Motivations and Practices of Companies Engaging in Open Source

A Case Study at Microsoft

Arun Kalyanasundaram[1], Judith Bishop[2], James D. Herbsleb[1]

[1] Institute for Software Research
Carnegie Mellon University,
Pittsburgh, PA, USA
{arunkaly, jdh}@cs.cmu.edu

[2] Department of Computer Science
University of Stellenbosch
Stellenbosch, South Africa
judithbishop@outlook.com

***Abstract*—A trend is emerging for companies of all sizes to open source their software. Large corporations like Microsoft have quickly established a powerful presence in the open source world. In order to gain a deeper understanding of how companies engage in open source, we conduct a multiple case study of 24 software projects at Microsoft with the following goals: (1) to understand the different motivations of industry open source projects, (2) to identify key practices for improving community engagement, and (3) to analyze techniques for evaluating community engagement. The results of our study show that convenience to collaborate, growing a terminated project, attracting contributors and a sense of obligation to the community are the primary motivations to open source. The key practices are more varied, and range from managing transparency and providing clear guidelines to maintaining multiple channels of communication and incentivizing contributors. Although evaluating an open-source project in general is complex, we show that the level of community engagement can be reliably measured by using together the measures of external contributions (code commits, bug reports and pull-requests) from GitHub. Our contribution to the literature comes from showing how these results are more important in an industrial setting, where a community must often be built from the ground up. With projects in our study ranging across different technologies and with applications in several domains, our findings can assist practitioners in building, managing and fostering open source communities in general.**

Keywords:- open source software; open source in industry; motivations; practices; community engagement; qualitative analysis; case study; Microsoft; GitHub;

I. INTRODUCTION

As of 2016, there are more than 331,000 organizations with employees contributing to open source projects on GitHub [1]. While industrial contribution to open source software is not a new phenomenon [2], the recent increase in companies opening up their proprietary code is significant [3]. This paper looks at the reasons why companies are open sourcing code, and how they can best be successful in this endeavor. What to measure and how to improve against these measurements are two of the key questions we address.

Companies have a broader range of motivations for open sourcing than a typical open source project. These include making a technology as a *de facto* standard [4], enabling compatibility [5], increasing demand for complementary services [6], signaling technical excellence [7], and raising a positive perception among the academic and student community [8].

However, many companies may have unrealistic expectations about the benefits of open source, and may not always be able to meet their objectives [9]. We present a categorization of objectives from an industrial study, which will enable researchers and practitioners to better understand and appreciate how to interact with open source industrial projects.

Many open source projects monitor their activities and try to measure their success in some way [10]. For industrial projects, doing so is even more important, since the mode of operation is still new and can be questioned by management. There is a wide range of metrics for measuring success [11]. The second contribution of the paper is to reveal from actual experience what industrial projects are actually doing to monitor their community, and the mechanisms they use to further these.

We use a multiple case study approach [12] to study a set of projects that were open sourced at Microsoft Corporation. We chose Microsoft for two reasons. First, Microsoft had a long history of closed source software development [13], but over the last few years has made available the source code of many of its key projects [14], and now encourages its employees to actively contribute to existing open source projects [15]. It can therefore be regarded as a newcomer, with fresh ideas. Second, and most importantly, Microsoft is the biggest corporate contributor to open source software on GitHub [1].

In 2016, there was an initiative at Microsoft to gather together over fifty disparate open source projects with the aim to making them more attractive and available to developers, and to academics in particular[1]. All of the owners of the projects faced challenges in ramping up their communities and were starting to evaluate their practices. In our study, we interviewed 22 core team members from 24 of these open source projects. Our interviewees were all employees of Microsoft and were either one of the founders of the project or the current maintainers of the project. The study itself was conducted over a two-month period during June and July of 2016 when two of the authors were physically present at Microsoft's corporate headquarters in Redmond, WA.

[1] http://aka.ms/open-source

The aim of the study was to learn, categorize and reveal motivations and key practices that could then be followed by others embarking on open sourcing in industry. The projects in our study covered a wide range in type and age. The key value of our work was to distill the data points they provided down into four main motivations and six key practices. While the motivations were seen to be angled towards industrial needs, the practices are universally applicable.

II. Research Questions

Releasing proprietary code as open source is not just about making the code publicly available but also involves fostering and managing a community [2]. Therefore, the key concern for a company is almost always the external community, which are users and developers who engage with the project but are not affiliated with the company. There are several activities that an external community can engage in with the project, which could range from just using the project to reporting bugs to making code contributions. We broadly refer to these activities as *community engagement*, and we particularly analyze the different practices to improve community engagement, and the various techniques used to evaluate the level of community engagement of a project. The goals of our paper can be broken down into the following three research questions:

RQ1: How and why do projects in a company differ in their motivations to open source?

RQ2: What are the key practices that projects open sourced by companies adopt, in order to improve their community engagement?

RQ3: What are the different techniques that projects open sourced by companies use, in order to evaluate their level of community engagement?

III. Background

Previous studies have broadly classified the process of open sourcing proprietary code into three different phases [16]. The first phase involves assessing the release readiness of a project, in other words whether a project is ready to be open sourced or not. However, depending on their motivations to open source, projects can be at different levels of release readiness when they decide to open source.

The motivations of a company to participate in open source can be classified into three broad categories, namely economic, technological and social [17]. Economic and technological motivations are often evidenced by acquiring indirect revenues from complementary services [6], and receiving feedback from the community to improve software quality [17]. The social motivations, however, are more subjective and is often associated with improving the perception of the company in the larger open source community [18]. However, motivations across projects even within a company could differ, and our research aims to understand how their motivations differ and why.

The second phase involves performing certain activities to help the project transition from proprietary to an open source environment [16]. This could involve activities such as refactoring the code, removing company specific comments and making the architecture more modular to facilitate participation [19]. A key part of this phase involves building, managing and fostering a community. Although general open source practices to build communities are also applicable in an industry setting, there are unique challenges that need to be understood and addressed. Therefore, in this paper we identify the key practices that projects in a company adopt in order to improve their community engagement and analyze its importance in an industry context.

The third phase involves evaluating the state of an open source community [16]. This often involves using the measures of success and health of open source projects, except in an industry context the emphasis is on the participation of the external community. Since companies often have different objectives, and this combined with the diversity of software products makes it difficult to evaluate an open source community using a consistent approach [5]. Therefore, in this paper we analyze the various techniques that projects in our sample use to evaluate their community engagement, and we use data from GitHub to propose reliable measures.

IV. Methodology

We conducted 22 semi-structured interviews, with core members of 24 open source projects at Microsoft. The project names along with their source code repositories are presented in TABLE I. The projects were chosen so as to cover a mix from different technologies, team size and age. The projects ranged from compilers to large scale distributed systems, from theorem provers to visualization toolkits, with applications including machine learning, security, education and software engineering.

Once we identified the projects we reached out to employees who were either the founders or the current maintainers of the projects. We scheduled in-person meetings with one person from each project, except four projects, which involved interviewing over Skype. These interviews were scheduled for 30 minutes each but some of them took up to 60 minutes. The goals of our interviews were to understand: (1) why and how the project became open source, (2) how did open sourcing impact the team's development process, (3) what were specific examples of their interactions with the external community, and (4) what types of contributions they received. After the first two interviews, we revised our protocol to also ask about their approaches for evaluating their community engagement, any specific challenges in managing the community, and the practices they adopt to improve community engagement.

The interviews were conducted by two of the authors, with one interviewer primarily involved in asking questions and the other involved in taking notes and asking additional questions. These notes captured the gist of the answer to each question along with pointers to artifacts (GitHub repository / issues, documents, code snippets, etc.) were noted down and in some cases their pictures were taken. Occasionally interesting quotes of interviewees were captured as they were spoken. After each interview, the notes from both interviewers were combined, formatted and grouped under the different categories of questions

mentioned above. This helped us quickly glance at the responses of all interviewees for a particular topic (or a set of questions). We intentionally did not audio record the interview because: (1) we solicited their participation for an informal conversation at their offices, and (2) the interview involved participants walking us through their tools, websites, dashboards and other relevant artifacts, which cannot easily be captured in an audio recording.

We qualitatively analyzed the interview data using a two-phase approach. The first phase involved open coding [20] where we assigned codes to parts of our notes (often an answer to a question or in some cases a couple of sentences) identifying different themes under each of our topics. In the next phase, we identified and merged emerging codes into specific categories, which helped find supporting evidence to answer our research questions.

We used activity data from the corresponding GitHub repositories of projects to: (1) compare the level of community engagement across different projects, and (2) assess the reliability of our measures. Three projects (*Codalab, F#, .NetCore*) in our sample had more than one GitHub repository associated with it. We found that the overall activity profile across different repositories of a project was similar. So we chose the one that had the most recent activity.

TABLE I. LIST OF PROJECTS IN OUR STUDY

Project Name	GitHub Repository	Year Open Sourced
.NetCore	dotnet/coreclr	2015
ChakraCore	Microsoft/ChakraCore	2016
CNTK	Microsoft/CNTK	2015
Codalab	codalab/codalab-competitions	2013
CodeContracts	Microsoft/CodeContracts	2015
Dafny	Microsoft/dafny	2012
F*	FStarLang/FStar	2014
F#	fsharp/fsharp	2005
IDD	predictionmachines/InteractiveDataDisplay	2015
Ironclad	Microsoft/Ironclad	2014
LEAN	Leanprover/lean	2013
Madoko	Codeplex Repo[2]	2014
MSAGL	Microsoft/automatic-graph-layout	2016
NETMF	NETMF/netmf-interpreter	2009
Orleans	dotnet/orleans	2014
PXT	Microsoft/pxt	2015
Room Alive	Kinect/RoomAliveToolkit	2015
Roslyn	dotnet/roslyn	2014
SEAL	Codeplex repo[3]	2015
Tabular	TabularLang/CoreTabular	2015
Touch Develop	Microsoft/TouchDevelop	2015
TPM	Microsoft/TSS.MSR	2013
TypeScript	Microsoft/TypeScript	2014
Z3	Z3prover/z3	2015

V. RESULTS - RQ1: HOW AND WHY DO PROJECTS DIFFER IN THEIR MOTIVATIONS TO OPEN SOURCE?

To our surprise, all our interviewees expressed little to no concern about losing Intellectual Property rights (IPRs) from making their code open-source. Even if there were risks, prior research has shown that the potential benefits of open sourcing could often outweigh their risks [16]. Although all projects had more or less similar kinds of expectations of potential benefits, the extent of these benefits and their priorities were determined by their motivations to open source. In this section, we present the different motivations to open source, which could help us better understand the tradeoffs considered when open sourcing proprietary code. We found that the primary motivations of projects in our sample can be grouped into one of four categories as discussed below. In TABLE II. we summarize our findings and identify the projects under each category.

A. Convenience

There are both technical and legal challenges when it comes to sharing proprietary code developed within a company with partners and collaborators outside the company. We found that open sourcing the code provides a convenient mechanism for sharing code and collaborating with potential partners. In most cases the need arose from interns wanting to coninue working on the projects after finishing their internship. In other cases, the projects wanted to benefit from collaborations with research labs in academia.

Open sourcing not only took care of legal issues that could arise from sharing and accepting contributions but also enabled the company to establish a uniform technical infrastructure so that anyone could build, develop and test the code without having to use any company specific resources. In one of the projects (*Dafny*), our interviewee mentioned that the code was first open sourced to enable students to contribute after their internship, however, it also helped form new collaborations with faculty in academia. Overall, however, the primary motivation to open source arose from their need to make it more convenient to share and collaborate with others outside the company.

B. Survival

A project in a company exists as long as it has enough resources allocated and employees assigned to work on it. It is quite common for companies to terminate projects for various reasons. In certain cases such as existing customers wanting to use and maintain the project, the company allows the code to be made available as open source. As prior research shows [19, 21], the process of transitioning the code to open source can be quite complicated, and therefore, the role of developers involved in this process is important. We found that employees who were working on the project were motivated because they wanted to see the impact of their work and the possibility of the project becoming entirely community driven.

To our surprise, our interviewees from three different projects (*Orleans, Tabular, NetMF*) used the exact same words to describe the motivation of their project to go open source, which were *"to prevent the project form dying"*. Among them, one project – *Orleans* – went on to build a very successful community, and we will discuss some of their practices in the subsequent sections. Another project –

[2] http://madoko.codeplex.com/
[3] http://sealcrypto.codeplex.com/

Tabular – did not receive as much community participation. One of the reasons we found was that the project (*Tabular*), even after open sourcing continued to use proprietary third party dependencies. Overall, the primary motivation to open source arose from their need to keep the project alive, the outcome, however, depends on its technical architecture and the team's community management practices.

C. Attraction

When asked about their primary motivation to open source, one of our interviewees *(ChakraCore)* mentioned – *"to stay relevant in the minds of the developers"*, in other words, to remain attractive to software developers. Making their code open source was one way that they could attract talented software developers to work on their project. Since new technologies are now being developed at a much faster rate than ever before, companies constantly need an influx of developers working on their projects. We found that open sourcing in this context helped in two ways: (1) it provided an opportunity for the company to adopt a more aggressive development model. As one of our interviewees (*Roslyn*) mentioned – *"every commit is a release"*, which results in faster turnarounds, and (2) it allowed developers to tinker with the code in ways that made the code more robust and eventually improved its quality.

In some cases, there was a need to attract a specific set of individuals or a particular community of developers to the project. As one of our interviewees (*Codalab*) mentioned – *"we wanted to bring onboard rockstars in the field"*. This is because in certain cases, a few individuals can be quite influential in inspiring and building a community. In other cases there could alredy be a community for a particular technology (*PXT*) and open sourcing can help the community gain confidence in adopting the project. Overall, the primary motivation to open source arose from the need to attract one or more individuals to the project.

D. Obligation

In certain technology domains such as cryptography, security and compilers, it is often customary to make the source code available along with the binaries. For example, users have more confidence in using encryption software that has the source code available. Our interviewee (*TPM*) mentioned that – *"the openness brings some sort of assurance, and they trust you"*. There is however, a risk that the source code could be used to reverse engineer some of the security features in the software. In fact, our interviewee (*TPM*) mentioned that the management raised concerns after a few developers were able to uncover two zero day vulnerabilities, which was made possible by the availablity of the source code. However, this is a trade-off that companies have to make when open sourcing their code.

Another interviewee (*Madoko*) mentioned that open sourcing helped increase adoption of his software tool even though his community were all end users who may have very little expertise to contribute source code - *"Although the intended audience is end users, people would not even use it if it was closed source. It gives them confidence in using it"*. This was because open sourcing gave users the confidence that the software will not be commercilized in future.

Another form of obligation is when the users explicitly request that the software project be made open source. One of our interviewee (*MSAGL*) mentioned that his project received several requests to access its source code at a time when it was being commercially licensed. Although the requests mostly came from academics, they decided to make the source code available for all (not just the users who requested for it) by releasing it under an open source license. This is another trade-off that companies have to make – balancing goodwill among users with monetizing its products. Overall, the primary motivation arose from the need to fulfil certain obligations to its community.

TABLE II. PRIMARY MOTIVATIONS TO OPEN SOURCE

Primary Motivation	List of Projects	Brief Description
Convenience	Dafny, F*, Ironclad, LEAN, Z3	Help collaborate with external partners.
Survival	CodeContracts, Orleans, Tabular, NETMF	Build an external community after project has been terminated internally.
Attraction	CNTK, PXT, Codalab, ChakraCore, Roslyn, TypeScript	Bring on-board external developers to participate in projects.
Obligation	F#, Madoko, MSAGL, Touch Develop, SEAL, TPM, .NetCore	Honor the expectations of the community, the domain / technology, and requests from users.

E. Reasons for Differences in Motivations

Our results show that projects even within a company could significantly differ in their motivations to open source. It is important to understand what causes these differences. Although almost all projects desired to build a strong and sustainable community around their software, their primary motivations to make their code open source differed. We discuss below three factors that can explain most of the differences in their motivations.

First, the division in which a project is present influences its motivations to open source. In our sample, projects either belonged to the product division or the research division of the company. The two divisions differed in a number of ways, however, the key differences were that the research division had a slower release cycle with more emphasis on exploration of new technologies. Therefore, projects in the research division open source their code to foster new collaborations with academic labs whereas projects in the product division open source to attract new developers who can make code contributions.

Second, the underlying technology of a project could influence its motivations to open source. The projects in our sample belong to a mix of different programming languages, technologies, application domain, and the type of user interface. As we discussed earlier, certain types of application domains such as cryptography and compilers, require that they be open sourced in order to gain acceptance within a community. In other cases, certain forms of technologies might already have a community of users, and

open sourcing could be a way of making inroads into the community. For example, a technology like the Arduino microcontroller [22] already has a thriving community of users and developers[4], and we found that the *PXT* project in our sample became open source to tap into the potential of an already existing community.

Third, the maturity level of a project could influence its motivations to open source. A project's stage of development such as planning, alpha, beta, production, etc. is a good proxy for a project's level of maturity [23]. We found that there are two extremes of maturity level of a project. In one extreme, we have projects like *LEAN* and *Codalab* in our sample that open sourced right at the inception of the project (less mature), and the other extreme are projects like *Z3* that open sourced after several years and a fully finished product (more mature). Our interviewee (*LEAN*) mentioned that open sourcing early could help attract more developers if the project is likely to be large and complex. On the other hand, making a mature project open source can build a stable community of users who may not be able to contribute code but can contribute by reporting bugs and answering other's questions (*Z3*).

VI. Results – RQ2: Key Practices To Improve Community Engagement

Even though the primary motivations of the projects in our sample differed considerably, we found that they all had to engage the external community to achieve their objectives. By external community, we refer to anyone who is not an employee of the company. Community engagement is a set of activities that the external community engages in with the project, and these activities could range from just using the project to reporting bugs to making code contributions. We asked our interviewees about the practices they adopt to improve their community engagement and grouped these practices into six categories, which we discuss below.

A. Incentivize Contributors

Although contributors of open source projects often have their own motivations such as learning, fun, and reputation [17]. Our interviewees mentioned that keeping the external community motivated to make contributions was a difficult challenge. Our interviewees addressed this challenge by finding ways to incentivize existing contributors. The *Roslyn* project in their first year of open sourcing shipped out custom made mugs to contributors. Each mug was personalized with the contributor's GitHub username and a unique identifier (SHA) of their commit, along with a logo of the project. Our interviewee (*Roslyn*) mentioned that this helped motivate some contributors to make recurrent contributions, and as previous studies have shown, continued contributions are more important for open source projects than one-time contributions [24].

Incentives like these, however, require more financial resources and therefore, a few other projects adopted a different approach to incentivize contributors. One project (*Typescript*), publicly acknowledged each contributor by honoring him or her in the project's release notes. In another project (*.NetCore*), names of contributors were mentioned in blog posts announcing a particular release. These practices could enhance the reputation of contributors within the open source community, there by serving as an effective incentive to motivate contributors.

Career development is a commonly known motivation for developers of open source projects. This is because open source contributions can help showcase a developer's skills to potential employers. We found that this could also be an incentive for industry open source projects especially because there is also a possibility of finding employment in the same company that open sourced the project (*Orleans*).

Although projects do not openly publicize potential employment through open source contributions, doing so might serve as an indirect incentive for contributors. Overall, we see that projects can choose to incentivize contributors in many different ways, but the key takeaway is that some of them might keep contributors motivated in the long term, while others only in the short term.

B. Manage Transparency

Our interviewees (*Roslyn*, *Typescript*) mentioned that contributors not only want to make contributions, but they also want to be part of the development process and be involved in the design decisions that are made. Previous studies have also shown that the perceived credibility and openness of a company could provide a strong intrinsic motivation for contributors [25].

However, we found that being completely transparent can be challenging for open source industrial projects. One of the problems our interviewee (*.NetCore*) mentioned was that discussing the design of new features in public can set false expectations with customers who use a proprietary version of the software. They might expect that these features will be delivered even though some of them might just be preliminary ideas or prototypes. There could be other reasons for not being transparent, such as building new features in a private branch or delaying certain features to prevent competitors from having an advantage. Therefore, managing transparency effectively is an important practice for open source industrial projects.

In order to keep the external community informed of the project's progress, one of the projects (*Roslyn*) regularly posted minutes of their internal meetings on GitHub. When for a couple of weeks they failed to post the minutes, one of the contributors opened an issue[5] on GitHub inquiring about it. That particular issue has had about ninety-one comments, and the contents of some of these comments clearly indicate that the external community not only want to use the software and contribute to it, but also want to be part of every step of the development process.

This level of transparency, which involves sharing notes of internal discussions, might not be appropriate for many industry open source projects. Lack of transparency could however, lead to distrust among external community

[4] https://www.hackster.io/arduino

[5] https://github.com/dotnet/roslyn/issues/12630

members in the project. Therefore, projects have to strike the right balance when managing transparency.

One of the projects (*Orleans*) had developed several major features in a private branch before making it available to everyone. They however, informed the external community what features they were working on and when they could be expected. Even though the development was not entirely transparent, there were no issues of mistrust among the members of the external community because they remained informed throughout the process.

We found that sharing project roadmaps was another practice of managing transparency that quite a few projects (*ChakraCore, TypeScript*) followed. Not only does project roadmaps help everyone know the future direction of a project, but it also allows the external community to participate in modifying and evolving the roadmaps during the course of the project.

Interestingly, we found that being transparent can also be helpful in setting the right expectations for the project. Our interviewee (*NETMF*) mentioned that when a company open sources a project, the external community expects a certain level of commitment, particularly from its employees involved in the project. However, not all projects might have the resources within the company to manage all the needs of an external community, and therefore, being transparent about it could actually be beneficial. This could also lead to members of the external community stepping up to answer other's queries and resolve open bugs (*NETMF*). Overall we see that there is both an increased amount of scrutiny and expectation from industry open source projects, which can best be handled by using these effective ways of managing transparency. The following quote nicely summarizes this practice:

"Our biggest challenge moving to GitHub was to adopt the GitHub paradigm of development. We try to do everything open. We maintain one branch." (TypeScript)

C. Provide Clear Guidelines

Successful open source projects are known to devise various ways of lowering the barriers of making contributions, particularly for new comers. This is even more important in industry open source projects because a company has access to more computing resources than an individual open source contributor. In most cases, a project developed within a company uses sophisticated infrastructure to compile and build the code. Therefore, the first step is to provide clear guidelines on how to build and deploy the software[6] (*ChakraCore*). In addition, we found that removing or rewriting proprietary dependencies and making certain build systems externally accessible were important activities projects performed before making their code open source.

It is also important to provide clear guidelines on how someone can submit contributions to the project. This is particularly complicated with industry open source projects because often companies expect contributors to sign a copyright license or have a specific process and quality level of accepting contributions. One of the projects (*Roslyn*) gives a very simple and clear guideline stating[7] – *"Before submitting a feature or a substantial code contribution please discuss it with the team and ensure it follows the product roadmap"*. Our interviewee mentioned that this guideline was quite useful, as it saves time and the pain of having to deal with rejected pull-requests for both the contributor and the core team.

Projects realize that contributing code is not the only way the external community can engage with the project. The guidelines of projects – *Z3* and *TypeScript* – clearly enumerate the different ways that a user can participate in the project. These include activities such as reviewing other's code, engaging in discussions on StackOveflow, contributing tests and benchmarks, tweeting about the project, and so on. These are clearly stated on the project's GitHub page along with links to the corresponding type of activity, a practice that we claim can be extremely useful for projects to improve their community engagement.

We found that of all the types of activities, guiding a newcomer to make a code contribution is the most challenging one. The *Roslyn* and *.NetCore* projects in our sample, implemented a practice where they explicitly label an issue on GitHub as "*up-for-grabs*" indicating that it is something that a newcomer might be able to fix. For example, as of March 2017, 565 issues in the *Roslyn* project had the 'up-for-grabs' label, and out of which 101 were closed. Our interviewee mentioned that most users who resolved these issues were people who otherwise would not have made any code contributions, thus this practice can be effectively used to help a newcomer make his/her first contribution, thereby lowering the barrier for community engagement.

D. Be Responsive

We found that smaller projects (with 1 - 3 internal core team members) found it particularly challenging to keep up with the needs of the external community, such as responding to queries, bug reports, pull-requests in a timely manner. Although in traditional open source projects, there is a gradual movement of members from the periphery to core [26], this process however, is much slower in industry open source projects for two reasons: (1) Most projects become open source after being developed as proprietary closed source for several years, and so it is difficult for an external community member to understand the codebase in a relatively short time, (2) The company's policies in certain projects prevents giving privileges such as commit rights to external community members thus creating an impediment for others to take up new responsibilities.

As a result the onus is often on a small internal core team to remain committed to its community. Responding to queries, bug reports and pull-request submissions in a timely manner is a way of telling the external community that the core team values every form of community engagement. Being responsive is important as it could lead to a sense of goodwill for both the project and the company. In addition, it

[6] https://github.com/Microsoft/ChakraCore/wiki/Building-ChakraCore

[7] https://github.com/dotnet/roslyn/wiki/Contributing-Code

could also lead to certain external community members stepping up to handle some of the queries, as in the case of *Z3*, where recently other members have started answering questions on StackOverflow. However, despite lacking the necessary manpower, we found that some of the teams had devised specific strategies to be highly responsive. In *Z3*, the two internal core team members would dedicate a couple of hours every morning exclusively to respond to queries on GitHub. In another project (*CNTK*), they had set up a roster where each core team member would respond to queries on a particular day of the week. In other projects (*Roslyn*, *.NetCore*), there was a greater emphasis on at least giving a first meaningful response to each bug report or pull-request even if they cannot be immediately resolved. Practices like these can be effective in making sure that every query is tended to in a timely manner, thus building goodwill and improving community engagement.

E. Archive Contributions

It may not always be feasible to accept new features submitted by contributors, for reasons such as the feature not aligning with project's goals, coding standards not being followed, or too many dependencies required. This is especially true in industry open source projects where they may have more rigid roadmaps. We found that some projects consolidated such features and made them available for others, even though they were not merged into the main branch (not made available part of any release). For example, the *Orleans* project created an organization (*OrleansContrib*) on Github to host features, extensions and forks of the original project. It has over 40 repositories contributed by the external community, and a link to the organization is available on the project website[8]. Another project (*F#*), lists on its website[9] a set of "community projects" that were developed using its compiler and libraries.

There are three reasons how this form of archiving contributions can be beneficial. First, there could be other users in the community who might find some of these contributions useful and benefit from them. Second, it could help contributors showcase their work to a larger audience (as opposed to having it hosted in their own fork) and could give them a sense of accomplishment. Finally, it also sends a strong signal to potential contributors about the welcoming nature of such projects.

F. Maintain Multiple Channels

While we have discussed all of the above practices in the context of GitHub. However, open source projects usually maintain several communication channels such as mailing lists, IRC (Internet Relay Chat), Twitter and so on. Each communication channel has different purposes and caters to a particular audience (e.g. mailing list for users vs. IRC for developers) or sharing a specific type of information (e.g. announcements on Twitter). We found that maintaining different communication channels was extremely important for certain industry open source projects when they had two vastly different stake holders: paid customers and the external community of users and developers.

One of our interviewees (*.NetCore*) mentioned that they use GitHub and StackOverflow primarily to interact with the open source community of developers and users, whereas they use blogs and Twitter to make announcements of upcoming releases and events. Since all channels are public, our interviewee mentioned that even if an interaction is meant for developers, it is likely that customers might read or participate in it. This could also lead to misunderstandings. If the design of an experimental feature or a prototype is discussed say on GitHub, some customers might not have read the entire discussion and could mistake it as a feature that will be available in a future release. So there are times when the creation of a private channel might be beneficial.

The *.NetCore* project created a semi-private mailing list which anyone could write to but only the internal core team had access to it. The purpose was to let the community share their concerns or issues with the core team in cases when it may not be appropriate to share publicly. In summary, maintaining different channels is crucial for improving community engagement by allowing different stakeholders to participate more effectively.

In TABLE III. we summarize our findings by showing the different practices and their effects on community engagement.

TABLE III. PRACTICES AND THEIR EFFECTS ON COMMUNITY ENGAGEMENT

Practices	**Improves Community Engagement By**
Incentivize Contributors	Maintaining continual contributions.
Manage Transparency	Increasing credibility and trust
Provide Clear Guidelines	Lowering the barrier to contribution
Be Responsive	Building goodwill and reciprocity
Archive Contributions	Acknowledging contributors whose contributions were not integrated
Maintain Multiple Channels	Allowing different stakeholders to participate more effectively

VII. RESULTS – RQ3: EVALUATING COMMUNITY ENGAGEMENT

Our interviewees described different techniques they had adopted to evaluate the level of community engagement of their projects. We grouped those techniques into two broad categories, which we describe below.

A. Users

For many projects in our sample, a key step in evaluating community engagement is to know the number of users of their software project. As literature suggests, the number of users is also one potential indicator of an open source project's success [10].

In some of the projects (*F*, Dafny, LEAN*), users could use the software through an online interface without having to install it on their machine. As a result, the number of users of these projects was readily available (through either user

[8] http://dotnet.github.io/orleans/

[9] http://fsharp.org/community/projects/

logins or unique IP addresses from corresponding server logs).

However, measuring the actual number of users of software installed on user's machines is difficult [10], since use of techniques like Telemetry [27] to gather usage information is usually frowned upon in the open source community. Most projects in our sample estimated the potential number of users through indirect measures such as: (1) download counts (*Madoko, SEAL, RoomAlive*), (2) page views (*CodaLab*) on their own website, (3) citations (*IronClad*) to software or related research papers in scientific publications, and (4) stars (*CNTK*) on GitHub, which is also a good indicator of the popularity of the project [28].

Thus projects use vastly different techniques to measure the same dimension of community engagement. Reliably estimating the number of users of a software project might be better accomplished by a convergent validity strategy [29], combining data from multiple sources [10] such as GitHub, StackOverflow, mailing lists, etc. However, except for a few projects (*.NetCore, TypeScript*) in our sample, other projects relied only on a single source of data. One of the reasons for this could be that, since open sourcing is still a new trend for companies, there is more focus on transitioning their projects and implementing best practices rather than evaluating them.

B. External Contributions

A few projects (*CNTK, .NetCore, Roslyn, TypeScript*) in our sample evaluated community engagement by measuring the following three forms of contributions from the external community: (1) code, (2) bug reports, and (3) discussions. It is evident that submitting code and bug reports are useful forms of community engagement, however, discussions (e.g. commenting on issues) can also be valuable forms of contributions as the following quote suggests:

"The GitHub discussions are our most powerful form of data to understand how people are using it. It is in some sense even more powerful than telemetry because we don't have to dig up in the telemetry logs to identify the problem, the users just post the problem and we can see how they are using it." (TypeScript)

Unlike the number of users, the external contributions can be more readily obtained from GitHub using the following measures: (1) code commits, (2) pull requests (PR), (3) bug reports, (4) PR comments, and (5) bug comments. We could identify a contribution as external if the GitHub username corresponding to the contribution is not a member of any of the GitHub organizations (about 30) belonging to Microsoft[10]. Although Microsoft maintains a list of GitHub usernames of its employees, this list is private and only one of the projects (*.NetCore*) in our sample used this list to measure external contributions.

In order to assess the reliability of these measures in measuring the level of community engagement, we first computed each of the five measures for projects in our sample (i.e. projects with GitHub repositories as shown in TABLE I.). To meaningfully compare these measures across different projects, we computed the proportion of external contributions, which is the ratio of external to total contributions for a given measure. A value greater than 0.5 for a given measure (say, code commits) would indicate that there are more external code commits compared to internal. Fig. 1 shows box plots of each measure, representing the spread of the proportion of external contributions across different projects.

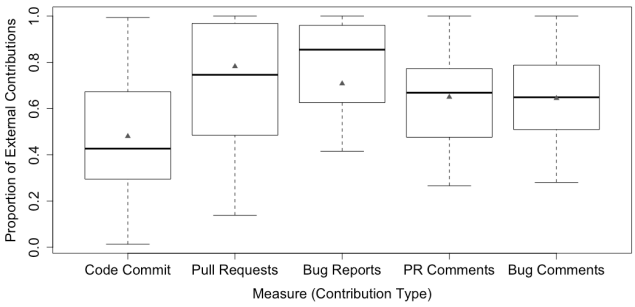

Fig. 1. Box plots showing the spread of the proportion of external contributions across all projects for each measure. (The mean is marked with a solid triangle)

We tested for differences across measures using a Kruskal-Wallis test [30]. It tests the null hypothesis that the medians of all measures are equal. We rejected the null hypothesis ($p < 0.01$), and performed a post hoc analysis using the Dunn's test, which showed two statistically differences: (1) code commits lower than bug reports ($p < 0.01$), and (2) code commits lower than pull-requests ($p < 0.05$). This is expected, and corroborates previous findings [31] that non-core members report a larger proportion of bugs than making code commits. Our finding shows that this is also applicable in an industry context, where the core and non-core is distinguished as members within and outside the company respectively. The differences between other pairs were statistically insignificant with $p > 0.05$.

We then looked for correlations between pairs of measures to determine if they could be grouped together as a scale and thereby assess its reliability. We found that three measures: the proportion of external code commits, bug reports and pull requests had a statistically significant ($p < 0.01$) pairwise Pearson's correlation of greater than *0.62*. These three measures also had a high Cronbach's Alpha (*0.84*) and removing any of the three measures resulted in a drop in alpha (< *0.8*). This indicates that the three measures can be used together to measure community engagement more reliably than any of the individual measures. The bug comments and PR comments measures, however, were only moderately correlated (*0.5*), and we neither found any significant differences between the two nor with other measures. Therefore, they can neither be used together nor individually as a measure of community engagement without analyzing additional data.

VIII. DISCUSSION

The growing trend of companies to open source their proprietary code calls for more active research to gain a

[10] https://opensource.microsoft.com/

deeper understanding of this phenomenon. Through our in-depth study at Microsoft, our work gives greater insights using empirical evidence into the three main phases involved in open sourcing proprietary code [16]. The first phase involves assessing whether a software project is ready to be open sourced. We showed that the decision on when to open source is driven by their motivations, which could differ widely across projects in a company. The second phase involves performing various activities to transition the project to an open source setting, which almost always requires building and managing a community. We identified key practices, and showed how they improved specific aspects of a project's community engagement. The third phase involves evaluating the state of their communities, which is usually very subjective. However, we showed that three measures of proportion of external contributions on GitHub (code commits, bug reports and pull requests) when used together, could reliably measure the level of community engagement.

A. Implications

Although the practices we identified can generally be applicable to any open source project, our contribution to the literature comes from showing why they are more important in an industrial setting, where a community must often be built from the ground up. This could help researchers conduct more focused studies in understanding how these practices might differ across companies, and whether or not the corporate culture [32] plays a role in success.

Our work contributes to the literature of measuring health (or success) of open source projects. Although these are considered subjective and difficult to measure [11], our work shows that there are specific components of health such as community engagement that can be reliably measured in an industry context. Therefore, by understanding the underlying phenomenon that is common to a group of projects, researchers can devise more effective ways of evaluating open source projects.

Finally, our work has value for practitioners, such as individuals and companies interested in releasing open source software and building communities. While we discussed key practices for improving community engagement, it also highlights some of the challenges that practitioners must address when building an open source community. Since practitioners are often interested in monitoring their communities, our work can inform the use of relevant metrics, and assist in the design of corresponding tools.

B. Limitations

Results of case studies are often only valid in the setting in which they were conducted, and our study is no different. However, in order to minimize threats to external validity, we chose to perform our study at Microsoft, which is both representative of a large corporation and is increasingly an active participant in the open source community.

Our findings are based on our interviews with project owners and maintainers within Microsoft. It is possible that their perceptions could be different from those of the external community, especially on how various practices help improve community engagement. This could pose a threat to the internal validity of some of our results, and can be addressed in future studies by also interviewing other developers and users of a project.

We measured the level of community engagement based on the techniques our interviewees were using. For projects in other companies, these techniques may not effectively measure community engagement and could therefore, pose a threat to the construct validity of our measures when used in other settings.

IX. Conclusion

We set out to gain a deeper understanding of the different motivations, the key practices and evaluation of community engagement for industry open source projects. The outcome of our study produced a classification of the primary motivations into four different categories. We identified six key practices and showed its impact on improving community engagement. We argued why these practices are more relevant in an industry setting, and in the process highlighted the major challenges the projects faced. We analyzed the different techniques for evaluating the level of community engagement and assessed the reliability of our measures using data from GitHub.

There is scope for future work, particularly in two main directions. First, replicating the study in other companies can shed some light into various factors that could influence our results. Conducting more focused studies in multiple companies can help us build a theory of the underlying phenomenon behind motivations and practices of industry open source projects, which is often overlooked in software engineering literature. Second, our work can be extended to create a framework for evaluating a company's open source initiatives. The framework could include metrics for evaluating community engagement such as the ones we identified, which could assist developers in building tools that are easy to use and of high practical value to companies.

Acknowledgment

We thank all the researchers and developers at Microsoft who gave of their time so willingly for this study.

References

[1] M. Asay, "Who's no. 1 in open source? Microsoft!," 2016. [Online]. Available: http://www.infoworld.com/article/3121792/open-source-tools/whos-no-1-in-open-source-microsoft.html. [Accessed: 07-Apr-2017].

[2] J. West and S. O'mahony, "The role of participation architecture in growing sponsored open source communities," *Ind. Innov.*, vol. 15, no. 2, pp. 145–168, 2008.

[3] S. J. Vaughan-Nichols, "Why Microsoft is turning into an open-source company," Jun-2016. [Online]. Available: http://www.zdnet.com/article/why-microsoft-is-turning-into-an-open-source-company/. [Accessed: 07-Apr-2017].

[4] J. West and S. Gallagher, "Challenges of open innovation: the paradox of firm investment in open-source software," *R&d Manag.*, vol. 36, no. 3, pp. 319–331, 2006.

[5] J. Henkel, "Selective revealing in open innovation processes: the case of embedded Linux," *Res. Policy*, vol. 35, no. 7, pp. 953–969, 2006.

[6] L. Dahlander and M. W. Wallin, “A man on the inside: unlocking communities as complementary assets,” *Res. Policy*, vol. 35, no. 8, pp. 1243–1259, 2006.

[7] J. Lerner and J. Tirole, “Some simple economics of open source,” *J. Ind. Econ.*, vol. 50, no. 2, pp. 197–234, 2002.

[8] C. Murphy, K. Buffardi, J. Dehlinger, L. Lambert, and N. Veilleux, “Community engagement with free and open source software,” in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, 2017, pp. 669–670.

[9] K. Fogel, Producing Open Source Software: How to Run a Successful Free Software Project, O’Reilly Media, 2005.

[10] K. Crowston, J. Howison, and H. Annabi, “Information systems success in free and open source software development: theory and measures,” *Softw. Process Improv. Pract.*, vol. 11, no. 2, pp. 123–148, 2006.

[11] K. Crowston, H. Annabi, and J. Howison, “Defining open source software project success,” *ICIS 2003 Proc.*, p. 28, 2003.

[12] R. K. Yin, Case Study Research: Design and Methods, SAGE Publications, 2013.

[13] B. Fitzgerald, “The transformation of open source software,” *MIS Q.*, vol. 30, no. 3, pp. 587–598, Sep. 2006.

[14] J. Matusow, S. McGibbon, and D. Rowe, “Shared source and open solutions: an e-government perspective,” *Proc. OSS*, vol. 1, pp. 263–266, 2005.

[15] P. Rubens, “Does Microsoft really love open source?” [Online]. Available: http://www.cio.com/article/2465694/open-source-development/does-microsoft-really-love-open-source.html. [Accessed: 07-Apr-2017].

[16] T. Kilamo, I. Hammouda, T. Mikkonen, and T. Aaltonen, “From proprietary to open source - growing an open source ecosystem,” *J. Syst. Softw.*, vol. 85, no. 7, pp. 1467–1478, 2012.

[17] A. Bonaccorsi and C. Rossi, “Comparing motivations of individual programmers and firms to take part in the open source movement: From community to business,” *Knowledge, Technol. {&} Policy*, vol. 18, no. 4, pp. 40–64, 2006.

[18] W. Stam, “When does community participation enhance the performance of open source software companies?,” *Res. Policy*, vol. 38, no. 8, pp. 1288–1299, 2009.

[19] A. MacCormack, J. Rusnak, and C. Y. Baldwin, “Exploring the structure of complex software designs: an empirical study of open source and proprietary code,” *Manage. Sci.*, vol. 52, no. 7, pp. 1015–1030, 2006.

[20] J. Corbin and J. Strauss, Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory, 3rd ed. Sage, 2008.

[21] P. S. Kochhar, N. Nagappan, T. Zimmermann, and C. Bird, “Transitioning from closed source to open source: analysis and observations from six projects,” Microsoft Research, Technical Report, Oct. 2015.

[22] S. F. Barrett, “Arduino microcontroller processing for everyone!,” *Synth. Lect. Digit. Circuits Syst.*, vol. 8, no. 4, pp. 1–513, 2013.

[23] S. Krishnamurthy, “Cave or community?: An empirical examination of 100 mature open source projects,” *First Monday*, 2002.

[24] T. Norikane, A. Ihara, and K. Matsumoto, “Which review feedback did long-term contributors get on OSS projects?,” in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2017, pp. 571–572.

[25] S. Spaeth, G. von Krogh, and F. He, “Research Note—Perceived firm attributes and intrinsic motivation in sponsored open source software projects,” *Inf. Syst. Res.*, vol. 26, no. 1, pp. 224–237, 2015.

[26] K. Crowston, K. Wei, Q. Li, and J. Howison, “Core and periphery in free/libre and open source software team communications,” in *System Sciences, 2006. HICSS’06. Proceedings of the 39th Annual Hawaii International Conference on*, 2006, vol. 6, p. 118a--118a.

[27] P. M. Johnson, H. Kou, M. Paulding, Q. Zhang, A. Kagawa, and T. Yamashita, “Improving software development management through software project telemetry,” *IEEE Softw.*, vol. 22, no. 4, pp. 76–85, Jul. 2005.

[28] H. Borges, A. Hora, and M. T. Valente, “Predicting the popularity of GitHub repositories,” in *Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering*, 2016, p. 9:1--9:10.

[29] D. T. Campbell and D. W. Fiske, “Convergent and discriminant validation by the multitrait-multimethod matrix.,” *Psychol. Bull.*, vol. 56, no. 2, p. 81, 1959.

[30] W. H. Kruskal and W. A. Wallis, “Use of ranks in one-criterion variance analysis,” *J. Am. Stat. Assoc.*, vol. 47, no. 260, pp. 583–621, 1952.

[31] K. Crowston and J. Howison, “Hierarchy and centralization in free and open source software team communications,” *Knowledge, Technol. {&} Policy*, vol. 18, no. 4, pp. 65–85, 2006.

[32] D. R. Denison, Corporate culture and organizational effectiveness, John Wiley & Sons, 1990.