Participatory Motivations and Determinants of Success in Open Source Software

Sarah Rittgers

Thesis Advisor: Patrick Cozzi

EAS 499 Senior Capstone Thesis

Department of Computer and Information Science

School of Engineering and Applied Science

The University of Pennsylvania

May 1st, 2019

Acknowledgements

I would like to thank Jean Paoli, former President of Microsoft Open Technologies and currently the founder of Docugami, and Gabby Getz, a Software Developer at Cesium, for agreeing to take time out of their schedules to be interviewed. Their input provided critical real-world information that shaped the direction of this thesis.

I would also like to give special thanks to Patrick Cozzi, a Principal Graphics Architect at Analytical Graphics, creator of Cesium, and lecturer at the University of Pennsylvania. During the spring 2019 semester, he guided my thesis from conception to its final state and was always willing to offer advice, valuable sources, and introductions to experts in the field.

Table of Contents

1. Introduction

Interest in open source has soared in the past decade as individuals and companies have discovered the power of unconstrained collaboration. While historically grown out of the community volunteer model, companies have increasingly invested time and resources in developing both their own and external open source projects. What we have witnessed through these efforts is the development of new standards and expectations for companies regarding their participation in these communities.

For those unfamiliar with community development models, the rationale behind why a party would choose to participate in open source or how such projects could succeed and make a profit when juxtaposed with proprietary solutions may seem unclear. We will closely examine these motivations, which will provide insight into the numerous benefits that can be realized from open source involvement.

Despite this newfound enthusiasm, it is no secret that the majority of these projects do not garner as much attention as Kubernetes or TensorFlow, for example. Millions of public repositories exist on code sharing sites such as GitHub and SourceForge, and yet few of these become the community-backed and fast-paced projects that we attribute to open source. Granted, not all of these have the goal of growing an open source community, but those that do seek to expand beyond their own internal developers often face challenges that eventually result in failure. As such, we will shed light on factors for both the success and failure of projects by drawing on relevant historical examples. This exploration will primarily take place from the viewpoint of companies participating in the open source community.

We will conclude with some speculation on the future direction of open source, focusing on industries where open models and development have the potential to lead to incredible growth and success. Our discussion will also include current changes within the community and areas for improvement as it continues to evolve.

2. Background

2.1. Terminology

a. Open Source

Open Source (OS) – We will use a general definition that is a subset of the requirements set forth by the Open Source Initiative. The software must: 1) provide redistribution without cost; 2) distribute both the source code and compiled version; and 3) permit changes to the code and development of derived efforts [1].

Closed Source – Also referred to as proprietary, software whose source code is not available publicly. Usually, the code is under a license that restricts modifications and redistribution.

b. Community Roles

User – People that rely on the open source project and integrate dependencies into their own project. Analogous to the consumer of a firm's product, they provide demand and feedback.

Contributor – Developers that submit contributions back to the project through pull requests (PRs). The contribution can range from a large feature to bug fixes or minor updates to documentation.

Maintainer – Responsible for project direction and many logistics related to handling contributions including reviewing PRs and delegating tasks.

c. Legal Concepts

Contributor License Agreement (CLA) – Form signed by contributors, individual or corporate, granting full permission to use and distribute those contributions.

Copyright – By default, all open source code is protected under copyright law. As such, only the creator has the right to create derivative works. Licenses exist to lift this "limitation."

Patent – Software protected by a patent prevents others from using, modifying, or redistributing it without the permission of the holder. The use of patents with open source software is contentious and licenses have been created with explicit clauses against the use of patents, such as the Apache license.

d. Licenses

Copyleft License – Copyleft licenses require that derivative works also adopt the same license. The most common and widespread copyleft license is the General Public License

(GPL). The restriction prevents code under the GPL from being mixed with proprietary software.

Permissive License – These licenses typically work well in business settings because derivative works are not required to adopt the same license. In particular, so long as the original code is properly attributed, derivatives are not required to adopt a license at all and may be closed source. There are several licenses that are commonly used:

i. MIT – The most commonly used permissive license. Permits distribution and modification under different terms [2].
ii. Apache License 2.0 – Similar to the MIT license, but provides explicit patent grants which seek to discourage leveraging patents by preventing the user from suing the provider over the patent [2].

2.2. Historical Context

The concept of open source code has actually been around since the mid 20th century when the World Wide Web had not yet been developed and the communal hacker culture was at the forefront [3]. However, in the 1980's, MIT licensed code to a commercial company that its developers, including Richard Stallman, had created. In 1985 Stallman, concerned about the shift from open to proprietary software, founded the Free Software Foundation and pioneered the free software movement. Free software emphasizes that the code is free for distribution, study, and alterations (not free of cost), and the hallmark of this movement was the GPL license. This license deliberately prevents the mix of code under the GPL with proprietary code, thus limiting the extent to which commercial entities can be involved. The premise of free software was established on this ideology.

At the same time, programmers at the University of California, Berkeley were working on the Berkeley Software Distribution (BSD) operating system. This group became an "early example of non-ideological free software" as members practiced open sharing and yet did not emphasize it as the group's core ideology [4]. The X license followed some years later, which explicitly permitted modifications with proprietary software [4]. As more permissive licenses developed, private companies began to participate in the culture of software sharing. In 1998, the concept of open source software was born with the creation of the Open Source Initiative [4].

This set the stage for acceptance of open source code comingled with proprietary works, allowing for companies to take on active roles in the open source community. The second decade after its formation witnessed the hiring of employees specifically tasked with contributing to particular open source technologies, popularity in open source infrastructure components over off-the-shelf products, and company and foundation sponsorship of projects rather than informal volunteer communities [5]. It is this open source culture that we will predominantly investigate in this thesis.

3. Motivations: Why Contribute?

3.1 Individual Contributors

a. Intrinsic Motivations

Given that open source mainly grew out of a volunteer community model, a considerable amount of research has been devoted to studying the motivations behind participant's enthusiasm to contribute, in particular those altruistic motivations that result in nothing other than personal reward. Despite the growth in corporate-backed contributions by employees, which provides an external motivation, these developers often still possess intrinsic drive as well.

A study published in 2016 sent surveys to first-time contributors from Mozilla and GNOME projects. When ranking motivations provided in a list from highest to lowest, the study found that the gratification earned or the intellectual challenge from programming was the top choice for 37.6% and one of the top three choices for 86% of respondents [6]. Additionally, contribution for the sake of learning accounted for 19.4% of first-ranked motivations and 72% of the top three motivations [5]. This demonstrates that a large proportion of contributors do so because of personal satisfaction and growth. In the same study, it was found that half of all respondents identified their contribution as an act of altruism [5].

Another investigation surveyed undergraduate students in order to understand their motivations. Because it was unlikely that these students were being financially rewarded for their contributions, their intrinsic motivations are of particular interest. It was found that adoption of open source was significantly and positively correlated with intrinsic motivations to learn and to experience some emotional stimulation, such as joy or excitement [6]. One final paper uncovered from their survey that 10 out of 52 core developers were motivated by a desire to volunteer and provide tools and support to the community while 7 had interest in the domain [7]. A paper published on one-time contributors discovered that the second most common motivation was a want to give back to the community [8]. It is evident that contributors enjoy the personal growth and challenges associated with open source. Developers gain a sense of fulfillment through engaging with the community and making improvements to projects that will be of use to others.

b. Extrinsic Motivations

Contributors also consider a number of extrinsic motivations for which they receive some form of reward external to themselves. The study surveying core developers found that the primary motivation, accounting for 31 out of 52 of the responses, was to improve the project because they are using it [7]. 5 out of 52 of these respondents stated it was explicitly because they are paid to do so [7]. 25.8% of newcomers ranked contributing for their own need as their primary reason, although only 40.9% ranked it as one of their top three motivations [5]. In fact, the study on one-time contributors found their primary motivation was because they needed a bug fix either for the use of them or their employer, the third most cited motivation was for work, and the fourth most popular reason was to add a new feature [8]. Our first observation is that people who use open source have a direct incentive to contribute back because bug fixes and new features benefit their

personal efforts. In addition, the growing widespread use of open source in the corporate world has significantly contributed to these extrinsic motivations through the provision of monetary rewards. Those that previously wanted to contribute to such projects but could not support themselves in such a manner now have the capability to do so. Individual contributors may find incentive in any combination of these extrinsic and intrinsic motives.

3.2 Benefits for Firms

Firms must also have some motivations that outweigh the costs associated with open source project management and disclosure of intellectual property. We now examine the primary motivators.

a. A Game Theory Analysis

The question of why free riding is not a greater cause for concern is a natural one to ask. Without much background, it is challenging to reason why a company would choose to release intellectual property when competitors can adopt the source code and steal market share. We have a tendency to view commercial businesses as profit maximizers, and open source development seems contrary to this goal. Using the iterated prisoner's dilemma, one piece of research models two individuals or teams participating in a two-person game. It is well known that in a prisoner's dilemma game of fixed iterations, the Nash equilibrium is for both parties to defect. However, in open source development the number of iterations is unknown because the extent to which involved parties must interact in a project cannot be known in advance, and cooperative outcomes are possible in the iterated version [9]. Cooperation here is defined as participating, or investing effort, while defection refers to investing little or no effort [9].

		Player B	
Player A		Cooperate	Defect
	Cooperate	$(F-I, F-I)$	$(F-I-I', F)$
	Defect	$(F, F-I-I')$	(0,0)

Figure 1: Payoff Matrix for Two-Person Iterative Prisoner's Dilemma [9]

The payoffs of the two parties according to their decisions are given in Figure 1, where F is the utility received, I is effort invested, and I' is the additional effort invested to make up for the lack of the other party's contribution [9]. For example, the optimal outcome for Player A is to defect while Player B cooperates, which is effectively the free riding scenario. We use p to denote the probability of cooperating given that the opponent cooperated in the previous round and q to be the probability of cooperating after the opponent defected. Due to the iterative, or tit-for-tat, nature of these interactions, the authors construct reactive strategies. That is, for each strategy of inter- and intra- group interactions $E_i^A = ((p_i, q_i), (p_i, q_i))$ of Player A at iteration i, there exists a strategy $E_i^B = ((p_i^*, q_i^*), (p_i^*, q_i^*))$ for Player B [9]. As shown in Figure 2, we find that both p and q stabilize at values of nearly 1 [9]. In other words, the two players end up cooperating regardless of what the other does. The authors conclude that this demonstrates cooperation, or the evolution

of open source communities, is natural and that sound decisions, not just altruism, serve as motivations [9].

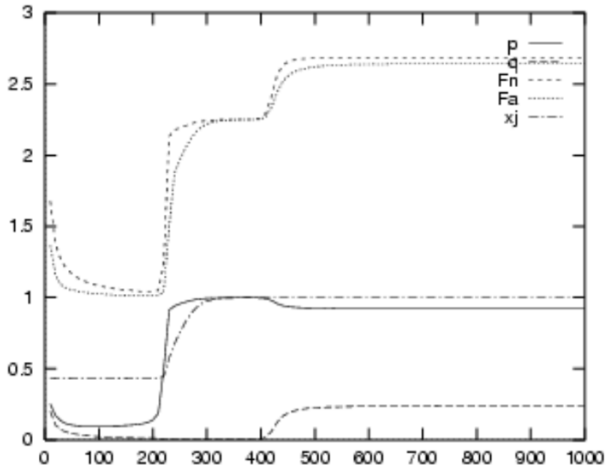

Figure 2: Results of Two-Person Iterative Prisoner's Dilemma [9]

We can justify these results intuitively as well. Eric Raymond, well known for *The Cathedral and the Bazaar*, argues that the tragedy of the commons is not an issue in software because greater use actually increases its value rather than leads to its degradation [10]. When a project becomes more popular, it will experience an increasing number of resources and contributions that improve the code quality and features. Because developers need resolutions to their problems within time constraints, it is more efficient to develop such solutions themselves rather than free ride and wait on someone else [10]. Additionally, these solutions are then merged upstream into the master project because it transfers the burden of maintaining the patch from the individual to the project owners and community [10]. Thus, developers, including corporations, are driven to collaboration within the open source community because of a number of motivators that increase their overall utility. We now expand upon these individual motivations.

b. Faster Innovation

One of the drivers for electing to open source code and participate in development is that it allows for companies to innovate at a quicker pace. For example, one case in which the open sourcing of the project revolutionized its development is the release of Apache Phoenix by Salesforce. This project originally started as the open source project Phoenix. Salesforce quickly realized that Phoenix did not become fully successful until it was receiving contributions from a diverse set of people invested in the project's future [11].

Not only did this mean that Phoenix was no longer dependent on only one company, but its growth was accelerating because a significantly greater number of people realized they too could reap the benefits of the project and began to contribute. Salesforce came to the understanding that a community of diverse people can build a project into much more than what a small team can, and eventually donated the project to the Apache Software Foundation (ASF) [11]. Besides the fact that this was faster than Salesforce alone developing Phoenix, it was also much more

economically efficient for the community to work on a joint project rather than each company come up with its own individual solution to the same problem. Both Salesforce and others were able to benefit from a higher-quality product than had this project been developed in a proprietary manner.

Researchers studying firm productivity when using open source created a model to calculate value when using nonpecuniary open source software. To measure value added, they formulated an equation based off the Cobb-Douglas production function, modifying it to include a nonpecuniary term to account for inputs such as nonpecuniary open source that have no price [12]. In the study, the researchers only included open source operating systems in the calculation for nonpecuniary factors, meaning their result is an underestimate of the actual value added since it is highly likely these firms used additional open source products [12]. The results of the study demonstrate that open source development leads to a significant and positive increase in value added and overall productivity [12]. The model found that, for example, for two firms at the 95th percentile in IT-intensity, meaning the firms are in industries producing IT, an increase by 1% in the degree of open source utilization leads to a value increase that can be monetized to $103,920 [12]. Incorporating open source fills gaps in one's workflow that previously required either proprietary or custom solutions, allowing firms to focus on the product and not secondary tools. Since these tools are being developed more quickly, the projects are more responsive to company needs, and the company in turn is able to develop its own products more quickly. Thus, open source has a waterfall effect in which faster innovation of the public project leads to higher levels of productivity and value for the company in their proprietary solutions.

c. Standardization and Influence

Businesses can gain influence through participating in open source and therefore engage in discussion surrounding standardization, advocating for their own wishes and concerns. In 2012, a number of companies in the automobile industry announced the creation of Automotive Grade Linux (AGL), an open source initiative to develop a software platform for vehicles that includes such features as music and navigation [13]. Cars traditionally have over 100 million lines of code that are essentially written from scratch, but the current version of AGL will separate this so that roughly 70% of that code is shared among vehicles while the remaining lines are customized to the individual car [13]. Not only does this project lead to faster innovation as previously mentioned, but Toyota's participation in AGL means that the company has the power to influence the direction of this standard platform that ideally will be the model for all future cars. Toyota can ensure that its own needs are sufficiently met throughout the project's development. This leadership also grants Toyota greater leverage in the automotive industry for major decisions going forward.

d. Interoperability

Interoperability refers to the ability for separate computer software or systems to interact with one another. Jean Paoli, former president of Microsoft Open Technologies, states that one of the principle concerns when developing Azure was interoperability [14]. This was ensured through creating initial software development kits (SDKs), seeking methods to develop for Azure with non-Microsoft open source tools like Eclipse, and partnering with the community to create

VMDepot, the first open source marketplace [14]. VMDepot provided prepackaged open source projects on virtual machines with deep integrations to make them run better on Azure [14]. In summary, Microsoft aimed to appeal to open source developers and ensure Azure's ease of use through integration with well-known open source projects.

Adobe also defined interoperability as a key concern when creating the open source Adobe Experience Data Models (XDM). Experience data refers to data that is generated from interactions occurring over multiple different channels, such as across the web, mobile devices, and IoT technology. Accordingly, the need for a common language between these devices to organize and understand data is crucial [15]. It is evident that open source facilitates the development of this common language as developers from different companies and industries can develop a standard and thus permit the interoperability of these devices.

e. Complementary Services

Another benefit for firms is that open source projects provide a platform on top of which they can build proprietary complementary services. Sentry is an error tracking company that publicizes its entire codebase on GitHub, except features related to billing [16]. Consequently, anyone can download this source code and use it free of cost if they have the means to host the project themselves. However, Sentry realizes that many companies do not want to expend their limited energy and resources on a support tool not directly related to business goals, and therefore provides a software as a service (SaaS) model that hosts the server on the cloud [16]. In addition to a free tier with the basic functionality of error tracking meant for personal use, Sentry also offers team and business tiers with higher caps on the number of errors reported per month, a longer error history, and support for other features like third-party integration. This framework allows Sentry to focus company efforts on areas that are often overlooked such as security and compliance [16]. In this model, all parties are offered the flexibility and independence to identify their own needs and act accordingly. Not only does the complementary product give Sentry the benefits of open source development, but it serves the use cases of anyone interested in the project by allowing those who wish to host the open source project for free to do so or pay for the privilege of not hosting it themselves.

One unique example of a company that has built its entire ethos around providing services for open source is Red Hat. While Red Hat offers numerous services, one of the central aspects of its business model is its subscription. While open source projects are known for having their challenges with installation and configuring, Linux in particular is infamous for being difficult to set up, even for experienced developers. As a leader of the OS community, Red Hat argues that the knowledge of its support team stands out because of its decades of experience actually contributing to these projects [17]. Red Hat also develops the Customer Portal, providing 24/7 support with access to up-to-date documentation and experienced engineers. Moreover, the company provides guarantees in security monitoring and stable code, access to seemingly unlimited technical information from the Knowledgebase, and the ability to influence future development direction through feedback [17]. Red Hat is in the best position to offer services and support because of the company's direct interaction with the open source code and it is the ultimate example of the complementary product model.

f. Lower Costs

Companies often flock to open source solutions over proprietary products since costs of use tend to be lower since, for example, technical support is dissipated over the community and more importantly they can avoid the threat of vendor lock-in. Vendor lock-in refers to a company's inability to switch to another provider without incurring significant costs. Open source, on the other hand, renders this concern irrelevant. Because the source code is available, users have the option to continue participating in the community or download the project and resume development themselves.

The history of Apache OpenOffice provides some insight into this economic flexibility that open source offers. OpenOffice originated with Sun Microsystems and was then continued under Oracle after the acquisition [18]. However, many of the developers left over leadership concerns and forked the project to create LibreOffice, providing another word editor for users to choose [18]. After failed attempts with building a proprietary solution and clashes with the community, Oracle decided to end all efforts related to the project [18]. Recognizing that many were still reliant on and interested in the continuation of OpenOffice, Oracle donated it to the ASF, where it has since been developed as Apache OpenOffice [18]. Despite this donation, though, the project has faced trials attracting contributors and most developers have turned their efforts to LibreOffice. In essence, users did not need to fear becoming solely reliant on OpenOffice because there was always the opportunity to develop the code on their own or find an alternative solution. While the consequences may not seem so dire in this context since there are many alternative word processors, this example demonstrates the adaptability inherent to open source development. Had OpenOffice instead been some proprietary cloud service deeply integrated within a business entity's workflow, the company would incur numerous costs trying to switch if a situation such as the one above was to occur.

g. Improved Code

Another benefit that firms gain is an improved codebase quality. In fact, a study by Forrester Research found that mature open source software met or exceeded expectations 92% of the time [19]. Arpit Mathur, an engineer at Comcast, has noticed that the quality of his team's work is higher when contributing to an open source project, possibly because it will be available for public scrutiny [20]. As a result, developers want to submit their best possible work to preserve their reputation and future opportunities. Another incentive is for individuals looking to build a portfolio for recruiting. Since companies that are hiring an individual who has contributed to open source can actually look at the quality of their work, said individual should ensure they are publishing quality code. Companies also have an incentive to maintain their codebase to attract contributors because experienced developers do not wish to waste time navigating unreadable code. One more explanation for this phenomenon is that the source code is available to extensive examination and testing from a diverse community [19]. This often leads to the implementation of best practices, which may not occur in proprietary code because it is hidden from external review and the firm may emphasize quantity over quality. All in all, open development encourages both parties to only integrate high-quality contributions into the source code.

h. Reputation and Trust

Companies are able to improve their reputation and thus build trust as a good Samaritan through participating in open source. One of the goals for Salesforce's open source program is to earn favorable repute among developers [11]. Through a positive company image, companies gain the trust of the open source community. According to the TODO Group, this can alter perceptions of the company to seem more attractive to developers and allow it to exercise greater influence in projects to which it contributes [21]. Another consequence is that developers may be more willing to contribute to open source projects owned by the company if they are perceived as fair and trustworthy. Several years ago, Comcast open sourced Speed-TestJS, a tool to test Internet speed. The company cites that its motivation was to be transparent about how they measure speed. Since one of Comcast's services is providing cable, this allowed customers to test the tool and develop more confidence that Comcast was holding up to its word [22]. As a result, the company gains a positive public image which leads to developers wanting to participate and interact in the project community. Additionally, this may make Comcast seem like an appealing workplace, thereby aiding recruiting efforts.

i. Recruiting

The technology industry is experiencing a workforce shortage of qualified employees. Having visible code that potential employees can examine is a huge benefit for companies trying to excite their candidates since they now have projects to point to [19]. This benefits recruiters as well since they can use the list of contributors as a pool of potential candidates with examples of their work already accessible [23]. Additionally, developers are always looking to increase their skillset, and having the ability to work on open source and be trained to work on such projects is an appealing incentive to work for a company.

Considering that the number of developers with experience in open source is a small subset of this already limited resource pool, employees who can assume these roles are in incredibly high demand. According to the 2017 Open Source Jobs Report, 89% of hiring managers are looking for developers with the right open source skills and 60% are looking to hire people with open source experience [24]. This is made even more challenging because experienced open source developers tend to be loyal to the project and community, and not necessarily to the company for which they work [24]. The 2018 version of the same report found that 55% of open source developers believe it would be easy to find a new job [25]. The solution to this goes hand in hand with building a high-quality codebase and becoming a reputable company. Developers will be more convinced that the company cares about its employees and the welfare of the project itself.

3.3 Common Misconceptions

a. Security

A common fallacy associated with open source is that it is less secure because the source code is available for attackers to deconstruct. On the contrary, a survey performed in 2017 by GitHub found that of the topics the survey inquired about, security was the only one where the majority (58%) of respondents believed open source to be better than proprietary [26]. One

important distinction is that the mere act of releasing source code publicly does not change the security of the system, because it will have had the same number of bugs before and after, but it increases its exposure [27]. That is, while attackers may have open access, so will the rest of the development community and they will be quicker and more incentivized to fix any vulnerabilities that have been found. More eyes will be monitoring the codebase for potential issues and public development invites a diverse set of opinions to determine the optimal solution.

On the other hand, if the code is kept hidden from everyone except the vendor, then all users are at their mercy and it is likely that bugs will remain a vulnerability for a long time [28]. The defender ends up in a place of more risk with proprietary code because the attacker can still easily take advantage of bugs, but the defender is left unaware of patches that are needed [27]. The key is that open source benefits from the peer review process. On the condition that others are taking the time to review the code for security problems in both proprietary and open source projects, this makes it more likely for vulnerabilities to be found and fixed in the latter case. To quote Eric Raymond, "given enough eyeballs, all bugs are shallow."

b. Cost-Free

It is often assumed that open source is free of all costs, both for users and developers. From the definition of open source, it is required that the source code by distributed free of cost, but this does not imply that services layered on top of this code must be free. As has been touched upon in the section addressing complementary services, there are several means to earn a profit from open source. This includes acting as a consultant and building a piece of proprietary code on top of the open source, which is how many SaaS products are structured [14]. It is common, however, that open source services have a propensity to cost less than their proprietary counterparts. For instance, open source tends to enjoy lower maintenance costs because the responsibility of features, such as support, is generally distributed over the entire community [29].

The task of owning and maintaining an open source project is also not cost-ridden. Although the company will receive contributions and improvements to the code or documentation without explicitly paying these developers, there is a real cost associated with creating and cultivating a community for successful projects [30]. We will touch upon this in the next several sections, but there is a plethora of areas in which the company must expend time and resources, including communication, integration, and testing tools, to make a project worth contributing to for developers. The primary difference in terms of cost is that while up-front costs may equal, if not exceed, proprietary solutions, the long-term costs of open source ownership are lower because they are distributed over the entire community.

c. Code Stealing

One more frequent misunderstanding associated with open source is that competitors will copy the published code and integrate it into their own products. Bryan Cantrill, CTO at Joyent, states that this is not a concern because competitors are simply too prideful [31]. Using code from another company and claiming it as one's own would reflect poorly on them, and consequently harm their reputation and trust. It would paint the firm as a follower rather an innovator. In turn,

the open source community would perceive that company as a poor sport and be skeptical to cooperate with them going forward.

4. Deciding to Open Source

While firms receive many benefits from open sourcing their code, it is not necessarily true that all projects should be publicly released. We will consider when an open source project might be the best solution and what rationales exist for not open sourcing code.

4.1 When to Start a Project

a. Define a Reason

A company should identify a distinct reason for deciding to open source. At Autodesk, this consists of a business decision regarding the potential benefits and costs. Guy Martin, the director of Open@Autodesk, states that the determination of whether to open source comes down to asking business leaders and engineers whether it would bring strategic value to the company and how it would affect the company's ability to patent [32]. Defining an objective keeps the project on track and will affect future decisions and responses to problems in the future. For example, if interoperability is a goal, then we would expect the project to emphasize recruiting diverse opinions from influencers in the industry. This identification process also clarifies project goals for contributors and may attract new participants to the community if they have similar interests.

b. Unique Needs

A new open source project should not be started if an existing one can meet the user's needs. One of the strengths and hallmarks of open source is collaboration, and thus a firm should contribute to an existing project even if a competitor is the one spearheading [21]. Twitter faced this consideration when they were deciding whether to adopt Apache Kafka, a distributed streaming platform for handling data. Twitter had previously used an earlier version of Kafka but replaced it with the custom solution EventBus because it did not meet their I/O needs [33]. However, after some time Twitter reevaluated Kafka and came to the conclusion that the team should switch back to the open source solution [33]. The first reason was that Kafka exhibited lower latency and significantly lowered costs for users [33]. Twitter also believed that Kafka's existing community would be of great benefit because the team could take advantage of the patches others were contributing, the extensive online support system when encountering development problems, and the comprehensive documentation that tends to accompany popular projects [33]. For Twitter, the resources that came with a network of developers outweighed the independence of allotting a few internal developers to EventBus.

On the other hand, Netflix decided to build their own internal project Titus, a container management platform which was later open sourced, when no solutions meeting their needs were available. Netflix has unique requirements for integration with AWS, and consequently decided that building its own solution that leveraged these needed features was in the company's best interest [34]. Netflix also faces distinct scaling requirements such as launching applications that vary in computing, memory, and network needs, and there was no single existing solution that addressed all of these issues [34]. Despite the benefits that come with already established open source projects, none of them aligned with the company's goals and could sufficiently meet all of

its use cases. Electing to begin a new open source project is appropriate and justified in these contexts.

4.2 What to Release

We now consider which, if any, projects should not be open sourced without some thought about what is in the best interest of the project and company. One noticeable observation to make is that, assuming the goal is to build a wide audience, open source is most successful when the project is meant for the use of developers and not geared directly towards nontechnical consumers [35]. If the end users lack the skill to contribute back to the community, then one cannot expect a large contributor following to develop.

In general, projects that can be labeled as infrastructure, libraries, or other secondary tools that are needed for development tend to be appropriate for open source [35]. Salesforce focuses on tools that aid the efforts of other companies in addition to samples and SDKs that benefit the customers these companies [11]. Such projects have the potential to become popular and gain a large community base because developers from a diverse set of backgrounds will be interested in it to simplify their own development or improve customer satisfaction. Conversely, a project should not be open sourced if it is the main product of the company. Google mentions in its open source documentation that the primary reason for not open sourcing is because the project has ties to its secret sauce and releasing it can either lead to security issues or harm the company's competitive advantage [23]. For example, it would not be wise for Google to release every detail of its PageRank algorithm because this is what differentiates its search engine.

Additionally, code should not become open sourced if it is mission-critical [21]. These projects tend to be time-constrained, have unique requirements that the developer must meet, and require a substantial amount of testing and security. One example of a contentious and problematic mission-critical open source project is the integrated electronic health record (iEHR) built for the Department of Defense (DoD) and Veterans Affairs (VA) [36]. Billions have been spent on this project with no success in accomplishing its original goals [36]. This project has an extensive number of requirements needed to make it suitable for government employment and its use cases are likely of little interest to anyone other than the government. It also forms the main core of the database, meaning that there exists very little tolerance for error. As such, the needs of iEHR are far more massive than what the open source community can and wishes to provide.

5. Defining Measure of Success

5.1 Numerical Metrics

There are a number of metrics that can be kept to give a broad idea of the health of an open source project. One of the most widely used metrics is the number of contributors and the ratio of external to internal contributors [37]. In particular, companies should be looking for a diverse range of contributors that are not exclusively from within the company and the number of external contributors should be growing over time as new users integrate the project into their own code [37]. Another statistic that is often maintained is the number of pull requests (PRs) submitted, open, and accepted, including how long they were open [37]. This is a signal of overall external interest in the project [37]. Kubernetes has managed its success by determining whether users are excited and using the project, and this is one such marker [37]. The length of time also gives evidence for how quickly PRs are being reviewed and if there may be a need to streamline this. Contributors can become frustrated if a branch they would like to merge has been waiting for a review, which may discourage them from future participation. One of Facebook's practices is to select the five projects with the largest number of open PRs, pinpoint issues, and discuss them with maintainers [37].

Another indication of the extent to which the project has been adopted and how quickly maintainers are responding to users is the number of issues submitted and the amount of time they remain open [37]. Having many issues can be a positive sign since it means people are using the project. While it is expected that a project will always have issues waiting to be tackled, if they have been open for an unnecessarily long time, it can cue a need for either more maintainers or for current maintainers to focus on addressing issues. The number of commits per contributor, and whether these are internal or external contributions, is also frequently used by projects [37]. Ideally, these numbers should be increasing as evidence that new users are interested and old contributors are not leaving [37]. Similarly, the number of external developers is often tracked. If this number stops growing or begins to decrease, it can indicate anything from a lack of interest to a project that has fully matured [37]. At Red Hat, although the definition of success varies by project, the company has recognized that in projects that are heavy in internal contributions like Fedora, keeping track of the number of contributions from other companies has become a useful tool [38].Whether it is an indication of success or future trouble depends on a number of other factors that may not be visible from numbers alone. It is important to keep in mind that while these metrics provide a general overview of trend, they should not be used as the only indicator [37].

5.2 Other Tracking Methods

There exist many subjective methods for evaluating success as well. Duane O'Brien, former open source programs evangelist at PayPal, recommends that every company ask four questions when assessing the health of an open source project. The first is "Who cares?" because if the answer is no one, then either the project may not be relevant or more effort need to be put into getting the word out [39]. Companies should also ask "Are we still using it?" since it is bad practice to open source a project that is not in use [39]. This can often be perceived as unloading the responsibility of the code onto the community. These projects tend to suffer from a lack of

investment and maintenance because the company is not familiar with best use cases, which can damage the company's image. The third question is "Are we committing our own resources?" [39]. If the company is not investing its own capital, then too much pressure may be placed on engineers to maintain the projects [39]. It can also be a signal to others that the firm is not fully committed. Finally, the question "Can we develop it all in the open?" should be asked [39]. If engineers do not want to participate in the community, this can signal cultural issues that need to be addressed [39]. Successful open source projects rely on diverse collaboration to further innovation, which necessitates engaging with external developers.

These methods can often vary on a company by company basis. IBM recognizes that successful open source communities exhibit certain properties and therefore employs five essential characteristics for evaluation. Responsible licensing and understanding the definitions and limitations of each license associated with a project is integral [40]. If a project breaks the license agreement, this can generate far-reaching consequences. The company also looks for an accessible commit process with a clearly outlined process for contribution so that external contributors feel welcomed [40]. Otherwise, the company may experience trouble growing its base if the barrier to entry is too high. A diverse ecosystem is sought out with various companies and independent software vendors using the project in their own products [40]. This will bring in diverse opinions for project development. IBM also values an active community and a defined process for contributors to grow their reputation [40]. The basis for this is to provide incentives for contributors to continue to involvement and make them feel recognized for their efforts. Lastly, open governance is sought out [40]. A company that claims to be open source but does not practice the culture can develop a poor name in the community and also does not benefit from open collaboration.

One final case study on this topic involves Facebook, which conducts a biannual survey that specifically targets the relationship between the open source program, recruiting endeavors, and internal developer satisfaction. This survey asks new hires three questions: are they aware of the open source program, how did that awareness influence their decision to join Facebook and does their experience with open source apply to their work now [37]. According to Christine Abernathy, Open Source Developer Advocate at Facebook, the survey grants the company an overview of the health of open source culture and developers' perspectives on projects [37]. From this, the company can also discern whether employees are satisfied in their current positions, if recruiting efforts have been successful, and how the open source community may perceive the company.

We have reviewed just some of the many methods that can be used for evaluating a project and company's success with practicing open source. All projects vary in their unique needs and technical requirements, and accordingly will vary in their precise definitions and measurements of success. However, overall patterns of community engagement, project growth, and preservation of company reputation are recognized in successful open source projects.

6. Infrastructure Behind Successful Projects

6.1 Open Source Program Office

Companies often create an open source programs office (OSPO), or similarly named, with the objective of overseeing all interactions with open source. One of the roles of the office is to establish effective processes for contributors to follow during development. In 2017, Google released documentation on its open source policies so that those curious could understand how the company internally employs open source [41]. The extensive documentation covers such procedures as getting approval, adding third-party code, and submitting patches [23]. These usage and interaction policies elucidate any sources of misunderstanding or confusion for developers and create a clear roadmap for those interested in working with open source.

The OSPO is also tasked with tying all open source related endeavors back to business goals. For example, Oath's (now known as Verizon Media) OSPO has developed five department aims with each targeted to achieving a certain company-wide aspiration. One of these goals is to be perceived as being open source friendly so others are willing to collaborate [42]. This relates to building a positive reputation, which gives the company more influence and fuels innovation forward as contributors are eager to participate. A second goal of the company's is for engineers to know Oath as a great place to work on open source projects [42]. In terms of business goals, this supports recruiting efforts as people will want to work and remain at the company. The OSPO guides the development of open source projects in parallel with the needs of the business.

Another responsibility of the OSPO is to maintain compliance and handle other legal-related functions. Compliance typically contains a legal representative who reviews all interactions with open source, such as usage and distribution, and is a source of guidance regarding licenses [43]. The team may also consist of a representative from engineering or product who requests approval for open source projects and ensures compliance from engineers through code reviews [43]. Without individuals to fulfill these formal roles, licenses or other legal agreements may be violated and the company can face unrecoverable trouble.

Lastly, the OSPO has the overarching task of providing general support to engineers. Comcast states that the goal of its OSPO is to serve as a consultant to its employees through providing guidance, advice, and coaching [44]. This can also mean simplifying the contribution process for developers to lower any barriers that may exist. At Facebook, this involves a Tools team that builds custom systems for managing open source projects [45]. Through supporting developer efforts, both internal and external contributors can focus less on tedious tasks and more on innovation, increasing project velocity and improving the experience for all parties.

6.2 Communication Channels

A critical aspect of any successful open source project is having the correct means of communication because developers working on the same project are likely not in the same room or working at the same company. Deborah Bryant, Senior Director of Open Source and Standards at Red Hat, believes the most important tool used at the company is a real-time communication

system Internet Relay Chat (IRC) because developers can receive immediate responses to questions and facilitates discussion [38]. Another indispensable platform is mailing lists, which is what the ASF sets up for every project [46]. This asynchronous method of communication is crucial for Apache because it creates archives that can be referred back to at a later time and tolerates the fact that many people cannot respond right away in these global communities [46]. One more crucial method for communication is a forum. Cesium, a JavaScript library for 3D mapping, maintains one central forum and recommends that users crosslink responses to create an archive of links [47]. This not only allows users to easily search for answers to questions or pose their own, but also acts as a record-keeping system to look back on historical reasons that decisions were made. Depending on the project, maintainers may wish to implement some combination of the above communication mechanisms, among others, as each serves a unique purpose.

6.3 Technology for Automation

There are a number of tools that automate the development process so that contributors can focus more on the content of their patches and less on the mechanical tasks associated with open source collaboration. Some of the tools in the technology stack that may be used include version control, continuous integration, compliance scanning, and bug tracking.

a. Version Control Systems

Version control (VC) systems are necessary for maintaining patches from a large number of contributors, keeping a historical record, controlling who has access to contribute, distributing source code, resolving conflicts caused by simultaneous changes to the same line of code, and much more. The use of VC is not limited to open source and has become a staple of software development. It provides the benefits of a complete history of all files including users that made the changes, branches to permit the independent development of patches, and traceability so that bugs can be pinpointed to certain changes [48]. VC simplifies the process of collaboration so that developers need not worry about the logistics of many people contributing to the same codebase.

While there is a variety of different options, the most widely used version control is Git, which is itself open source, and GitHub is a popular hosting service for Git. GitHub also provides an easy-to-use interface for source code distribution and accumulates a number of useful metrics, such as number of commits, that can be used to gauge project health. Many companies, such as Facebook, Google, and Capital One currently use GitHub to host their projects.

b. Continuous Integration

Continuous integration (CI) is another technical concept that has been gaining ground as a popular technique for software development. CI encourages committing small changes often with tests running on every commit so that the contributor knows the state and behavior of their changes. By integrating test cases and blocking PRs from merging until all tests pass, the main branch is protected from easily preventable bugs and code quality is maintained. There are a number of CI tools, some common ones being Jenkins, Travis CI, and Go. No matter which one is chosen, owners configure the service to a project's specific test suites and needs.

Cesium is one project that utilizes Travis CI to run tests after every commit and PR [49]. A file called travis.yml specifies such options as the programming language and the steps to build the project [49]. The first tool that is run is JSHint, which detects syntax and style errors [49]. Static linters like this are vital to provide confidence that code quality does not degrade when a group of people with different experience are contributing code to the same project. If JSHint fails, a signal is sent to fail the task so that contributors are aware of fixes that need to be made [49]. Following this, unit tests are run from specs written with Jasmine [49]. The company uses Karma to run its tests in a browser and Electron to allow the browser to run headless [49]. Like with JSHint, the contributor is given a clear indicator of any tests that failed so that they can be fixed before merging the code. The code is built and then deployed to an S3 bucket so that other developers can access and run the changes [49]. Of the files deployed, one is a zip file containing the entire release of the branch [49]. This is particularly useful for further testing that cannot necessarily be covered by a test suite. Since certain bugs may only manifest in the release, unit tests are run once more [49]. Lastly, Cloc is used to gather overall statistics about the codebase [49]. This process is not definitive and there always exists the ability to change the build process and integrate new tools as needs evolve. The true benefit of CI is the degree to which a project can configure the build to its individual use cases. This ultimately saves time for developers and maintainers while ensuring that code quality be of certain standards.

Facebook has enjoyed countless benefits from using CI in production, and not only in their open source projects. The company previously used a cherry-picking development model, but quickly faced scalability issues [50]. They now employ a quasi-continuous system in which commits to the master branch are first pushed to 50% of employees, then 2% of production, and finally to all of production [50]. Each step is monitored so that if a bug is discovered, the push can be rolled back [50]. When the commit is initially made to the master branch, linting and automated testing through unit, integration, and end-to-end tests are performed [50]. As a result, their mobile project has improved drastically from four to one-week releases [50]. Specifically with regards to CI for open source, it alleviates the burden off maintainers so that individual contributors are responsible for monitoring the integrity of their contributions. It also provides a better environment for global development because engineers in different time zones can participate on their own schedules, which is especially relevant to open source projects [50]. CI grants flexibility to developers that alleviates obstacles caused by collaborative development and further feeds into the goal of faster innovation.

c. Other Tools for Quality and Standards

Bug reporting systems are essential for the operations of any software project, and especially in open source. The chance that any software is bug-free is practically zero, and users need a platform for submitting these issues so that developers are aware and can create patches. Such systems also supply a means for getting the community involved since anyone can take on the task of fixing a bug, not just consistent committers. There are many solutions available, both open source and proprietary, such as Bugzilla, JIRA, and GitHub Issues [51].

Another tool that proves to be indispensable for open source scans the code to check for compliance. While these systems are not fully comprehensive, they provide confirmations that certain aspects of a license agreement are not being breached, for example. One tool might survey

the Developer Certificate of Origins (DCO) to determine whether code from another project can be used without license conflicts [51]. Other tools related to managing the legal characteristics of code exist, and some that are often used check copyrights and dependencies [51]. These programs do not replace but rather supplement the role of a legal expert in an OSPO.

d. Custom Tools

Companies with the means often choose to create their own development tools that add further automation to the contribution process. Microsoft created GHCrawler, an open source solution to with the purpose of tracking even more details on GitHub including permission changes and the team that made the commit [52]. The company also created a custom portal for onboarding and managing projects, repositories, and teams [51]. Additionally, Facebook has its own Open Source Tooling team that develops any tools teams may need to simplify project deployment [53]. One such project is ShipIt, which gives project owners the ability to export changes from Facebook's internal codebase and import external PRs [53]. Without ShipIt, developers at the company had to perform a number of mundane manual steps if they wished to interact with the remote project. Companies should take a look at their own contribution processes and determine pain points that could be alleviated through automation.

6.4 Legalities

a. Licenses and Choosing the Right One

The choice of a license defines the intention of a project and can later influence its success as well. According to Jean Paoli, today everyone understands the importance of a good license [14]. It is clear that certain licenses, specifically the GPL, are more restrictive and less business-friendly that others, such as the MIT or Apache. A company would likely not choose to develop under the GPL because it prevents the source code from being integrated with a proprietary project, and thus acts against firm interest. In fact, Bryan Cantrill states that anti-collaborative licensing, referring to the strong copy left, is an anti-pattern for companies because it conflicts with most other licenses and introduces barriers to development [31]. While we will later touch more upon the relevance of the GPL today, we now examine what considerations should be made for business licenses.

License choice can vary depending on the project and business needs. In particular, a company may wish to also consider the community that it is trying to attract [54]. For example, if the project is to be used as a dependency within other projects, such as a library, then it is recommended to use the license that is most commonly employed for similar projects [54]. This limits the potential conflicts that might occur from users wanting to integrate the project into their project but not being able to due to conflicting incompatible licenses. It is also good practice to use the standard if one exists. Facebook tends to use the Berkeley Software Distribution license (BSD), but considers switching if the community has a strong preference [55]. If the goal of the project is to gain the interest of large companies, Apache 2.0 is beneficial because it contains patent protections [54]. Companies should consider their own goals for the project and what restrictions they will place on themselves when deciding on a license.

Moreover, a license can affect who decides to participate in a project. According to the GitHub 2017 open source survey, 64% of respondents said the license is very important in deciding whether to use a project, and of even more interest, 67% said it is very important in deciding whether to contribute [26]. Restrictive licenses may be perceived as less useful to a developer because they limit what that developer is able to create, which in turn increases the risks for adoption [56]. A study of the impact of license choice on users and developers found that those projects with a nonrestrictive license attract greater user interest and that project interest has a positive effect on the amount of development activity [56]. Thus, using a nonrestrictive license limits the community, which can then impact the speed of innovation within the project since there are fewer contributors. For projects that are not corporate-maintained, these factors should be taken into consideration when comparing permissive and copyleft licenses. On the other hand, a project may specifically seek out the GPL or similar if an objective is to prevent corporate use entirely. Again, these decisions depend on clearly defined goals.

b. Patents and Intellectual Property

The effectiveness of patents to protect intellectual property (IP) has long been a debate within the open source community. While patents are most often used to protect corporate interests, their employment can be perceived as a malicious act that contradicts the premise of open development. Kubernetes suggests completely letting go of IP. Craig McLuckie, the founder of Kubernetes, believes donating the project to the Linux Foundation and giving up IP increased participation and led to its success, allowing Google to build proprietary complementary services and churn a profit [57]. He states that if they had instead held onto IP, it would have held the project back and, in turn, harmed the company [57].

In 2017, Facebook faced backlash against the BSD+Patent license it was using for React, a frontend JavaScript library, and a number of other open source projects [58]. This license contains a patent grant to protect Facebook from contributors creating their own patents and using them against the company [59]. Due to the license's limitations and risks, the ASF classified it under Category X, meaning React could not be used in Apache projects [58]. Facebook's use of patents clearly not only provoked the community, but it also hurt Facebook because it severely limited the extent to which the company could collaborate with the community it had intended to reach. After listening to complaints and realizing that the license would only hold the project back, Facebook relicensed React under the MIT license, which does not contain a patent grant [58].

This suggests that patents are especially harmful for general-purpose libraries that, ideally, should have the intention of becoming integrated in as many other projects as possible. However, if companies were not permitted to use any patent protection, this would act as a deep disincentive for participation. The key is providing patent protection on projects that differentiate the company from the market and that do not necessarily have the desire to become all-reaching in their impact. Patents introduce uncertainty regarding the development of competitive products, and thus the community will be skeptical about the project's true intentions. Contributors want to ensure that their patches are benefiting the project and not the commercial entity. The company's reputation can also suffer as a result. This may turn some developers away from contributing and lead to classifications such as that by the ASF. If the project is willing to take these risks for the benefit of protecting IP, then a patent may still be beneficial.

c. Contributor License Agreements

Contributor license agreements (CLAs) define the terms under which IP is contributed to a project and grant the company the right to use contributions submitted by the signee. Many companies use CLAs, including Google and Microsoft. However, not every project necessitates a CLA and there is increasing advocacy for not utilizing a CLA. Since the purpose is to protect IP, for the same reasons in the previous section, contributors may be skeptical of contributing to a project that requires one. Furthermore, the CLA introduces another, yet minor, obstacle for developers to face before they can supply a patch. For example, Node.js removed the CLA in order to lower the barrier to entry and expand the number of developers that are interested in contributing [54]. Bryan Cantrill writes that CLAs can impair the health of a community and, at best, introduce more processes to contributing since employed developers must obtain permission from their legal department [60]. For this reason, illumos, an open source Unix operating system, has never required CLAs, and Joyent did not use them on any of its projects other than Node.js [60].

There are instances where CLAs can be beneficial, predominantly for commercial entities because they protect the company from a contributor claiming IP in their patch. If the project license does not contain a patent grant but one is necessary from contributors, especially if those contributors are from other companies, then a CLA can be beneficial in granting added protection [54]. It may also be the case that lawyers representing the company explicitly want contributors to sign a CLA, even though the license is sufficient for protecting project interests [54]. As with patents, every company has its own goals and use cases for its projects, hence whether the use of a CLA is beneficial is situational. The chief takeaway is that CLAs should not be considered a default and required for all projects because they can deter contributors from participating in a project. Forethought should be given as to why a CLA would be beneficial in a given context.

6.5 Marketing

Just as a company markets its proprietary products to create market share, marketing is also critical for open source projects to expand interest, adoption, and contribution. Since successful open source projects rely on a community, there must exist some strategy for attracting contributors. Deirdré Straughan, an experienced member of the open source community who currently works for AWS's Open Source Blog, defines the role of marketing as competing for already limited attention and resources [61]. According to Straughan, public source code on a platform such as GitHub is the most indispensable form of marketing because it essentially acts as a resume [61]. Potential contributors can determine the state of the project – whether others are contributing, the quality of the code, and more. She also states that every interaction that takes place between project and user is marketing because positive or negative exchanges may influence an individual's drive to participate; as a result, the community should be nurtured [61]. We will discuss more about how this is accomplished in upcoming sections on the community behind successful projects.

Since growing a strong contributor base is crucial for success, media platforms play an influential role. Social media is especially useful for outreach because it amplifies a message with little to no costs [62]. The structure of Twitter is particularly well-fitted because it is easy to use,

has high engagement, and people can follow specific hashtags of interest [62]. At Red Hat, Twitter chats replicating a live event are used to create conversation and survey community opinion [62]. Red Hat's objectives are to strengthen awareness of the project and increase the number of people involved in these discussions [62]. The chats bring attention to the company and project with the intention of increasing interest in use and contribution. Other common marketing practices include project websites, effective domain names, and blogs. While a project need not implement all of these, it should apply those methods deemed most effective for cultivating its presence.

7. Having a Successful Impact

7.1 Attract Diverse Contributors

A project should not just have a large number of contributors, but these contributors should come from different backgrounds in order to provide diverse opinions. As previously mentioned, opening up Apache Phoenix to the community and building a diverse community consisting of more than just Salesforce contributors enabled the project to achieve eminence [11]. One study performed in 2018 sought to explore the relationship between social capital and project success [63]. The researchers used four years of data from SourceForge projects and defined success by the number of contributions [63]. It was found that role diversity, referring to diversity in participants' knowledge and skills, and the success of a project have a positive correlation [63]. If people have the same background and experience, they likely will only reinforce each other's ideas rather than introduce unique viewpoints that can drive innovation within a project. The project will be built with the limited goal of the one company in mind rather than realize its full potential and impact in the greater community. Diverse opinions will expand a project's scope, allowing its reach to expand even further.

7.2 Defining Governance Structures

Successful projects require some form of governance to mediate discussions, ensure that contributions are aligned with goals, and facilitate decision-making. The scope of these roles can vary with some projects having benevolent dictators and others employing a pure community model. The decision of which to adopt depends on the level of control a company wishes to maintain and to what extent power and trust will be given to the community. We will first discuss some of the most common structures that can be found.

One type of model employs complete control over the project with little to no community feedback. This may be because there is a single core team focused on one goal and it is difficult to find contributors that share a similar vision [64]. Or, the project is not attempting to gain contributors and is simply pushing public code to claim that it is open source without participating in the community, which is poor practice [64]. The point of significance is that decisions are made entirely by the project owners without input from external developers and users. This structure may be beneficial for projects that decide tight ownership is critical. However, we will not consider these forms further as we have premised the success of open source projects on collaboration.

A more common governance structure is the benevolent dictator. These projects will have a single person, possibly the founder, deliver the final call regarding major decisions [64]. However, unlike with the previous model, there exists significant community participation which encompasses patches and voicing opinions [64]. This is common in small projects that do not have many maintainers or projects owned by commercial entities [65]. While still benefitting from the diverse opinions of the community, owners are still able to control project direction when needed. The maintainer should ensure that contributors are aware that final decisions are made in this manner. They should also not abuse their power and only use it when necessary since the goal of this model is to facilitate collaboration as much as possible.

One last frequently used governance design is based on meritocracy. Just as in the benevolent dictator formulation, the project relies on heavy input from contributors. Those contributors that are the most active are given prominent roles, such as the ability to directly contribute code without review or even the responsibility of a maintainer [65]. This provides incentive for people to earn their influence. For a company that maintains this class of project, even its own developers must contribute to gain recognition. When large decisions must be made that affect the entire project, resolution is normally achieved through voting [65]. All Apache Software Foundation projects employ such governance [65]. This structure encourages developers to take ownership of their work and creates a community of trust through granting power to those most deserving.

While the benevolent dictator and meritocracy models have contrasting implementations of distribution of power, one theme that both should keep in mind is open governance. This refers to a community in which many participants are collectively and democratically making decisions. While meritocracy inherently employs more open governance than the benevolent dictator, it is still crucial in the latter that the person in this role hear opinions from those involved and only exercise power if necessary. IBM is one such company that stresses using open governance because it secures the long-term success of its projects [66]. There are risks involved with sole ownership, such as maintaining a limited project scope, and open governance removes this threat [67]. Cloud Foundry originated as a project solely owned by VMWare, but for these reasons IBM and several other companies pushed for the creation of the Cloud Foundry Foundation [68]. In addition to reducing risks, open governance has the benefit of naturally constructing trust and openness. Contributors can be sure that a project is not simply being developed in the interest of the one company, but for everyone. Projects that seek external contributions should include at least some trace of open governance.

7.3 Show Community Support

a. Work with the Community

Building a successful impact with a project also implies having a positive impact on the community. Those companies that obtain the most influence provide a consistent number of high-quality contributions to the project [69]. These contributions should not be features that the one commercial entity needs but should instead benefit all parties involved. This is a signal to the community that the company is acting unselfishly and is genuinely invested in the project. The same effect is also achieved through contributing patches upstream rather than maintaining individual forks. First of all, forks are harmful to developers because they accrue technical debt as the maintainer attempts to resolve difference between its own fork and the upstream branch [37]. But perhaps of even more significance is that forks created without the intention of merging in the future deprive the submitter of developing a positive reputation in that community. Especially in systems based on meritocracy, from this reputation spurns influence and ability to make impact.

b. Open Source Foundations and Community-Building Events

Another manner in which companies can support the community is through investing in external open source organizations. There are many such organizations such as the Linux Foundation, Apache Software Foundation, and Eclipse Foundation, each with their own unique goals and projects. For companies with the resources, providing funding is one of the most common means of expressing support. The ASF, for example, has a formal sponsorship program that allocates the funds towards the maintenance involved in its projects [70]. In turn, the foundation grants benefits, with many of them serving to promote the contributor through various marketing tactics [70]. This demonstrates to the open source community that the company is being an exemplary citizen. If a company does not have such means, contributing to the projects maintained by these foundations can be a similar signal of support.

It is also encouraged that firms with the ability provide sponsorship to events that promote open source development such as hackathons, conferences, and summits. Such endeavors serve the interests of the firm because they are excellent for networking, recruiting, and marketing its own projects. For example, Google hosts the Summer of Code every year to introduce students to open source, thereby increasing the community size [71]. Additionally, it is a source of candidates for future full-time positions. Moreover, Google sponsors conferences, especially those related to Internet infrastructure because these projects have perhaps the most outreach and impact [71]. Through participation in such events, companies display their commitment to the health and advancement of the open source community.

8. Community Behind Successful Projects

8.1 One-Time Contributors

One-time contributors (OTCs) are developers that have had one patch accepted to a project and never made such a contribution again. Gaby Getz, a developer at Cesium, states that despite their limited number of contributions on an individual level, these participants play an integral role because they tend to primarily supply bug fixes and, in aggregation, often provide the greatest number of contributions [72]. Naturally, the questions are posed if OTCs possess any other motivations that can be exploited to increase their number of contributions or if there were negative experiences that turned them away. With answers to these questions we can arrive at solutions, and project velocity will demonstrably grow.

A study performed in 2017 found that the majority of OTCs submitted patches to fix bugs that they had encountered during development and did not have any prior motivation or intention to become a long-term contributor [8]. A survey was sent to over 4,000 OTCs to popular open source repositories [8]. The researchers found that almost all motivations were extrinsic or need-based, and the only intrinsic motivation (responsibility) was not one driven by enjoyment of the activity, but rather was perceived more as a duty [8]. This is likely not an area where changes by project maintainers can attract OTCs to submit another patch.

It was also learned that the majority of OTCs held positive opinions of their interactions when contributing, but some poor impressions were caused by lack of responsiveness [8]. While it is understandable that maintainers are busy, an attempt should be made at responding to all contributors in a reasonable timeframe. Getz recommends responding once every 24 hours to let people know where their patch stands [72]. The primary barriers to entry faced by OTCs were challenges surrounding the submission process, such as understanding licensing, and one change that might increase the chances of return would be clarifications to this procedure [8]. Thus, while there exist minor changes that project managers should keep in mind to convert OTCs, these community members should not be the main focus of retention efforts since most only contribute on a need-by-need basis. We now consider those contributors that regularly submit patches.

8.2 Sustainability: Attracting and Keeping Contributors

Since we have success as having a diverse set of contributors, we must note why these contributors may be influenced to participate in certain projects and how to retain their interest. We now contemplate these factors of influence.

a. Meritocracy

We have discussed meritocracy as a governance structure and now consider how the model is ideal for maintaining community health. Building a system in which developers earn their influence is key to motivation to join and sustained participation. Influence can be recognized as a measure of one's value to a project; greater responsibility and power signals that the developer is trusted and integral.

Research has been performed on the role of situational learning, which refers to the social aspects of learning, and identity construction in contributor retention [73]. Situated learning allows contributors to understand what efforts are appreciated and awarded within the community, and thereby construct their identity through engaging in such efforts [73]. It was ascertained that positive situational learning and identity construction interactions can lead to a desire for long-term recognition based on esteem [73]. That is, the ability to have contributions be socially accepted and rewarded acts as a motivator to achieve being highly regarded within the community. The process of situational learning and identity construction recognizes the skills and efforts that developers put into their patches and guarantees that those most fit for the role earn this sought-out status and persist in the community to maintain said status [73]. In summary, we ascertain that for people to be motivated to remain within a community, a system should be instituted that allows them to build their identity through acknowledgement of their talents. This is the system of meritocracy.

b. Transparency

Being transparent about project decisions and goals is essential to successfully attracting and engaging a community. Acknowledging the value of this quality, SAP, a firm that produces business operations software, redesigned its OSPO [74]. One of its changes involved developing an explicit statement regarding work that the company had participated in and how it would be using open source in the future [74]. When contributors are aware of how their efforts will applied in the grand scheme, they can decide if they are interested in the project intentions and whether they wish to participate. Transparency sets expectations and terms of trust upfront so that contributors can validate that a company's objectives are beneficial to everyone.

Netflix has also altered its open source program to emphasize transparency about which projects are archived [75]. This lets developers know that they should not expect any support or company efforts directed toward the project. They can then make an informed decision on whether it is worth their time and effort to maintain a fork themselves or if they should find another solution. It would reflect negatively on the company to falsely claim that an inactive project was still being maintained. Subsequently, others would regard the company as a poor open source citizen and would avoid participating in future interactions with Netflix. Transparency reinforces developers' confidence that the company will not act maliciously with their patches and reduces any miscommunication that may lead to negative perceptions.

c. Remove Barriers of Entry

Barriers of entry pose a threat to new contributors looking to participate but not having the sufficient knowledge or tools to do so. Lowering barriers to entry will play a significant role for those projects seeking to attract new contributors. Perhaps one of the largest barriers is lack of thorough documentation. GitHub's 2017 Open Source Survey found that 93% of respondents experienced incomplete or outdated documentation [26]. Effective documentation is essential because it lowers the learning curve and improves the experience of contributors, increasing the likelihood that they will return [76]. On the other hand, neglected documentation can turn users and contributors away if they decide it is not worth their time to piece together how the source

code behaves [76]. Consequently, companies should make an increased effort to regularly update project documentation.

Documentation also refers to the guidelines that developers use to submit contributions. As part of its efforts to change its open source culture, Adobe received feedback from contributors that the patching process was unclear to the point that many employees were ignoring or avoiding it [77]. There was a lack of centralization since individual teams determined their policies, which mean submitters were responsible for keeping track of each team's unique process [77]. Adobe resolved this through developing a clear checklist of steps to follow in addition to a startup repo that includes all necessary files such as a README and license declaration [77]. The company realized that when feasible, the burden should be taken off contributors as much as possible so they can concentrate their efforts elsewhere. Netflix also faced similar issues for individuals wanting to get started with the company's codebase. Since contributors often found setup and configuration to be tricky, Netflix started packaging its projects in Docker format to minimize the amount of energy needed in this area [75].

There exist other steps that can be taken to improve the beginner contribution experience. To ease the process, Mozilla developed a single contributing guide that aggregates all relevant documents contributors may find useful during the entire patch process [78]. This includes a list of sources for people to find bugs that need fixing, links to the developer guide and review checklist, and steps for determining the best person to ask for a code review [78]. This resolves any ambiguity that new contributors may have regarding the correct procedure to follow and points them to helpful resources when they encounter problems.

d. Acknowledge and Reward

As discussed in the section related to meritocracy, establishing a system of earning influence leads to increased retention. Contributors feel appreciated when their involvement in the community is acknowledged. It has been observed that when people's identities are publicly known, they experience greater benefits from disclosing their innovations, and thus maintainers are incentivized to socially recognize them [79]. There are many other forms of appreciation besides meritocracy that should be utilized to reward significant efforts. Cesium advocates making contributors rock stars through championing accomplishments on media platforms like blogs, forums, and Twitter [47]. These contributors experience positive sentiments of pride and value and are then encouraged to repeat the steps that brought them to this point.

Not everyone can be publicly recognized in such a manner, and it is equally crucial to show appreciation to all those who have invested time and energy in improving the project for the better. Kubernetes Helm, a package for managing Kubernetes applications, follows a subtle yet effective approach. In its internal maintainers guide, the project has a rule stating that at least one positive comment be left on a review when asking for changes [80]. This demonstrates to the submitter that the effort they put into this patch is recognized and appreciated despite any changes that need to be made [80]. While they may have a negative reaction to the requested changes, the effects will be counteracted by the supportive comment. This improves the chance that the developer leaves the project with an overall positive outlook on the contribution process, making it likely that they will submit another patch.

e. Maintain High Project Quality

The state of a codebase is a reflection of the status of a project and how successful maintainers are at managing. A 2017 study concluded that participants associate a codebase of perceived high quality with a greater probability of developing successful software, which in turn makes them more likely to contribute [81]. Additionally, contributors do not have the desire to traverse a chaotic repository to understand the code and determine how to proceed with a patch. If they find it particularly difficult to navigate the codebase, this may deter them from returning in the future. Another piece of research established that codebases that are less monolithic in design and more modular increase contributors' motivations to join and continue involvement [82]. Modularity allows a developer to work independently of others and reduces the complications that may occur with intertwined dependencies. This is especially of significance for open source projects because more often than not, development occurs by people that are physically isolated from one another. A clean codebase reduces the likelihood that developers have negative experiences associated with a contribution, which strengthens the possibility of their return.

f. Be Responsive

As touched upon in the section on OTCs, project owners should be responsive to contributors, and in particular when responding to PR reviews. Mozilla found that contributors who received code reviews within 48 hours of request were much more likely to return and contribute again [83]. If a contributor does not receive a response within a reasonable timeframe, they may become frustrated or feel their efforts are unappreciated. Jean Paoli stresses the idea that the community is a project's friend [14]. This indicates that the project has a responsibility to treat contributors as such and promptly respond to their attempts for engagement. Otherwise, a contributor will be quick to find another project where their efforts are mutually received.

g. Build a Safe Environment

This should be self-explanatory, but it is critical to create an environment where people feel safe from judgement and attack. As Jean Paoli mentions, people can be afraid to make their code public or voice their opinion out of fear of being misjudged or ridiculed [14]. In an environment such as open source that relies on open collaboration, it is necessary to reduce this fear to the greatest extent possible. If a developer feels that a particular community is unsafe or they begin to feel mocked, they will leave for a warmer and less judgmental project. Even if only one person acts hostile, this may lead to the loss of a number of potential contributors due to perceptions of aggression [84]. To take steps toward alleviating this anxiety, projects should include and enforce a code of conduct that defines community etiquette and consequences for violations [84]. Many projects adopt the Contributor Covenant, including Intel because it clearly defines expectations and is representative of best practices in the open source community [85]. Such documentation ensures that contributors feel protected and safe to fully commit to participation.

9. Reasons for Failure

New open source projects are created every day and yet few of them rise to the status of projects like Kubernetes or Visual Studio Code. In this section, we evaluate common mistakes in open source projects alongside case studies that highlight these errors.

9.1 No Transparency or Communication

As discussed in the previous section, transparency by project maintainers is crucial to developing trust and setting forth expectations. This can take a variety of forms, but in all cases the community should be informed regarding any major decisions, and ideally with some notice ahead of time so that those affected can plan ahead. Consider the unexpected shut down of the npm package Kik after facing a battle for naming rights with another project. The developer of kik, Azer, quietly unpublished that project in addition to 272 other packages, including a dependency called left-pad, that led to the failure of thousands of other dependent packages [86]. Luckily, left-pad was open source and a solution was created within hours [86]. However, this created unnecessary panic and confusion that could have been completely avoided had Azer given advanced warning. The issue was not that a solution did not exist, but that developers were not given buffer time to develop a solution to prevent total project failure. This broke the chain of trust between Azer and those dependent on his work, ultimately harming Azer's reputation. It is acceptable and understandable to shut down a project, however this process should be done gracefully with a clear plan that is publicly announced prior to the beginning of the transition [87]. Lack of warning is an indication to users that the maintainers do not care about how their actions impact the community.

In general, it is not sufficient to merely communicate large changes to users and how these changes might affect them. While this may not apply in all scenarios, the community should be included in the decision-making process as much as possible and at least be aware that a discussion is occurring. NetBeans, a Java IDE, was developed by Sun Microsystems before its acquisition by Oracle. Sun had a habit of making impactful decisions at private company meetings [88]. With no input into the discussion, those outside of the company often struggled to understand the motivations for decisions [88]. This meant that it was challenging to blindly accept the decisions as the best path forward for the project [88]. Collaboration in open source is paramount, and it is difficult to collaborate when one side is receiving directions from the other with no opportunity for two-sided discussion. Users should be told what changes are being deliberated so that they may voice their opinions. As previously mentioned, some situations demand that a benevolent dictator make the final decision. In such scenarios, a clear acknowledgement of community opinion with an explanation behind the final decision is sufficient. This creates the back-and-forth dynamic that open source relies on to be successful.

9.2 Lack of Community

We have stated that the existence of a community is critical to success. When a project does not have a sufficient community built around it, the project lacks the resources to support itself in its endeavors. Apache Storm is an event processor initially started and open sourced by

Nathan Marz. Marz had a very clear vision for the project direction, but this left little room for anyone else to provide major contributions and made him a bottleneck in the development process [89]. Since Marz became the center of all development, others perceived him as the single point of failure [89]. Because the project was not robust, this severely slowed down development and innovation. Marz learned that much more could be accomplished by switching to consensus-driven development [89]. Through engaging the community in all capacities and trusting it to fulfill its responsibilities, project velocity will increase significantly the project will grow to its full potential.

While processes may restrict the ability for a community to develop, it is also necessary that these contributors in the community exist in the first place. Heartbleed was a security vulnerability discovered in the OpenSSL cryptography library in 2014 with the major threat of the unencrypted exchange of user data over the Internet. Despite its widespread use in the TLS network communication protocol, OpenSSL is an open source project with contributions from a small group of maintainers and volunteers [90]. We have already discussed that open source software tends to enjoy better security than its proprietary counterpart, on the condition that there is a sufficient number of people examining the code for issues. However, OpenSSL is a project that consists of very view contributors and, at the time of the vulnerability, only one full-time employee [90]. Additionally, resources are strained since the OpenSSL Software Foundation (OSF) has never earned over one million in annual gross revenue [90]. It is evident that when the vulnerability was exploited, OpenSSL lacked the community and therefore the ability to support itself on the scale that it was running. Users and firms especially need to be more diligent about contributing back to projects on which they rely.

9.3 Not Committing to the Culture

Open source does not simply mean that the source code is freely published. Successful projects recognize that there endures an underlying culture that promotes openness and collaboration. One anti-pattern regarding building an open source culture is providing self-serving contributions that exist solely for the firm's interest. Kite, a code editor plug-in for developing in Python, made one such mistake. A minimap is a condensed view of an entire file's code, often displayed on the side of a text editor. The startup hired a developer for atom-minimap, who proceeded to add a feature that displayed Kite links at the bottom of the minimap whenever a Python file was opened [91]. Users were angered by what they perceived to be a marketing scheme and Kite received widespread backlash [91]. After several months, Kite finally agreed to remove the controversial changes [91]. Kite broke the social agreement that open source development exists for the benefit of all those involved and is not a platform for one company to promote itself. It was observable that this selfish act served no use other than as an advertisement for Kite.

This issue also manifests itself in projects with maintainers that make no effort or express no interest in collaborating with the community. App.net was a social networking application of similar design to Twitter. After an initial response of disinterest, the project decided to revamp itself by using open source development [92]. However, it was clear from the start that the company did not care much for building a community around its project. App.net disregarded concerns about barriers to entry rather than addressing them and refused to give flexibility and independence to its developers [92]. For contributors, this was a signal that the company focused

more on the business model and earning a profit rather than supporting its community and allowing the project to grow into its full potential. As a consequence, contributors decided that collaborative opportunities would be better in other projects and that App.net was not worth their efforts. It should be noted that with open source, the motivation to contribute to software does not necessarily change but the motivation to contribute to a specific project does change [93]. This can occur if developers feel that they are not fully appreciated, or they disagree with the project direction. If this manifests itself, they will choose to collectively leave and transfer the learned social ties to another project [93].

9.4 Rigid Requirements and Inflexibility to Risk

We have mentioned that projects considered to be mission-critical generally should not be open sourced. These projects tend to have tunnel vision with exceptionally specific goals and rigid requirements. We expand upon our discussion of the DoD's iEHR. Due to its complexity, an incredible amount of oversight is required for such a project that cannot necessarily be provided by maintainers of open source [36]. This project instead requires a high degree of centralization to ensure that implementation is exactly as specified. iEHR also has considerably less room for errors. While many open source projects develop and mature as the process of collaboration occurs, the nature of iEHR demands that all core features already be implemented, meaning that considerable investment has to take place before a working version is developed. This disincentivizes external contributors because the potential benefits they could reap from the project must be realized far into the future instead of the immediate present. Lastly, iEHR has little tolerance for risk-taking. Being a product of the government, the software must essentially be guaranteed to function as expected with few surprises and bug reports. Furthermore, security must be rigorously tested to prevent confidential data from leaking. These are all features that proprietary software firms can commit to and guarantee. However, while security and bugs are ideally addressed through open source development, innovation tends to be the core focus. Other aspects like bug patches may be a secondary prioritization and development can be more reactive to issues that pop up than a sought-after initiative.

9.5 Open Source to Proprietary

The creator of a new project should give serious consideration as to whether it should be open source or proprietary prior to starting. Switching from one to the other, and especially from open source to proprietary, poses massive risks that might not be surmountable. Compiere provides business solution software. It began with an open source community version of its software and evolved to develop proprietary editions with more features and services. However, this shift to proprietary was not viewed in a positive light by many volunteer developers at the time [93]. The balance between open source and proprietary was broken, eventually causing the community to lose a large portion of its developers [93]. A fork was made and the open source project Adempiere was the result. Changes that introduce privatization such as these introduce concerns that a company will use the open source contributions to build its proprietary product. Contributors, and volunteers in particular, want to ascertain that their efforts are not purely for the profit of the company maintaining the project, since otherwise they could elect to work as an employee instead. Companies should be cautious about the signals, intentional or otherwise, being sent to the community when developing open source and proprietary software in tangent.

9.6 Incorrect License or Usage

An open source license is a reflection of the intentions of the project of which it is in use. We have already discussed how license choice can affect developers' decisions to contribute to certain projects. Occasionally project owners misunderstand these consequences and will choose or create an inappropriate license. One such case involves MongoDB's switch from the GNU AGPLv3 license to their newly formulated Server Side Public License (SSPL) within the last year [94]. The change occurred because the company was exasperated by cloud providers hosting proprietary versions of its source code [94]. The new license articulates that anyone wishing to use the software as a service must obtain a commercial license or open source their code [94].

Consequently, MongoDB faced considerable fallout with some members of the open source community. Debian Linux removed MongoDB from its distribution and Red Hat later announced that it too would drop MongoDB from its Enterprise Linux [95]. Tom Calloway, Red Hat's Technical and Community Outreach Program Manager, stated that the license serves to discriminate against specific users and is an attempt to trigger fear, uncertainty, and doubt in commercial users [95]. It also places restrictions on companies by forcing them to use MongoDB's commercial products [95]. While licenses serve to protect the owner's rights, it is evident that the SSPL abuses this power by explicitly serving proprietary interests. This acts contrary to open source ideals of cooperation toward a common goal. The decision ultimately harmed the project's reputation as a benevolent open source citizen and led to the loss of users. Companies should be wary of the licenses that they employ in their projects and the potential consequences of balancing open source altruism with commercial matters.

10. Future of Open Source

10.1 Industries of Current and Future Potential

We expect adoption of open source to continue to grow, with some companies already adopting a default-to-open-source policy on new projects. While open source is currently most successful in secondary software such as infrastructure and libraries, we anticipate that open source will begin to take a central role in all software development. We now discuss several industries in which open source has substantial prospects.

a. In the Government

Software built for government purposes faces unique challenges regarding pace of development, testing, and a number of other features, as demonstrated through our discussion of iEHR. Yet we frequently learn about a new project that some branch of the government has recently open sourced. In fact, the federal government maintains the website Code.gov which aggregates thousands of open repositories. NASA already sustains a well-established open source program and is currently developing the best strategies for accepting contributions from non-NASA developers [96]. In 2016, the Obama administration unveiled a pilot policy requiring government agencies to open source 20% of the codebase at minimum, excluding those projects that pose legal issues or security threats if publicly released [97]. The change was sought because open sourcing in the government allows inter-departmental collaboration, which increases shared projects and decreases duplication and its associated costs [97]. As we have discussed, this leads to increased innovation as contributors can focus their efforts on novel features in one project rather than continuously reinvent the wheel.

In particular, we consider how open source would be beneficial to the Department of Defense's (DoD's) goals. Security experts recognize that the DoD could benefit from many of the advantages we have touched upon including improved code with fewer bugs, lower cost of ownership, and effective collaboration [98]. Specifically, the DoD could integrate open source methodologies into the development of its databases, middleware, and toolkits, which would allow it to maintain and develop platform software systems more efficiently [98]. Open source would also provide the environment for improved interagency and international collaboration on shared projects [98]. One more source of motivation is improved competition for private technology that interacts with DoD systems because visibility to source code permits firms to add needed features and understand current limitations [98]. If properly implemented, the DoD stands to reap these benefits. While the DoD is already a large consumer of open source, it has difficulties creating and maintaining both its open source and proprietary software [98]. This is due to an inability to utilize various best practices, instead allowing bureaucratic processes to hold back development [98]. This in turn prevents the DoD from taking full advantage of the benefits that open source has to offer.

As we have mentioned, one of the common mistakes made in open source projects is not fully committing to the culture. To address this, it is advised that the DoD's senior leadership team express explicit support for greater open source integration [98]. This enthusiasm will trickle down the leadership chain so that others understand the overarching benefits. We have argued throughout

this thesis that the establishment of a community is critical to open source success because it brings diversity of opinions. If the DoD does not adopt an improved open source mentality, software development and quality will suffer, and these degradations will manifest in the DoD's military systems [98]. As a first step, the DoD must determine projects that can become a joint effort with other governmental agencies and accordingly develop those relationships [98]. To preserve its reputation as a security powerhouse and technological leader, the DoD should adopt not just the legal requirements of open source, but also the mindset that enables projects to evolve into their most innovative forms.

b. Hardware

While this thesis has focused on open source software, the ideology of open development can expand to hardware. Open hardware involves the licensing of hardware designs such as schematics, blueprints, and Computer Aided Design (CAD) files so that they may be publicly released and modified [99]. These modifications can include updating the source code associated with the hardware or changing the hardware itself [99]. One well-known example of open hardware is Arduino, a microcontroller used to load source code to a circuit board. As with open source software, open hardware advocates releasing the designs that are typically privately controlled with the goal of promoting collaboration, advancing innovation, and improving the original project.

Companies realize that open hardware can grant many of the same benefits as open source. In 2011, Facebook partnered with Intel, Rackspace, Goldman Sachs, and Andy Bechtolsheim to create the Open Compute Project [100]. It was founded on the belief that the open sharing of IP and ideas in general maximizes innovation while minimizing complexity [100]. The group's mission is to develop the most efficient designs, specifically for scalable computing [100]. Facebook's interest in open hardware began with the public release of its designs for a new data center, which the company sought to make the world's most energy efficient [100]. Facebook hoped that this would attract others with similar interests to contribute improvements so as to advance these data centers and, by extension, cloud computing.

In addition to innovation, open hardware makes improvements upon security. In the same way that open source is more secure, open hardware is not in itself secure. However, the ability to verify oneself whether there are vulnerabilities in the design gives users the power to take security into their own hands and not be at the mercy of the vendors [101]. One example of a preventable incident was the infiltration of SuperMicro hardware by a Chinese-made microchip [101]. The vulnerability made it possible for outsiders to access the server functions of governmental bodies and some of the country's most prominent firms [101]. Without the source, it was much more difficult to determine whether the hardware had been hacked and how this was possible [101]. If the source code had been available, these organizations could have performed more thorough examination and testing as required for their specific use cases. Had they found that the security was not up to par or discovered a vulnerability, the entity could then make the necessary modifications themselves [101]. Private hardware designs are a security risk in the same manner as proprietary software because users are reliant on the private firms, who more often than not cannot detect and fix all of the vulnerabilities that an entire community could.

c. Research

Research is one more area in which the principle of open sharing and development has propagated, providing tremendous potential for future advancement. Open research, also termed open science, refers to transparency within all steps of the research process, including data, results, and publication [102]. Some of the aims of open research include addressing the crisis of irreproducible research, quickening the dissemination process, and increasing intra- and inter-disciplinary collaboration [102]. A number of organizations have been founded to support the growth of open research including Wellcome Open Research and Gates Open Research. Researchers have observed how open source has launched the software industry forward and they seek the same cooperative environment for their own work.

Although not entirely adopting of open research principles, Google DeepMind, a project dedicated to artificial intelligence (AI), recognizes the power of shared research. In its commitment to advancing the scientific community, DeepMind publishes its research and distributes materials such as open source environments, data sets, and code [103]. The group's objective is to foster a culture of collaboration and allow the entire community to use DeepMind's work as a stepping stone in their own ventures [103]. Intel is also aware of the strength of shared knowledge in research as being the fastest route to discovery [104]. The company believes that shared knowledge leads to the creation of new platforms, discovery of fresh approaches to research, and strengthening of the cooperative ethos associated with open source [104]. As a result, Intel began investing money into shared research projects at the university level [104]. With the purpose of research being to seek out previously unknown knowledge to propel society forward, the introduction of open source principles is of incredible value because it works to grow the innovative potential of a project. The popularization of open research in the future will advance technology at an unprecedented pace.

d. Quantum Computing

As we begin to reach physical and computational limits on our classical machines, quantum computing has come to the forefront to overcome these barriers. Some open source tools for quantum computing have already been released including Circ, a framework developed by Google to work with quantum circuits [105]. Companies such as Google, Microsoft, and IBM have been investing resources to gain influence in quantum computing [105]. As a field that has recently been enjoying increasing popularity and momentum, the community is in a stage of definitive formation and standards are still being developed. This makes open source quantum computing tools an ideal form of development since, as we have discussed, open source drives the creation of standards through collaboration from experienced and knowledgeable participants. These interactions also foster innovation, the production of novel ideas, and the spreading of knowledge to the next generation of computer scientists [106]. Jeff Henshaw, Group Program Manager for a quantum division at Microsoft, believes open source is imperative for developing the tools and programs that will lead to progress in the development of new concepts, algorithms, and applications [105]. By using open source to develop these basic building blocks, researchers and companies can expand their individual efforts and make significantly more progress in the field at a much faster pace.

10.2 Changes to Licensing

a. Relevance of the GPL

We have touched on the historical context for the creation of the GPL. Its purpose is to prevent the mixing of open source and proprietary code. This introduces severe barriers and restrictions for any firm that would like to utilize the software because it requires that the project incorporating it as a dependency also be licensed under the GPL. Otherwise, the firm must find an alternative that has a business-friendly license or build their own solution. Especially with the increasing interest of companies in open source and the innovation accelerations that this has introduced, the debate as to whether the GPL is still relevant today is growing.

Bryan Cantrill holds the opinion that anti-collaborative licensing, embodied by the strong copyleft, is an anti-pattern in open source development [31]. It has been observed that the 50 most watched projects on GitHub have experienced greater decline when licensed under the GPL than business friendly licenses [31]. He states that the GPL is at war with other licenses because the inability to mix open source licensed under the license with proprietary code excludes competitors, leading to barriers to create [31]. While one of the greatest benefits of open source development is faster and more collaborative innovation, the GPL is in direct conflict with this and can actually be harmful to innovation. Especially with the growth in use of open source by private entities, copy-left licenses exclude entire communities. This prevents interested parties from working together to build one master project and creating the collaborative environment that we observe successful projects to maintain. It also severely limits a project's user base, implying that the project can only grow to a certain extent before having difficulty finding users that want and are able to contribute.

Despite all of these drawbacks regarding building a community to encourage innovation, we must recall the original intentions of the GPL. The free software movement was founded on the principles that all source code should be free for everyone to examine and use, and companies break this agreement with proprietary software. If the goal is to build software with this aim in mind, then the use of the GPL is completely valid. However, we must also realize that the community as a whole has changed in the past decade with the free software movement being overtaken by open source. Although there still exist proponents of free software, the goals today have shifted towards driving innovation forward and fostering large communities. While the use of the GPL is justified in principle, project owners must acknowledge that the license places extreme limitations on the user base, which may lead the project to fail because it cannot establish a community.

10.3 Areas for Improvement

a. Gender Diversity

Despite the well-known issue that diversity is struggling within the tech sector, the imbalances are even further exacerbated in the open source community. The GitHub 2017 open source survey found that of the 5,500 randomly selected respondents from 3,800 repositories, 95% were male and a mere 3% identified as female [26]. The survey also determined that while 68% of

women and 73% of men say they are very interested in contributing in the future, only 45% of women say they are very likely to follow through compared to 61% of men. Slightly improved yet similarly disproportionate statistics were found by Digital Ocean's Currents report. Out of 4,340 respondents that were surveyed, 88% identified as male while 11% identified as female [107]. It is evident from this data that the open source community is male dominated. To address this, programs with the purpose of recruiting females into the open source community should be developed. Similar in concept to the Grace Hopper Conference, resources can be invested in woman-centric open source conferences so that female developers can be exposed to greater opportunities for networking and expand their visibility. Outreach programs should also be instituted at beginner levels, like Google's Summer of Code or Code-in, to begin the recruitment process earlier.

There also needs to exist significant investment in welcoming and retaining female developers. The GitHub survey discovered that 25% of women have been in situations that made them unwelcome, 12% experienced stereotyping, and 6% had unsolicited sexual advances while only 15%, 2%, and 3% of males respectively confronted these circumstances [26]. Women also stated that they more often ask for help directly (29% of women versus 13% of men) instead of in public areas like forums [26]. Open source communities must exert a greater effort in making everyone feel accepted and comfortable to seek out help anywhere. We have repeated many times that open collaboration from a diverse set of contributors is a necessity for successful open source projects, and this can only be the case is a safe environment is provided for everyone. This discussion needs to continue to be brought up in the greater technical community.

b. Support for Non-Employees

Accompanied by the increasing number of companies involved in open source is the growth of employees that are paid for their work in such projects. This investment in full-time developers has led to an acceleration in project velocity and innovation for communities. More PRs can be reviewed, bugs tracked and resolved, and questions answered. Yet the involvement of these employees can make those who volunteer in their spare time feel ostracized from the community and underappreciated [108]. The majority of the time this is not intentional and is simply a byproduct of the meritocracy governance in which those who contribute more gain wider recognition [108]. While meritocracy builds a system for motivation, we must also recognize that not everyone has the ability to contribute the same amount and that all contributions should be appreciated.

Nonetheless, sometimes these interactions deliberately put down volunteers. Rich Bowen, a community manager at Red Hat and longtime member of the open source community, has been told by an employed developer that he felt non-fulltime contributors were taking too much credit because they were listed as contributors but did not contribute as much as he did [108]. If this mentality continues to persist, volunteers will be driven out and open source will become a collaboration among firms. This corporate-driven open source will hamper cooperation and innovation among diverse participants with different backgrounds and opinions. Going forward, we must sustain a balance between acknowledging those developers who go above and beyond in the degree to which they contribute because they have the time and rewarding those developers who also provide high-quality contributions but are limited by the time they can invest.

11. Conclusion

We have witnessed how the integration of open source within the software industry has led to a technological revolution that will only continue to grow going forward. Despite the seemingly contradictory nature of publicly releasing IP, both individuals and companies have motivations for electing to collaborate rather than develop in secrecy. These benefits range from pure altruism to building a better product and evolving a company's reputation. It is apparent that those who do not choose to participate in open source are disregarding innovation, efficiency, and development for the communal good.

Not all projects that attempt the open source development model are successful, however, and we have examined factors that contribute to this realization. Clear goals must be laid out from the start and forethought must be given to specific reasons for deciding to develop openly rather than behind closed doors. Entire legal and technological infrastructures must exist to reduce tedious tasks and allow developers to focus their efforts on building the software. Most importantly, projects should recruit and seek out contributions from a diverse set of people. These developers must be adequately supported through a number of initiatives such as rewards, transparency from leadership, and responsiveness by maintainers if the project hopes to maintain these contributions. Above all, the community is the source of power and progress within the project, and as such must be given respect. Of the several reasons for failure that we investigated, the source for many of them was treating the community as secondhand employees rather than peers in a group effort.

We wholeheartedly anticipate that open source adoption will expand in the future. The ideology of open source is not only limited to software, and its concepts have particular relevance to the fields of hardware and research. As the community matures, so too will the development environment and culture. With all of the changes that will occur for the better, we must also remember to maintain the same principles of diversity and collaboration that originally brought the movement success. We hope that more industries of innovation will observe the invaluable benefits that open development introduces and will follow in the footsteps of open source software by embracing this mindset of cooperation as a means to progress.

12. Works Cited

[1] "The Open Source Definition," *The Open Source Definition*, 22-Mar-2007. [Online]. Available: https://opensource.org/osd. [Accessed: 16-Mar-2019].

[2] "Licenses," *Choose a License*. [Online]. Available: https://choosealicense.com/licenses. [Accessed: 16-Mar-2019].

[3] E. A. V. Hippel and G. V. Krogh, "Open Source Software and the Private-Collective Innovation Model: Issues for Organization Science," *Organization Science*, vol. 14, no. 2, pp. 209–223, Apr. 2003.

[4] K. F. Fogel, *Producing open source software: how to run a successful free software project*. Orange Grove Books, 2009.

[5] Masson, "Twenty Years and Counting," *Open Source Initiative*, 22-Dec-2017. [Online]. Available: https://opensource.org/node/909.

[6] C. Hannebauer and V. Gruhn, "Motivation of Newcomers to FLOSS Projects," *Proceedings of the 12th International Symposium on Open Collaboration - OpenSym 16*, Aug. 2016.

[7] J. Coelho, M. T. Valente, L. L. Silva, and A. Hora, "Why we engage in FLOSS: Answers from Core Developers," *Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering - CHASE 18*, Mar. 2018.

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

[9] D. Eckert, S. Koch, and J. Mitlöhner, "Using the Iterated Prisoner's Dilemma for Explaining the Evolution of Cooperation in Open Source Communities" in *First International Conference on Open Source Systems*. *Proceedings*, Genova, Italy, 2005, pp. 186–191.

[10] E. S. Raymond, "The Inverse Commons," in *The Magic Caldron*, 2000. [Online]. Available: http://www.catb.org/~esr/writings/cathedral-bazaar/magic-cauldron/ar01s05.html.

[11] "Salesforce," *TODO: Talk openly, develop openly*. [Online]. Available: https://todogroup.org/guides/casestudies/salesforce/. [Accessed: 17-Mar-2019].

[12] F. Nagle, "Open Source Software and Firm Productivity," *Management Science*, vol. 65, no. 3, pp. 1191–1215, May 2018.

[13] N. Tajitsu, "Toyota uses open-source software in new approach to in-car tech," *Reuters*, Thomson Reuters, 31-May-2017. [Online]. Available: https://www.reuters.com/article/us-toyota-tech/toyota-uses-open-source-software-in-new-approach-to-in-car-tech-idUSKBN18R1CW.

[14] J. Paoli, private communication, Feb. 2019.

[15] A. Bhambhri, "Adobe Takes Open-Source Approach to Experience Data Model," *Adobe Blog*, 24-Sep-2018. [Online]. Available: https://theblog.adobe.com/adobe-takes-open-source-approach-to-experience-data-model/.

[16] E. Campbell, "How Sentry Thrives as an Open Source Software Company," *Sentry Blog*, 14-Feb-2019.

[17] "Red Hat subscription service overview," *Red Hat*. [Online]. Available: https://www.redhat.com/en/resources/how-open-source-subscriptions-deliver-business-value. [Accessed: 18-Mar-2019].

[18] C. Kanaracus, "Oracle submits OpenOffice.org code base to Apache," *InfoWorld*, 01-Jun-2011. [Online]. Available: https://www.infoworld.com/article/2621760/oracle-submits-openoffice-org-code-base-to-apache.html.

[19] "Using Open Source Software to Speed Development and Gain Business Advantage," *The Linux Foundation*, 23-Feb-2017. [Online]. Available: https://www.linuxfoundation.org/blog/2017/02/using-open-source-software-to-speed-development-and-gain-business-advantage/.

[20] J. Reyes and Comcast, "How Comcast does open source," *Technical.ly Philly*, 15-May-2015. [Online]. Available: https://technical.ly/philly/2015/05/15/comcast-open-source/.

[21] "Starting an open source project," *TODO*. [Online]. Available: https://todogroup.org/guides/starting/. [Accessed: 18-Mar-2019].

[22] D. Foster, "Comcast: Open Source Program Success Depends on Business Strategy Alignments," *The Linux Foundation*, 29-Sep-2017. [Online]. Available: https://www.linuxfoundation.org/blog/2017/09/comcast-open-source-program-success-depends-on-business-strategy-alignment/.

[23] "Docs," *Google Open Source*. [Online]. Available: https://opensource.google.com/docs/. [Accessed: 18-Mar-2019].

[24] "Recruiting open source developers," *TODO*. [Online]. Available: https://todogroup.org/guides/recruiting-developers/. [Accessed: 18-Mar-2019].

[25] The Linux Foundation and Dice, "The 2018 Open Source Jobs Report," rep. [Accessed: 19-Mar-2019].

[26] "Open Source Survey," *Open Source Survey*, 2017. [Online]. Available: https://opensourcesurvey.org/2017. [Accessed: 19-Mar-2019].

[27] J.-H. Hoepman and B. Jacobs, “Increased security through open source,” *Communications of the ACM*, vol. 50, no. 1, pp. 79–83, Jan. 2007.

[28] C. Payne, “On the security of open source software,” *Information Systems Journal*, vol. 12, no. 1, pp. 61–78, Feb. 2002.

[29] “6 Reasons Why Open Source Software Lowers Development Costs,” *The Linux Foundation*, 28-Feb-2017. [Online]. Available: https://www.linuxfoundation.org/blog/2017/02/6-reasons-why-open-source-software-lowers-development-costs/.

[30] P. Finch and C. Grams, “Mozilla identifies 10 open source personas: What you need to know,” *Opensource.com*, 11-Sep-2018. [Online]. Available: https://opensource.com/open-organization/18/9/mozilla-open-archetypes.

[31] B. Cantrill, “Corporate Open Source Anti-Patterns: Doing It Wrong,” in *The International Free Software Forum*.

[32] S. Bhartiya, “Guy Martin: Open Source Strategy at Autodesk,” *The Linux Foundation*, 08-Aug-2018. [Online]. Available: https://www.linuxfoundation.org/blog/2018/08/guy-martin-open-source-strategy/.

[33] P. Su, “Twitter's Kafka adoption story,” *Twitter*, 28-Nov-2018. [Online]. Available: https://blog.twitter.com/engineering/en_us/topics/insights/2018/twitters-kafka-adoption-story.html.

[34] A. Joshi, A. Leung, C. Dwyer, F. Kung, S. Dhillon, T. Bak, A. Spyker, and T. Bozarth, “Titus, the Netflix container management platform, is now open source,” *Medium Netflix Technology Blog*, 18-Apr-2018. [Online]. Available: https://medium.com/netflix-techblog/titus-the-netflix-container-management-platform-is-now-open-source-f868c9fb5436.

[35] A. Kulkarni, “Open source and the demise of proprietary software,” *Timescale Blog*, 10-Oct-2018. [Online]. Available: https://blog.timescale.com/open-source-demise-of-proprietary-software-a49f73f54165/.

[36] D. Baum, A. Huff, J. Clingan, and P. Bostrom, “The Department of Defense (DoD) and Open Source Software,” Oracle, rep., Sep. 2003. [Accessed: 20-Mar-2019].

[37] C. Abernathy, C. Aniszczyk, J. Beda, S. Novotny, and G. Yehuda, “Measuring your open source program's success,” *TODO*. [Online]. Available: https://todogroup.org/guides/measuring/. [Accessed: 20-Mar-2019].

[38] D. Neary and D. Bryant, “Red Hat,” *TODO*. [Online]. Available: https://todogroup.org/guides/casestudies/redhat/. [Accessed: 20-Mar-2019].

[39] “Tips for Evaluating a Company's Open Source Culture,” *The Linux Foundation*, 05-Oct-2016. [Online]. Available: https://www.linuxfoundation.org/blog/2016/10/tips-for-evaluating-a-companys-open-source-culture/.

[40] “Open source community,” *IBM Developer*. [Online]. Available: https://developer.ibm.com/open/community/. [Accessed: 20-Mar-2019].

[41] C. Aniszczyk, J. McAffer, W. Norris, and A. Spyker, “How to create an open source program,” *TODO*. [Online]. Available: https://todogroup.org/guides/create-program/. [Accessed: 21-Mar-2019].

[42] “Oath,” *TODO*. [Online]. Available: https://todogroup.org/guides/casestudies/oath/. [Accessed: 21-Mar-2019].

[43] “Enterprise Roles in Open Source Compliance,” *The Linux Foundation*, 03-Jan-2018. [Online]. Available: https://www.linuxfoundation.org/blog/2018/01/enterprise-roles-open-source-compliance/.

[44] “Comcast,” *TODO*. [Online]. Available: https://todogroup.org/guides/casestudies/comcast/. [Accessed: 21-Mar-2019].

[45] “Facebook,” *TODO*. [Online]. Available: https://todogroup.org/guides/casestudies/facebook/. [Accessed: 21-Mar-2019].

[46] “What is the Apache Software Foundation?,” *How the ASF works*. [Online]. Available: https://www.apache.org/foundation/how-it-works.html. [Accessed: 21-Mar-2019].

[47] S. Chow, “The Road to 200,000 Downloads: The Cesium Story,” in *FOSS4G NA 2018*. Available: https://cesium.com/presentations/files/foss4g2018_schow.pdf. [Accessed: 21-Mar-2019].

[48] “What is version control,” *Atlassian Git Tutorial*. [Online]. Available: https://www.atlassian.com/git/tutorials/what-is-version-control/. [Accessed 22-Mar-2019].

[49] G. Getz, “Cesium Continuous Integration,” *Cesium*, 07-Apr-2016. [Online]. Available: https://cesium.com/blog/2016/04/07/cesium-continuous-integration/.

[50] C. Rossi, “Rapid release at massive scale,” *Facebook Code*, 31-Aug-2017. [Online]. Available: https://code.fb.com/web/rapid-release-at-massive-scale/.

[51] C. Aniszczyk and J. McAffer, “Tools for managing open source programs,” *TODO*. [Online]. Available: https://todogroup.org/guides/management-tools/. [Accessed: 22-Mar-2019].

[52] “The open source program at Microsoft: How open source thrives,” *TODO*. [Online]. Available: https://todogroup.org/guides/casestudies/microsoft/. [Accessed: 22-Mar-2019].

[53] J. Marcey, E. Nakagawa, and C. Delahousse, “Open source: 2018 Year in review,” *Facebook Code*, 02-Jan-2019. [Online]. Available: https://code.fb.com/open-source/open-source-2018/.

[54] “The Legal Side of Open Source,” *Open Source Guides*. [Online]. Available: https://opensource.guide/legal/. [Accessed: 23-Mar-2019].

[55] R. DeCausemaker, “Head of Open Source at Facebook opens up,” *Opensource.com*, 22-Oct-2014. [Online]. Available: https://opensource.com/business/14/10/head-of-open-source-facebook-oscon.

[56] K. J. Stewart, A. P. Ammeter, and L. M. Maruping, “Impacts of License Choice and Organizational Sponsorship on User Interest and Development Activity in Open Source Software Projects,” *Information Systems Research*, vol. 17, no. 2, pp. 126–144, Jun. 2006.

[57] “5 Tips on Enterprise Open Source Success From Capital One, Google, and Walmart,” *The Linux Foundation*, 16-Feb-2017. [Online]. Available: https://www.linuxfoundation.org/blog/2017/02/5-tips-on-enterprise-open-source-success-from-capital-one-google-and-walmart/.

[58] C. Osborne, “Facebook relicenses React in the face of open-source dev backlash,” *ZDNet*, 25-Sep-2017. [Online]. Available: https://www.zdnet.com/article/facebook-relicenses-react-in-the-face-of-open-source-dev-backlash/.

[59] “BSD Patent,” *Open Source Initiative*. [Online]. Available: https://opensource.org/licenses/BSDplusPatent. [Accessed: 23-Mar-2019].

[60] B. Cantrill, “Broadening Node.js Contributions,” *Triton | Joyent*, 11-Jun-2014. [Online]. Available: https://www.joyent.com/blog/broadening-node-js-contributions.

[61] P. Baker, “How to Market an Open Source Project,” *The Linux Foundation*, 18-Dec-2017. [Online]. Available: https://www.linuxfoundation.org/blog/2017/12/marketing-open-source-project/.

[62] “Growing Your Open Source Community With Twitter,” *The Linux Foundation*, 13-Feb-2017. [Online]. Available: https://www.linuxfoundation.org/blog/2017/02/growing-your-open-source-community-with-twitter/.

[63] S. Daniel, V. Midha, A. Bhattacherhjee, and S. P. Singh, “Sourcing knowledge in open source software projects: The impacts of internal and external social capital on project success,” *The Journal of Strategic Information Systems*, vol. 27, no. 3, pp. 237–256, Sep. 2018.

[64] “Open Source Archetypes: A Framework for Purposeful Open Source,” The Mozilla Foundation, rep., May 2018. [Online]. Available: https://blog.mozilla.org/wp-content/uploads/2018/05/MZOTS_OS_Archetypes_report_ext_scr.pdf. [Accessed: 24-Mar-2019].

[65] “Leadership and Governance,” *Open Source Guides*. [Online]. Available: https://opensource.guide/leadership-and-governance/. [Accessed: 24-Mar-2019].

[66] “Become an open enterprise,” *IBM Developer*. [Online]. Available: https://developer.ibm.com/open/culture/. [Accessed: 24-Mar-2019].

[67] “Open source community,” *IBM Developer*. [Online]. Available: https://developer.ibm.com/open/community/. [Accessed: 24-Mar-2019].

[68] M. Maximilien, “The history of IBM’s contributions to Cloud Foundry, part 2,” *IBM Developer*, 04-Mar-2019. [Online]. Available: https://developer.ibm.com/blogs/history-cloud-foundry-2/.

[69] “Improve your open source development impact,” *TODO*. [Online]. Available: https://todogroup.org/guides/impact/. [Accessed: 25-Mar-2019].

[70] “The Apache Software Foundation Sponsorship Program,” *The Apache Software Foundation*. [Online]. Available: https://www.apache.org/foundation/sponsorship.html. [Accessed: 25-Mar-2019].

[71] “Community,” *Google Open Source*. [Online]. Available: https://opensource.google.com/community/. [Accessed: 25-Mar-2019].

[72] G. Getz, private communication, Feb. 2019.

[73] Y. Fang and D. Neufeld, “Understanding Sustained Participation in Open Source Software Projects,” *Journal of Management Information Systems*, vol. 25, no. 4, pp. 9–50, 2009.

[74] “SAP,” *TODO*. [Online]. Available: https://todogroup.org/guides/casestudies/sap/. [Accessed: 26-Mar-2019].

[75] A. Spyker and R. Meshenberg, “Evolution of Open Source at Netflix,” *Medium Netflix Tech Blog*, 28-Oct-2015. [Online]. Available: https://medium.com/netflix-techblog/evolution-of-open-source-at-netflix-d05c1c788429.

[76] J. Davies, “Building great open source documentation,” *Google Open Source*, 15-Oct-2018. [Online]. Available: https://opensource.googleblog.com/2018/10/building-great-open-source-documentation.html.

[77] J. Gray, “How Adobe is Changing Its Culture Around Open Source,” *Medium Adobe Tech Blog*, 12-Nov-2018. [Online]. Available: https://medium.com/adobetech/how-adobe-is-changing-its-culture-around-open-source-c9c7daa6dc2a.

[78] “Contributing to the Mozilla code base,” *MDN Web Docs*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Developer_guide/Introduction. [Accessed: 26-Mar-2019].

[79] E. A. V. Hippel and G. V. Krogh, “Open Source Software and the Private-Collective Innovation Model: Issues for Organization Science,” *SSRN Electronic Journal*, pp. 209–223, Apr. 2003.

[80] “Lessons Learned from Growing an Open Source Project Too Fast,” *The Linux Foundation*, 15-Mar-2018. [Online]. Available: https://www.linuxfoundation.org/blog/2018/03/lessons-learned-from-growing-an-open-source-project-too-fast/.

[81] S. Y. Ho and A. Rai, “Continued Voluntary Participation Intention in Firm-Participating Open Source Software Projects,” *Information Systems Research*, vol. 28, no. 3, pp. 603–625, Jun. 2017.

[82] C. Y. Baldwin and K. B. Clark, “The Architecture of Participation: Does Code Architecture Mitigate Free Riding in the Open Source Development Model?,” *Management Science*, vol. 52, no. 7, pp. 1116–1127, Jul. 2006.

[83] “Building Welcoming Communities,” *Open Source Guides*. [Online]. Available: https://opensource.guide/building-community/. [Accessed: 27-Mar-2019].

[84] “Your Code of Conduct,” *Open Source Guides*. [Online]. Available: https://opensource.guide/code-of-conduct/. [Accessed: 27-Mar-2019].

[85] I. Sousou, “Intel Adopts Contributor Code of Conduct for Open Source Projects,” *Intel Open Source Technology Center*, 01-Nov-2018. [Online]. Available: https://01.org/blogs/2018/intel-covenant-code.

[86] I. Schlueter, “kik, left-pad, and npm,” *The npm Blog*, 23-Mar-2016. [Online]. Available: https://blog.npmjs.org/post/141577284765/kik-left-pad-and-npm.

[87] C. Abernathy, C. Aniszczyk, G. Martin, and D. Wheeler, “Shutting down an open source project,” *TODO*. [Online]. Available: https://todogroup.org/guides/shutting-down/. [Accessed: 28-Mar-2019].

[88] C. Jensen, “Collaboration, leadership, control, and conflict negotiation in the Netbeans.org community,” *"Collaboration, Conflict and Control: The 4th Workshop on Open Source Software Engineering" W8S Workshop - 26th International Conference on Software Engineering*, Jan. 2004.

[89] N. Marz, “History of Apache Storm and lessons learned,” *thoughts from the red planet*, 06-Oct-2014. [Online]. Available: http://nathanmarz.com/blog/history-of-apache-storm-and-lessons-learned.html.

[90] C. Warren, “Heartbleed Exposes a Problem With Open Source, But It's Not What You Think,” *Mashable*, Mashable, 14-Apr-2014. [Online]. Available: https://mashable.com/2014/04/14/heartbleed-open-source/#VHre5l4QXiqd.

[91] K. Collins, “Programmers outraged after coding startup Kite infiltrated open-source projects to market its products,” *Quartz*, 07-Aug-2017. [Online]. Available: https://qz.com/1043614/this-startup-learned-the-hard-way-that-you-do-not-piss-off-open-source-programmers/.

[92] S. Phipps, “App.nets open source failure,” *InfoWorld*, 12-May-2014. [Online]. Available: https://www.infoworld.com/article/2608256/app-net-s-open-source-failure.html.

[93] D. Ehls, “Open Source Project Collapse - Sources and Patterns of Failure,” *Proceedings of the 50th Hawaii International Conference on System Sciences (2017)*, pp. 5327–5336, 2017.

[94] F. Lardinois, “MongoDB switches up its open-source license,” *TechCrunch*, TechCrunch, 16-Oct-2018. [Online]. Available: https://techcrunch.com/2018/10/16/mongodb-switches-up-its-open-source-license/.

[95] S. Vaughan-Nichols, “MongoDB ‘open-source’ Server Side Public License rejected,” *ZDNet*, ZDNet, 16-Jan-2019. [Online]. Available: https://www.zdnet.com/article/mongodb-open-source-server-side-public-license-rejected/.

[96] “Share Your Code,” *NASA Open Source Software*. [Online]. Available: https://code.nasa.gov/#/share. [Accessed: 17-Apr-2019].

[97] M. Rockwell, “White House releases open source policy,” *Federal Computer Week*, FCW, 08-Aug-2016. [Online]. Available: https://fcw.com/articles/2016/08/08/open-source-rockwell.aspx.

[98] B. FitzGerald, J. Parziale, and P. Levin, “Open Source Software and the Department of Defense,” Center for New American Security, rep., Aug. 2016. [Accessed: Mar-29-2019].

[99] “What is open hardware?,” *Opensource.com*. [Online]. Available: https://opensource.com/resources/what-open-hardware.

[100] “About,” *Open Compute Project*. [Online]. Available: https://www.opencompute.org/about. [Accessed: Mar-29-2019].

[101] J. Pearce, “How open source hardware increases security,” *Opensource.com*, 31-Oct-2018. [Online]. Available: https://opensource.com/article/18/10/cybersecurity-demands-rapid-switch-open-source-hardware.

[102] D. Kingsley, “Open Research,” *Open Research*, 11-Oct-2016. [Online]. Available: https://osc.cam.ac.uk/open-research. [Accessed: Mar-29-2019].

[103] “Open Source,” *DeepMind*. [Online]. Available: https://deepmind.com/research/open-source/. [Accessed: Mar-29-2019].

[104] “Seeding Future Technology Breakthroughs: Open Source Inspires Collaboration in Academia,” Intel, brief, 2013. [Online]. Available:

https://software.intel.com/sites/default/files/Open_Source_in_Academic_Research_Tech_Brief.pdf. [Accessed: Mar-29-2019].

[105] R. Bhatia, "Launching Open Source Initiatives Is the Next Battleground In Quantum Computing," *Analytics India Magazine*, Analytics India Magazine, 23-Jul-2018. [Online]. Available: https://www.analyticsindiamag.com/launching-open-source-initiatives-is-the-next-battleground-in-quantum-computing/.

[106] G. Pacchioni, "The rise of open source in quantum physics research," *On your wavelength*, 09-Jan-2019. [Online]. Available: http://blogs.nature.com/onyourwavelength/2019/01/09/the-rise-of-open-source-in-quantum-physics-research/.

[107] "Currents - A Seasonal Report on Developer Trends in the Cloud: Open Source Edition," Digital Ocean, rep., Oct. 2018. [Online]. Available: https://www.digitalocean.com/assets/media/currents-research/pdf/DigitalOcean-Currents-Q3-2018.pdf. [Accessed: Mar-29-2019].

[108] R. Bowen, "Success at Apache: For Love or Money: Volunteer vs. Professional Open Source," *The Apache Software Foundation Blog*, 05-Feb-2019. [Online]. Available: https://blogs.apache.org/foundation/entry/success-at-apache-for-love.