager, cargo; WebAssembly goals and stabilization; and the growth and increased productivity of Rust teams. This seems consistent with the 2019 roadmap's shift in emphasis towards team-specific roadmaps.

In our email interviews, 19 people (PS002, PS005, PS006, PS007, PS008, PS013, PS016, PS018, PS021, PS023, PS025, PS028, PS029, PS030, PS032, PS035, PS036, PS037, PS039) responded to our question about how roadmaps could be improved; all but two of these were people on teams. Most of the suggestions seemed aimed at reinforcing the roadmap's role as a commitment to achieve goals. The most common suggestion (7 respondents: PS006, PS007, PS008, PS028, PS030, PS032, PS035) was better reflection about the process, in most cases at the end of the year during preparation of the next roadmap. One respondent said: *"It'd be nice to have a retrospective that examines how much work for the year kept to plan, and to give a summary of how the language advanced in the desired direction"* (PS007, email interview). Seven respondents were satisfied with the process (PS002, PS036, PS037) or said they had no opinion (PS005, PS013, PS023, PS029), but the rest had ideas for improvements. Other suggestions were: less ambitious goals, more specific/concrete goals, and better estimation of effort levels. Only two non-team members responded to this question; one of these called for more stakeholder involvement, saying: *"Figuring out low threshold way of bringing library stakeholders into the projects where minimal time commitment is paramount"* (PS018, email interview).

5.2.5 Summary. The intention and process for creating a roadmap gave the community an opportunity and shared artifact around which to talk about and balance priorities, and define boundaries and shared purpose when forming teams. During the year it was in effect, community members used it in online discourse as justification for discouraging off-topic work, and as justification for encouraging on-topic work. It also tipped the balance for individual decisionmaking about work allocation by providing evidence that on-topic efforts would be supported by other community members. Afterwards it served as a standard against which to evaluate progress over the year.

6 DISCUSSION

Rust's roadmap process strikes a balance between openness to new ideas and people, and unifying around common goals. As a popular programming language, there are many potential contributors who could be welcomed and encouraged to help; but as mentioned above in Subsection 2.1.3, eliciting help from the peripheries of a community requires a balance between welcoming openness, and predictable direction. Rust's process seems to strike that balance by creating some ceremony around the transition from openness to direction: they welcome input when building the roadmap, then visibly commit to one direction when the roadmap is released. Although few new ideas from outsiders appear to enter the roadmap through this process, they are enumerated, summarized, and listened to. The fact that new ideas from outsiders have a non-zero chance of being heeded may well be important for encouraging participation, just as the infinitesimal but non-zero chance of winning a lottery is effective in encouraging broad participation.

Another advantage of the transparent roadmap creation process is that it confers legitimacy on the governing process [31]. A document with no visible grounding in such a process might not be trusted as out of date, or as one individual's interpretation of the community's goals, or even as intentions of a sponsoring organization like Mozilla. In contrast, by offering prospective contributors the ability to gain knowledge and trust of a community's true intentions, Rust might be allowing them to more quickly gain a sense of belongingness to the community, a well-studied motivator for contribution [38]. The fact that we observed non-team participants encouraging others to work on PRs relevant to the roadmap suggests that they may be visibly signalling their commitment to the community by demonstrating their familiarity with the roadmap.

When individual contributors can trust that planned work will be done by others in a known timeframe, "divide and conquer" approaches to coordination may become more viable. Howison and Crowston [39] found concurrent development of dependent contributions to be rare in open source. When studying how open source projects performed complex multi-person tasks, Howison and Crowston only observed developers either immediately adding contributions when the necessary supporting code was already in place, or deferring contributions in the hopes that someday that support would become available. They did *not* observe a pattern of *multi-person interdependent work*, in which one developer proceeded on a feature, trusting that another developer would be writing supporting code at the same time. We hypothesize that such *co-work* may be more common in projects that provide some trustable signal about others' intentions. Searching for such examples in Rust would be fruitful future work.

Team members, particularly the core team itself, play an important role in curating suggestions and articulating a common vision. The core team influences the consensus built and maintained by the roadmap process by:

- Framing community survey questions and requests for pre-roadmap blog posts, then choosing among the answers to build a coherent set of initiatives.
- Using their visibility and respect to argue for their vision publicly, in blog posts, RFC and issue discussions, forums, team meetings.
- Holding voting privileges over RFCs and merge rights for PRs; as mentioned earlier, while most accepted RFCs do not align with the roadmap, the roadmap is sometimes used a way to frame rejection of RFCs, usually that are problematic for other reasons.
- A roadmap allows core team members to take a role similar to a manager; this can be seen, for example, in P008's strategy in steering team and contributor effort by using the roadmap as an agreed upon validation.

7 IMPLICATIONS FOR OTHER PROJECTS

A case study is useful for providing a deep example of how a process has played out in the real world: as such it can provide experiences that other projects can learn from, but other projects considering roadmapping need to consider how it applies to their own context.

A project may want to consider a roadmapping process if it is struggling to balance diverging priorities and wants to strengthen a sense of shared direction. Based on our observation of a single case, we suggest the following guidance:

- Actively solicit input from the larger community of developers as well as the core team. As we saw in this case, the overlap in ideas can be very helpful in identifying areas of consensus that already exist, and in letting those harboring ideas lacking in consensus that there is unlikely to be significant effort, in the aggregate, applied to their ideas.
- Adopt a non-zero number of ideas from the community. It seems likely that in order to keep the larger community engaged and interested, a few of the ideas from beyond the core team should make it into the roadmap.
- The evaluation process should be open and fair. As with any form of governance, fairness and openness convey a sense of legitimacy around the decision-making and enhance the likelihood that the community will accept and act on the roadmap.
- Don't expect all – or even most – of the development work and discussion to focus on roadmap items. Nevertheless, significant progress on these items can be made, especially by the most frequent contributors.
- Reflecting on the community's progress against the roadmap and on the process by which the roadmap was constructed can be helpful in creating future versions.

As we caution in the next section, however, this paper describes Rust's experience building a roadmap process for its own particular needs. It is not clear how this process would need to be different for a community building different software, with different developers, for different users.

8 THREATS TO VALIDITY

Our results rely in part on detailed qualitative analysis. Qualitative studies mostly do not aim at generalizability but at providing "a rich, contextualized understanding of human experience through the intensive study of particular cases" [63]. We looked at the Rust community as a case study example on how OSS communities use roadmaps as organizational tools to manage and allocate work effort to shared work goals. Interviewees may not have been representative of the entire community; although our response rate was fairly high, there is a long tail of contributors, and there may be some self-selection bias especially among low-volume contributors.

We do not know how typical Rust is of OSS communities with regard to its roadmap, so we only speculate about how our findings might apply beyond Rust.

We identified a specific list of roadmap topics, and classified issues, PRs, and RFCs according to those topics using a heuristic, described in Appendix B, that may undercount what work is or is not from the roadmap. The boundaries of these topics are not well-defined, since features interact, and work on a non-roadmap feature may be needed where it interacts with a roadmap feature, or vice-versa. However we relied on titles and labels assigned by the community themselves, and our mapping from roadmap topics to labels in many cases had a great deal of face validity.

We do not attempt to tease out the effectiveness of roadmaps as a coordination mechanism, as compared to other ways of governing. Our focus was on understanding how this community constructed and used roadmaps. Future work could address questions of effectiveness by, for example, comparing quality, productivity, or community satisfaction before and after roadmap adoption

9 CONCLUSIONS

In this work we set out to understand the functions of roadmaps for the Rust community, and how they used it to fulfill those functions. To do this, we qualitatively examined the creation, management, and reflection on consensus through the roadmap process, and estimated the proportions of roadmap-related work done throughout the planned year.

We have shown that roadmap's purposes included building and legitimizing consensus, focusing and prioritizing collective attention, particularly for team members, building group identity, and creating external visibility for the community's plans.

The community accomplishes these purposes by assembling work groups around the roadmap's structure, using roadmap goals as justification for directing people towards roadmap-related work, and by using the roadmap to ground reflection at the end of the year when planning for the next year.

The power that the roadmap has to influence contributors' choices during the year comes from the fact that it comprises exactly those initiatives where collaborators are willing to help. Its transparent process provides evidence of that willingness to other developers who are deciding where to contribute their effort. During the roadmapped year, instead of strictly constraining activity, the roadmap rather functioned to nudge contributors to work on collectively agreed upon topics in case their focus would wander off to other, individually motivated, topics. In this way, the roadmap enables the community to guide itself to areas of mutual interest, rather than commanding effort on shared goals.

It thus guides the community, without the need to exert hierarchical power, and provides a useful prediction about future development for people working on dependent projects.

REFERENCES

[1] Brian Anderson. 2016. Feature: north-star. https://github.com/brson/rfcs/blob/north-star/text/0000-north-star.md Last accessed 13 January 2020.

[2] John Anvik, Lyndon Hiew, and Gail C Murphy. 2006. Who Should Fix This Bug?. In Proc. International Conference on Software Engineering (Shanghai, China) (ICSE '06). ACM, New York, NY, USA, 361–370.

[3] Open Service Broker API. 2019. Roadmap & Release Planning. https://github.com/openservicebrokerapi/servicebroker/projects/1 Last accessed 13 January 2020.

[4] A Barcomb, A Kaufmann, D Riehle, K Stol, and B Fitzgerald. 2018. Uncovering the Periphery: A Qualitative Survey of Episodic Volunteering in Free/Libre and Open Source Software Communities. IEEE Trans. Software Eng. (2018), 1–1.

[5] Hoda Baytiyeh and Jay Pfaffman. 2010. Open source software: A community of altruists. Comput. Human Behav. 26, 6 (Nov. 2010), 1345–1354.

[6] Stefan Kambiz Behfar, Ekaterina Turkina, and Thierry Burger-Helmchen. 2018. Knowledge management in OSS communities: Relationship between dense and sparse network structures. Int. J. Inf. Manage. 38, 1 (Feb. 2018), 167–174.

[7] Willem Bekkers, Inge van de Weerd, Marco Spruit, and Sjaak Brinkkemper. 2010. A Framework for Process Improvement in Software Product Management. In Systems, Software and Services Process Improvement. Springer Berlin Heidelberg, 1–12.

[8] Mariette Bengtsson. 2016. How to plan and perform a qualitative study using content analysis. NursingPlus Open 2 (2016), 8–14.

[9] Yochai Benkler. 2002. Coase's Penguin, or, Linux and "The Nature of the Firm". Yale Law J. (2002), 369–446.

[10] Bruce Lawrence Berg, Howard Lune, and Howard Lune. 2004. Qualitative research methods for the social sciences. Vol. 5. Pearson Boston, MA.

[11] Matthew J Bietz, Eric P S Baumer, and Charlotte P Lee. 2010. Synergizing in Cyberinfrastructure Development. Comput. Support. Coop. Work 19, 3-4 (July 2010), 245–281.

[12] Christopher Bogart, Christian Kästner, James Herbsleb, and Ferdian Thung. 2016. How to Break an API: Cost Negotiation and Community Values in Three Software Ecosystems. In Proc. International Symposium on Foundations of Software Engineering (Seattle, WA, USA) (FSE 2016). ACM, New York, NY, USA, 109–120.

[13] Yuanfeng Cai and Dan Zhu. 2016. Reputation in an open source software community: Antecedents and impacts. Decis. Support Syst. 91 (Nov. 2016), 103–112.

[14] AWS Cloudformation. 2018. CloudFormation Public Coverage Roadmap. https://github.com/aws-cloudformation/aws-cloudformation-coverage-roadmap Last accessed 13 January 2020.

[15] J Coelho, M T Valente, L L Silva, and A Hora. 2018. Why We Engage in FLOSS: Answers from Core Developers. In Intl. Workshop on Cooperative and Human Aspects of Software Engineering (CHASE). 114–121.

[16] John W Creswell and Vicki L Plano Clark. 2017. Designing and conducting mixed methods research. Sage publications.

[17] John W Creswell and Cheryl N Poth. 2016. Qualitative inquiry and research design: Choosing among five approaches. Sage publications.

[18] Kevin Crowston and Ivan Shamshurin. 2016. Core-Periphery Communication and the success of free/libre open source software projects. IFIP Advances in Information and Communication Technology 472 (2016), 45–56. https://doi.org/10.1007/978-3-319-39225-7

[19] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository. In Proc. Conference on Computer Supported Cooperative Work (Seattle, Washington, USA) (CSCW '12). ACM, New York, NY, USA, 1277–1286.

[20] Carlo Daffara. 2012. Estimating the economic contribution of open source software to the European economy. In The First Openforum Academy Conference Proceedings. books.google.com.

[21] Jean-Michel Dalle, Paul A David, and Others. 2003. The allocation of software development resources in 'open source'production mode. SIEPR-Project NOSTRA Working Paper,(15th February)[Accepted for publication in Joe Feller, Brian Fitzgerald, Scott Hissam, Karim Lakhani, eds. , Making Sense of the Bazaar, forthcoming from MIT Press in 2004] (2003).

[22] Premkumar Devanbu, Pallavi Kudigrama, Cindy Rubio-González, and Bogdan Vasilescu. 2017. Timezone and Time-of-day Variance in GitHub Teams: An Empirical Method and Study. In Proc. International Workshop on Software Analytics (Paderborn, Germany) (SWAN 2017). ACM, New York, NY, USA, 19–22.

[23] Zakir Durumeric, Frank Li, James Kasten, Johanna Amann, Jethro Beekman, Mathias Payer, Nicolas Weaver, David Adrian, Vern Paxson, Michael Bailey, and J. Alex Halderman. 2014. The Matter of Heartbleed. In Proc. Internet Measurement Conference (Vancouver, BC, Canada) (IMC '14). Association for Computing Machinery, New York, NY, USA, 475–488. https://doi.org/10.1145/2663716.2663755

[24] Christof Ebert. 2007. The impacts of software product management. J. Syst. Softw. 80, 6 (June 2007), 850–861.

[25] Christof Ebert and Sjaak Brinkkemper. 2014. Software product management–An industry evaluation. J. Syst. Softw. 95 (2014), 10–18.

[26] Nadia Eghbal. 2016. Roads and Bridges: The unseen labor behind our digital infrastructure. Technical Report. Ford Foundation.

[27] Anna Filippova and Hichang Cho. 2016. The Effects and Antecedents of Conflict in Free and Open Source Software Development. Proc. Conf. on Computer Supported Cooperative Work & Social Computing (CSCW) (2016), 705–716.

[28] Brian Fitzgerald. 2006. The Transformation of Open Source Software. MIS Quarterly 30, 3 (2006), 587–598.

[29] Uwe Flick. 2018. An introduction to qualitative research. Sage Publications Limited.

[30] Samuel A Fricker. 2012. Software product management. In Software for People. Springer, 53–81.

[31] Archon Fung. 2006. Varieties of Participation in Complex Governance. Public Administration Review 66, s1 (2006), 66–75.

[32] Michael J. Gallivan. 2001. Striking a balance between trust and control in a virtual organization: A content analysis of open source software case studies. Information Systems Journal 11, 4 (2001), 277–304. https://doi.org/10.1046/j.1365-2575.2001.00108.x

[33] Mohammad Gharehyazie, Daryl Posnett, Bogdan Vasilescu, and Vladimir Filkov. 2015. Developer initiation and social interactions in OSS: A case study of the Apache Software Foundation. Empirical Software Engineering 20, 5 (Oct. 2015), 1318–1353.

[34] Shane Greenstein and Frank Nagle. 2014. Digital dark matter and the economic contribution of Apache. Research Policy 43, 4 (May 2014), 623–631.

[35] Gordon Haff. 2018. How Open Source Ate Software: Understand the Open Source Movement and So Much More. Apress.

[36] A. Hars and Shaosong Ou. 2001. Working for free? Motivations of participating in open source projects. In Proc. Hawaii International Conference on System Sciences. 9 pp.–.

[37] Andrea Hemetsberger and Christian Reinhardt. 2009. Collective development in open-source communities: An activity theoretical perspective on successful online collaboration. Organization Studies 30, 9 (2009), 987–1008. https://doi.org/10.1177/0170840609339241

[38] Guido Hertel, Sven Niedner, and Stefanie Herrmann. 2003. Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel. Research Policy 32, 7 (July 2003), 1159–1177.

[39] James Howison and Kevin Crowston. 2014. Collaboration through open superposition: a theory of the open source way. Miss. Q. 38, 1 (2014), 29–50.

[40] Chris Jensen and Walt Scacchi. 2010. Governance in open source software development projects: A comparative multi-level analysis. In IFIP International Conference on Open Source Systems. Springer, 130–142.

[41] Hans-Bernd Kittlaus and Samuel A Fricker. 2017. Software Product Management: The ISPMA-Compliant Study Guide and Handbook. Springer.

[42] Florian Kohlbacher. 2006. The use of qualitative content analysis in case study research. In Forum Qualitative Sozialforschung/Forum: Qualitative Social Research, Vol. 7. Institut für Qualitative Forschung, 1–30.

[43] Klaus Krippendorff. 2018. Content analysis: An introduction to its methodology. Sage publications.

[44] Sandeep Krishnamurthy, Shaosong Ou, and Arvind K Tripathi. 2014. Acceptance of monetary rewards in open source software development. Research Policy 43, 4 (2014), 632–644.

[45] K Lakhani. 2005. Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects. Perspectives on Free and Open Source Software (2005), 3–21.

[46] Charlotte P Lee, Paul Dourish, and Gloria Mark. 2006. The human infrastructure of cyberinfrastructure. Comput. Support. Coop. Work (2006), 483–492.

[47] Jung Hoon Lee, Hyung-Il Kim, and Robert Phaal. 2012. An analysis of factors improving technology roadmap credibility: A communications theory assessment of roadmapping processes. Technol. Forecast. Soc. Change 79, 2 (Feb. 2012), 263–280.

[48] M M Lehman, J F Ramil, P D Wernick, D E Perry, and W M Turski. 1997. Metrics and laws of software evolution-the nineties view. In Proceedings Fourth International Software Metrics Symposium. IEEE, 20–32.

[49] Andrey Maglyas, Uolevi Nikula, and Kari Smolander. 2013. What are the roles of software product managers? An empirical investigation. J. Syst. Softw. 86, 12 (Dec. 2013), 3071–3090.

[50] M Lynne Markus. 2007. The governance of free/open source software projects: Monolithic, multidimensional, or configurational? Journal of Management and Governance 11, 2 (2007), 151–163.

[51] Niko Matsakis. 2015. Priorities after 1.0. https://internals.rust-lang.org/t/priorities-after-1-0/1901 Last accessed 13 January 2020.

[52] Philipp Mayring. 2004. Qualitative content analysis. A companion to qualitative research 1 (2004), 159–176.

[53] Nora McDonald, Sarita Schoenebeck, and Andrea Forte. 2019. Reliability and inter-rater reliability in qualitative research: Norms and guidelines for CSCW and HCI practice. Proceedings of the ACM on Human-Computer Interaction 3, CSCW (2019), 1–23.

[54] Rebeca Méndez-Durón. 2013. Do the allocation and quality of intellectual assets affect the reputation of open source software projects? Information & Management 50, 7 (Nov. 2013), 357–368.

[55] Martin Michlmayr, Francis Hunt, and David Probert. 2007. Release management in free software projects: Practices and problems. IFIP Int. Fed. Inf. Process. 234, December 2006 (2007), 295–300.

[56] A Mockus, D M Weiss, and Ping Zhang. 2003. Understanding and predicting effort in software projects. In 25th International Conference on Software Engineering, 2003. Proceedings. IEEE, 274–284.

[57] Jürgen Münch, Stefan Trieflinger, and Dominic Lang. 2019. Product roadmap–from vision to reality: a systematic literature review. In 2019 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC). IEEE, 1–8.

[58] Siobhán O'Mahony and Beth A Bechky. 2008. Boundary organizations: Enabling collaboration among unexpected allies. Administrative science quarterly 53, 3 (2008), 422–459.

[59] Stack Overflow. 2019. Most Loved, Dreaded, and Wanted Languages. https://insights.stackoverflow.com/survey/2019#technology-_-most-loved-dreaded-and-wanted-languages Last accessed 13 January 2020.

[60] Gang Peng, Yun Wan, and Peter Woodlock. 2013. Network ties and the success of open source software development. The Journal of Strategic Information Systems 22, 4 (Dec. 2013), 269–281.

[61] Robert Phaal and Gerrit Muller. 2009. An architectural framework for roadmapping: Towards visual strategy. Technol. Forecast. Soc. Change 76, 1 (Jan. 2009), 39–49.

[62] Gustavo Pinto, Luiz Felipe Dias, and Igor Steinmacher. 2018. Who Gets a Patch Accepted First?: Comparing the Contributions of Employees and Volunteers. In Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering (Gothenburg, Sweden) (CHASE '18). ACM, New York, NY, USA, 110–113.

[63] Denise F Polit and Cheryl Tatano Beck. 2010. Generalization in quantitative and qualitative research: Myths and strategies. International journal of nursing studies 47, 11 (2010), 1451–1458.

[64] Germán Poo-Caamaño, Eric Knauss, Leif Singer, and Daniel M German. 2017. Herding cats in a FOSS ecosystem: a tale of communication and coordination for release management. Journal of Internet Services and Applications 8, 1 (2017).

[65] Germán Poo-Caamaño, Leif Singer, Eric Knauss, and Daniel M German. 2016. Herding cats: A case study of release management in an open collaboration ecosystem. IFIP Adv. Inf. Commun. Technol. 472 (2016), 147–162.

[66] Huilian Sophie Qiu, Alexander Nolte, Anita Brown, Alexander Serebrenik, and Bogdan Vasilescu. 2019. Going Farther Together: The Impact of Social Capital on Sustained Participation in Open Source.

[67] Hector Ramos. 2018. Open Source Roadmap. https://facebook.github.io/react-native/blog/2018/11/01/oss-roadmap Last accessed 13 January 2020.

[68] David Ribes and Thomas A Finholt. 2009. The long now of infrastructure: Articulating tensions in development. Journal of the Association for Information Systems (JAIS) (2009).

[69] Rust. 2019. Governance. https://www.rust-lang.org/governance Last accessed 13 January 2020.

[70] Rust. 2019. Production users. https://www.rust-lang.org/production/users Last accessed 13 January 2020.

[71] Read Rust. 2018. Rust 2018: Hopes and dreams for Rust in 2018. https://readrust.net/rust-2018 Last accessed 13 January 2020.

[72] Read Rust. 2019. Rust 2019: Ideas from the community for Rust in 2019, and the next edition. https://readrust.net/rust-2019 Last accessed 13 January 2020.

[73] W Scacchi. 2002. Understanding the requirements for developing open source software systems. IEE Proceedings - Software 149, 1 (Feb. 2002), 24–39.

[74] Sonali K Shah. 2006. Motivation, Governance, and the Viability of Hybrid Forms in Open Source Software Development. Manage. Sci. 52, 7 (July 2006), 1000–1014.

[75] Maha Shaikh and Ola Henfridsson. 2017. Governing open source software through coordination processes. Information and Organization 27, 2 (2017), 116–135.

[76] Cuihua Shen and Peter Monge. 2011. Who connects with whom? A social network analysis of an online open source software community. First Monday 16, 6 (June 2011).

[77] Param Vir Singh, Yong Tan, and Vijay Mookerjee. 2011. Network Effects: The Influence of Structural Capital on Open Source Project Success. MIS Quarterly 35, 4 (2011), 813–829.

[78] Matthias Stürmer. 2013. Four types of open source communities. https://opensource.com/business/13/6/four-types-organizational-structures-within-open-source-communities. Accessed: 2020-1-5.

[79] Tanja Suomalainen, Outi Salo, Pekka Abrahamsson, and Jouni Similä. 2011. Software product roadmapping in a volatile business environment. Journal of Systems and Software 84, 6, 958–975.

[80] Yong Tan, Vijay Mookerjee, and Param Singh. 2007. Social capital, structural holes and team composition: Collaborative networks of the open source software community. Proc. International Conference on Information Systems (2007), 155.

[81] Antony Tang, Taco de Boer, and Hans van Vliet. 2011. Building roadmaps: a knowledge sharing perspective. In Proc. International Workshop on SHAring and Reusing Architectural Knowledge. 13–20.

[82] Niels C Taubert. 2008. Balancing requirements of decision and action: Decision-making and implementation in free/open source software projects. Science, Technology & Innovation Studies 4, 1 (2008), 69–88.

[83] Jonathan Taylor. 2017. Rust 2017 Survey Results. https://blog.rust-lang.org/2017/09/05/Rust-2017-Survey-Results.html Last accessed 13 January 2020.

[84] Libra Engineering Team. 2019. Libra Core Roadmap #2. https://developers.libra.org/blog/2019/12/17/libra-core-roadmap-2 Last accessed 13 January 2020.
[85] Scala Team. 2017. Scala 2.13 Roadmap. https://www.scala-lang.org/news/roadmap-2.13.html Last accessed 13 January 2020.
[86] The Rust Core Team. 2018. A call for Rust 2019 Roadmap blog posts. https://blog.rust-lang.org/2018/12/06/call-for-rust-2019-roadmap-blogposts.html Last accessed 13 January 2020.
[87] The Rust Core Team. 2018. New Year's Rust: A Call for Community Blogposts. https://blog.rust-lang.org/2018/01/03/new-years-rust-a-call-for-community-blogposts.html Last accessed 13 January 2020.
[88] The Rust Core Team. 2018. Rust's 2018 roadmap. https://blog.rust-lang.org/2018/03/12/roadmap.html Last accessed 13 January 2020.
[89] The Rust Core Team. 2019. Rust's 2019 Roadmap. https://blog.rust-lang.org/2019/04/23/roadmap.html Last accessed 13 January 2020.
[90] The Rust Survey Team. 2018. Rust Survey 2018 Results. https://blog.rust-lang.org/2018/11/27/Rust-survey-2018.html Last accessed 13 January 2020.
[91] Jonathan Turner. 2016. 2016 Rust Commercial User Survey Results. https://internals.rust-lang.org/t/2016-rust-commercial-user-survey-results/4317 Last accessed 13 January 2020.
[92] Jonathan Turner. 2016. State of Rust Survey 2016. https://blog.rust-lang.org/2016/06/30/State-of-Rust-Survey-2016.html Last accessed 13 January 2020.
[93] Aaron Turon. 2016. Refining Rust's RFCs. http://aturon.github.io/blog/2016/07/05/rfc-refinement/ Last accessed 13 January 2020.
[94] Aaron Turon. 2017. Rust's 2017 Roadmap. https://blog.rust-lang.org/2017/02/06/roadmap.html Last accessed 13 January 2020.
[95] Tuukka Turunen. 2018. QT Roadmap for 2018. https://www.qt.io/blog/2018/02/22/qt-roadmap-2018 Last accessed 13 January 2020.
[96] I van de Weerd, S Brinkkemper, R Nieuwenhuis, J Versendaal, and L Bijlsma. 2006. Towards a Reference Framework for Software Product Management. In International Requirements Engineering Conference (RE'06). 319–322.
[97] Konstantin Vishnevskiy, Oleg Karasev, and Dirk Meissner. 2015. Integrated roadmaps and corporate foresight as tools of innovation management: The case of Russian companies. Technol. Forecast. Soc. Change 90 (Jan. 2015), 433–443.
[98] Georg Von Krogh, Stefan Haefliger, Sebastian Spaeth, and Martin W Wallin. 2012. Carrots and rainbows: Motivation and social practice in open source software development. MIS Quarterly (2012), 649–676.
[99] Kangning Wei, Kevin Crowston, U Yeliz Eseryel, and Robert Heckman. 2017. Roles and politeness behavior in community-based free/libre open source software development. Information & Management 54, 5 (July 2017), 573–582.
[100] Joel West and Scott Gallagher. 2006. Challenges of open innovation: the paradox of firm investment in open-source software. R&D Management 36, 3 (2006), 319–331.
[101] Joel West and Siobhán O'Mahony. 2008. The Role of Participation Architecture in Growing Sponsored Open Source Communities. Industry and Innovation 15, 2 (April 2008), 145–168.
[102] Chorng-Guang Wu, James H Gerlach, and Clifford E Young. 2007. An empirical analysis of open source software developers' motivations and continuance intentions. Information & Management 44, 3 (2007), 253–262.
[103] Xuan Xiao, Aron Lindberg, Sean Hansen, and Kalle Lyytinen. 2018. " Computing" Requirements for Open Source Software: A Distributed Cognitive Approach. Journal of the Association for Information Systems 19, 12 (2018), 1217–1252.
[104] J Xie, M Zhou, and A Mockus. 2013. Impact of Triage: A Study of Mozilla and Gnome. In International Symposium on Empirical Software Engineering and Measurement. IEEE, 247–250.
[105] Yunwen Ye and Kouichi Kishida. 2003. Toward an Understanding of the Motivation Open Source Software Developers. In Proc. International Conference on Software Engineering (Portland, Oregon) (ICSE '03). IEEE Computer Society, Washington, DC, USA, 419–429.
[106] Robert K Yin. 2017. Case study research and applications: Design and methods. Sage publications.

A EMAIL INTERVIEW QUESTIONS

- Q1. How much do Rust roadmaps influence your decision about what work you contribute to the Rust project?
  `No influence at all 1 2 3 4 5 A lot of influence`
- Explain (optional)
- Q2. In your opinion, how helpful are roadmaps for the Rust community?
  `Not at all helpful 1 2 3 4 5 Very helpful`
- Can you explain in what way they are helpful or unhelpful? (optional)
- Q3. How much do Rust roadmaps (e.g. for working groups or projects) match your own priorities for Rust?
  `Do not at all represent my priorities 1 2 3 4 5 Represent my priorities very well`
- Explain (optional)
- Q4. How could the use of roadmaps in Rust be improved in the future?
- Q5. How many years have you been involved with Rust?
- Q6. Have you been on any official Rust team or working group?
  `Yes No`

B ROADMAP TOPIC HEURISTICS

We began by manually extracting a list of topics from the 2018 roadmap. To assign topics to particular issues, PRs, and RFCs, we used the following method:

- Two researchers independently compiled a list of topics from this document, identifying bullet points or lists in the text that appeared to identify specific features. One researcher's list was strictly longer (36 items) than the other's (23 items), so the two discussed each of the additional topics and included all but two of them, resulting in 34 topics.
- Using the generated list, one researcher generated a list of proposed search keywords for each topic, using acronyms, distinctive terms, or word sequences found in that part of the roadmap, that the researcher judged would have high selectivity for distinguishing text about that topic from general Rust discussion. The final list is shown in Table B
- Labels (short strings used by GitHub to tag issues, RFCs, and pull requests) were assigned to roadmap topics by applying the keywords to the labels' descriptions as shown here: https://github.com/rust-lang/rust/labels; for example the label `A-net` was assigned to topic "network services" because it matched the search term "networking". Both researchers checked through this list of labels and their descriptions, and agreed that they matched the topics.
- This mapping was used to assign topics to all issues, PRs, and RFCs in rust (excluding so-called "Rollup" PRs). An issue, PR, or RFC was assigned to a topic if it was tagged with a label that mapped to that keyword.
- Topics were also assigned to RFCs, and tracking issues (a subset of issues formally tied to certain RFCs) if the search terms matched the item's title.
- We then spread activation from RFCs to issues, issues to PRs and RFCs, and PRs to issues: that is, an issue inherits the topic of an RFC if the RFC lists the issue as an official tracking issue. A PR inherits the topic of an issue if the PR mentions the issue ID in its initial description. This was not done recursively.
- We assign a commit to a topic if it was part of a non-Rollup PR of that topic that was eventually merged into the main thread. We omitted commits with multiple parents (to avoid double counting merges of commits) and commits of more than 100 files (to avoid commits that were mass moves of files).

- ”Discussion effort” was operationalized as characters of text in the header and commentthread of each RFC discussion, issue, or PR, excluding code embedded in those comments(which is delimited by triple backticks).
- ”Coding effort” was operationalized as lines of code deleted plus lines of code added.
- ”Team contributors” were operationalized as anyone who was a member of the one of the teams listed on Rust’s governance page at the beginning of 2018.

Also note that some development happened outside these repositories; for example there is a rust-lang/cargo repository; we only capture aspects of development that affect the main compiler project.

Table 6. Search terms for identifying 2018 roadmap topics in labels and text. The left and middle columns are used as search terms within the descriptions of labels; the right column shows the labels that matched.

2018 Topic	**Search Terms**	**Labels**
add edition flag to rustfix	(edition AND rustfix) OR (2018 AND lint* AND rustfix)	
async/await	(async AND await) OR (async/await)	`A-async-await`, `AsyncAwait-Triaged`, `AsyncAwait-Focus`, `AsyncAwait-OnDeck`, `F-async_await`
build system integration		
cargo custom registries	(Cargo AND registry) OR (Cargo AND registries)	`A-registry`
Cargo/Xargo integration	cargo AND xargo	
CLI apps	(CLI app*) OR (CLI application*) OR (command AND line AND app*) OR (command AND line AND application*)	
Clippy	(Clippy AND rustup) OR (Clippy AND 1.0) OR (Clippy AND 1 AND 0)	`A-lint`
compiler optimizations	(optimization*) OR (optimisation*) OR (optimize) OR (optimise)	`A-optimization`, `A-LLVM`, `A-mir-opt`
compiler parallelization	(parallelization) OR (parallelisation)	
compiler-driven code completion for RLS	(auto-complete AND RLS) OR (completion AND RLS)	
const generics		`A-const-generics`, `F-const_generics`
custom allocator	custom AND allocator*	`A-allocators`
custom test frameworks	custom AND test AND framework*	`F-custom_test_frameworks`
embedded device	embedded	`WG-embedded`
GATs	(generic AND associated AND type*) OR (associated AND type AND constructor*)	`F-generic_associated_types`
generator		`A-generators`, `F-generators`

Table 6. (continued)

2018 Topic	Search Terms	Labels
improve compiler error message	error* AND message*	`A-diagnostics`, `F-on_unimplemented`
incremental compilation	incremental AND complilation	`A-incremental`, `A-incr-comp`, `WG-compiler-incr`
internationalization	(internationalization) OR (internationalisation)	
macros 2.0 hygiene	(macro* AND hygiene) OR (macro* AND 2.0) OR (macro* AND 2 AND 0) OR (hygiene)	`A-hygiene`, `A-macros-2.0`
MIR-only rlibs	MIR AND rlib*	
modules revamp	modules	`A-modules`
network services	networking	`A-net`
non-lexical lifetimes	(NLL) OR (non AND lexical AND lifetime*) OR (non-lexical AND lifetime*)	`A-NLL`, `NLL-complete`, `NLL-diagnostics`, `NLL-fixed-by-NLL`, `NLL-performant`, `NLL-polonius`, `NLL-reference`, `NLL-sound`
public dependencies in cargo	(cargo AND libstd) OR (cargo AND std) OR (cargo AND xargo)	
revise cargo profiles	cargo AND profile*	`A-profile`
RLS 1.0	RLS	`A-language-server`, `A-rls`
rustdoc RLS-based edition	RLS AND rustdoc	
rustfmt	rustfmt	
Ship or drop ergonomics RFCs	(ergonomics AND rfc) OR (ergonomics AND initiative)	`Ergonomics Initiative`
SIMD		`A-simd`, `F-simd_ffi`
stabilize impl Trait	impl Trait	`A-impl-trait`, `F-impl_trait_in_bindings`, `F-type_alias_impl_trait`
tokio		
web assembly	(webassembly) OR (wasm) OR (web assembly)	`O-wasm`

Received June 2020; revised October 2020; accepted December 2020