A Case Study of Motivations for Corporate Contribution to FOSS

Iftekhar Ahmed, Darren Forrest, Carlos Jensen
School of EECS
Oregon State University
Corvallis, OR, USA
ahmedi@oregonstate.edu, darrenforrest@gmail.com, Carlos.Jensen@oregonstate.edu

***Abstract*— Free/Open Source Software developers come from a myriad of different backgrounds, and are driven to contribute to projects for a variety of different reasons, including compensation from corporations or foundations. Motivation can have a dramatic impact on how and what contribution an individual makes, as well as how tenacious they are. These contributions may align with the needs of the developer, the community, the organization funding the developer, or all of the above. Understanding how corporate sponsorship affects the social dynamics and evolution of Free/Open Source code and community is critical to fostering healthy communities. We present a case study of corporations contributing to the Linux Kernel. We find that corporate contributors contribute more code, but are less likely to participate in non-coding activities. This knowledge will help project leaders to better understand the dynamics of sponsorship, and help to steer resources.**

Keywords*— *Mining Software Repositories; Bug Reports; Commit Messages, Mailing list; FOSS; Corporate Participation

I. INTRODUCTION

Free/Open Source Software (FOSS) is a key component of our current computing ecosystem. FOSS projects are critical to the Internet and other key infrastructure, as well as systems used by corporations around the world. FOSS not only provides free or low cost products and services to meet critical needs, it serves as a vendor-neutral way for the IT community to solve common problems and develop and share standards. FOSS is now also commonly found on personal computers; browsers like Mozilla's Firefox, productivity software like Open Office, and operating systems like Ubuntu Linux. The success and growth of the FOSS movement is therefore important to the overall well-being of the computing ecosystem.

While FOSS development is similar to closed-source development in many ways, there are some key differences. In traditional software, access to the source code is limited to the development team, treated as a trade secret. In FOSS, access is freely available to all, and projects encourage open participation. This means that anyone with the desire and skill necessary can get involved with and contribute to the project. The resulting fluid boundaries of who is on the development team makes the community structure, the development process, and the project management process different than in more traditional projects.

Key to most FOSS projects is the meritocratic system, where contributions and contributors are judged by what they contribute to the project, and thus awarded influence over the project and engineering decisions. According to De Souza, project leaders hold significant power, deciding which submissions are included and rejected [9]. "Because Apache is a meritocracy, even though all mailing list subscribers can express an opinion by voting, their action may be ignored unless they are recognized as serious contributors" [26]. The way to become, or be accepted as a project leader or serious contributor, is in most cases determined by the amount of code the person has previously contributed [22].

In spite of this leadership structure, FOSS is often described as a grassroots movement. Developers are driven by altruistic reasons [5], but studies have shown that this is not always the case [17]. Compensation is fairly common among FOSS developers, at least for the larger projects [22]. Project sponsors sometimes pay developers directly for their work on FOSS projects, while other developers receive funding through foundations or other non-profit organization [15]. Some developers also receive compensation in the form of equipment or time off work. These and other forms of compensation mean many FOSS projects are supported by a complicated barter economy.

This FOSS economy is by no means a bad thing. Large projects are complicated to run, and usually need more formalized processes and support. However, mixing volunteer and paid labor may change the dynamics of projects, especially when rank or influence within the project is determined by the amount of code contributed. Furthermore, it is safe to assume paid developers are predisposed to address the needs of those that pay them [5]. This is not necessarily detrimental to the project; the sponsors' priorities will likely overlap with the priorities of the community at large, but maybe not all the time.

The purpose of our work is not to malign or downplay the good corporations do for FOSS projects. Corporations contribute 80% of the code to the Linux kernel [6]. Many FOSS projects depend on corporate participants, especially larger projects. Our goal is to determine how corporations are participating and how that impacts the project as a whole. This knowledge will allow project leaders to better understand the dynamics of sponsorship, and how to steer development and foster trust.

978-1-5386-0443-4/17/$31.00 ©2017 IEEE

Forrest et al. began to examine the impact of corporate participation by looking at the Linux Kernel and GCC projects [13]. They examined bug and code repositories to determine the affiliations of contributors, and identified some interesting trends. They found that corporations had vastly larger numbers of developers contributing code than they did participating in bug triaging. The lack of participation in bug repositories (as reporters or fixers) was seen as a lack of coordination about priorities with the rest of the community. From this Forrest et al. concluded that corporations focused on their own needs before those of the project. This again is not necessarily bad for the project; their needs were not orthogonal to the project, but rather points to a lack of coordination and communication.

Forrest et al. did not look at mailing lists or do a qualitative analysis of the contributions made. It is therefore not clear what percentage of contributions were fixes to bugs not publicly reported, something which unambiguously would be aligned with project priorities, versus adding new features which might not. This paper builds on the work of Forrest et al. through a more detailed and nuanced view of how corporations involve themselves in FOSS projects. More specifically we seek to investigate the following:

Assumption1: Sponsored contributors are more prolific code developers than unaffiliated contributors

Shown to be true by Forester et al., this ties directly to the meritocratic model. We will adjust our sampling of projects to try and pick a mix of more or less biased entities.

RQ1: Do corporate contributors show a preference for code development over other contributions or interactions?

At stake here is to what level contributors coordinate their actions with the project as a whole, either through discussions, bug reports, or other planning. The assumption is that the less discussion there is, the less likely it is that code contributed is in line with the projects priorities.

RQ2: When corporate contributors contribute code, do they skew toward new features (potentially selfish needs) or bug fixes (unambiguously in the communities interest)?

For security or PR reasons, it may not be wise to publicly discuss bugs, but we can assume that all bug fixes are in the communities best interest. New features or code however potentially increase maintenance costs and affect performance.

Corporations are going to engage with developers in different ways and have different expectations and policies. They should therefore not be painted as a monolithic group. However, we would argue that organizations that overwhelmingly focus on code contributions, who do not engage the community in discussion and governance, and who almost exclusively focuses on adding new features is a less desirable partner than one which does engage and communicate with the community, and that does address the problems facing the community at large.

The outline of this paper is as follows. We discuss related work in the FOSS space and development techniques. Next, we describe our research approach, followed by results and discussion. Finally, we outline future work and conclusions.

II. Related Work

FOSS projects have been well studied over the past decade, and researchers have put forth theories about community development [7], software development processes [33], and even the quality of the software itself [16].

A. FOSS Governance & Contributor Motivations

FOSS community structures have been also studied and several models have been proposed. Perhaps the best known is the onion model [8, 22, 35]. This model describes the roles and role transitions, with the more specialized and authoritative roles occupied by more senior or high-status members, and individuals progressing through these roles from the outer, less important roles to the inner, more important ones. Jergensen et al. expanded on this by looking at how developers migrate between projects, and proposed an onion patch model [24]. This model accounts for how experienced developers are able to move more quickly or skip through the ranks because of their experience and reputation with other projects. Still others have suggested that the onion model is only accurate for a handful of projects and should not be considered true of FOSS in general [8].

Compensation is just one of the reasons developers give for participating in FOSS [27]. While the findings varied widely, Hars and Ou claim that 16% of FOSS developers are directly paid for FOSS work with an additional 34% considering FOSS development part of their job expectations [17]. Core developers had higher rates of compensation according to Jensen and Scacchi [22]. Herraiz et al. found that developers who are paid to contribute to FOSS also experience a "sudden integration" process, effectively skipping levels, while volunteers more closely follow the onion model [19].

Bonaccorsi and Rossi surveyed corporate leaders and found that firms and individuals tended to focus on technical and economic reasons for participating in FOSS, while individual developers claimed social and personal reasons [5]. Ye and Kishida found community membership, a desire to learn, and reputation important to individuals [35].

Further work by Nguyen found that compensation was correlated with efficiency in FOSS projects [10]. Nguyen studied bug resolution time and found that paid developers were able to resolve more issues faster than their non-paid counterparts. There is however no proof of causality (do corporations tend to hire the most effective developers, or are developers more effective, or able to spend more time, when they have corporate sponsors, or some combination).

Conflict is a common phenomenon in groups, especially when differing motivations or values are involved. The distributed work style of FOSS projects also has its own set of challenges [11]. Elliot and Scacchi found that the social ties within a development community are key to mitigating conflict [12]. Stewart and Gosain identified and studied core FOSS values and how they affect the success of FOSS projects [34]. They found that affective trust was one of the main drivers for success.

B. FOSS Project Coordination

FOSS development is often distributed, facilitated through mainly text-based communication, the logs of which have been the source of numerous studies. Some of the major sources are bug reports, code repositories, and mailing lists.

Bug reports are vital to FOSS projects. The need for open bug reporting and wide participation is captured in Linus' law, which states "many eyes make all bugs shallow" [30]. Bug reports are one of the ways end users interact with the development team [32], though this can lead to noisy data. Bettenburg identified features of good report and provided tools to help users provide better reports [4]. While bug reports are almost always a positive for FOSS development, Ko and Chilana found that bug reporting by power users could have a chilling effect [25]. Even redundant reporting can be positive; some developers report prioritizing based on the number of duplicate bug reports [3].

Revision Control systems (RCs) are used by projects to manage their code base, and have also been extensively studied. German focused on ways to visualize code changes [14], while others have used it to study things like developer turnover and role identification [31]. De Souza et al. found that the structure of the code mimicked that of the development team [9]. More importantly for our work, they found that control over the code was managed through RCs, and that core development teams used these tools to enforce decisions about direction and community structure.

Mailing lists are used by most FOSS projects as the primary means for discussion and making community decisions [28]. Mailing lists are used to broadcast messages to all members. Through moderation and social protocols, the mailing list can also be used to control the behavior of participants [9]. Conflicts in mailing lists were studied by Jensen et al. [23], and Bergquist and Ljungberg [2]. For the Linux kernel, the Linux Kernel Mailing List (LKML) is the primary forum of discussion and decision-making [20].

III. Methodology

Because we want to compare and contrast our results with the work of Forrest et al [13], our methodology closely mirrors theirs. We chose to study the Linux kernel 2.6. While the work of Forrest et al. [13] was based solely on data from bug and code repositories, we provide a more complete picture of corporate participation by incorporating mailing list logs.

A. Sampling

We used a multiple case study methodology, looking at the eight most prominent corporate participants, as measured by number of code contributors. Our selection includes both hardware and software companies. For the purposes of this study we limited our investigation to contributors who had email addresses that indicated their affiliation (e.g: @companyname.com).

Hardware companies were expected to add drivers and support for their products. Such development requires deep knowledge of the hardware, and possibly access to proprietary information. Drawing form Forrest et al. findings we selected two companies skewed toward code submission, Fujitsu and Samsung, and two with a more balanced approach, AMD and Intel. We followed a similar selection process for software companies, picking Oracle and Google for code heavy participation and Redhat and IBM for a more balanced approach.

We used this selection criteria for 2 reasons: we wanted to compare our findings with the findings of Forrest et al., and we also wanted make sure that the companies selected would represent different types of behavior. The companies studied authored 17.9% of all code commits to the Linux Kernel during this period, and employed 12.2% of all code authors. 7.7% of bugs had contributions by these companies while only 2.2% of contributors to bug reports were affiliated with these companies. The 2012 Linux Foundation report shows that these eight corporations are now even more prominent code developers, accounting for 30.1% of commits [6]. This sample is therefore very influential and an important focus of study. As a control, we studied contributors using gmail accounts, who were more likely, but not guaranteed to be independent developers.

B. Data Collection

We collected data for the Linux Kernel 2.6.34 from November 6th, 2002 to July 29th, 2010, so we could directly compare our results with those of Forrest et al. To answer our research questions, we gathered 95% of bug reports for the Kernel. The remaining were inaccessible either due to permissions or repository errors. Our data included the email address of each reporter, assignee, and commenter. Data for code submissions was similarly gathered. We downloaded the Linux kernel source and used git log information to obtain author names, email addresses and commits.

We gathered mailing list data from the University of Indiana Linux Kernel Mailing List (LKML) mirror site[1]. We used this mirror because it allowed grouping messages by authorship. Since LKML sites do not include email addresses, matching code, bug, and mailing list contributors required matching real names to email addresses. We successfully mapped 98% of email addresses in the bug repository to a name in the LKML.

With code submissions, we had to deal with several complications. The most important was that 40% of email addresses were obfuscated in some manner. Examples took the form of: CD45F355109A9B@domain.com, which appeared at first to be commit hashes. Further investigation revealed that these were reference addresses related to code submissions. We chose to exclude these addresses from our dataset because they would not be tied to any one individual. After removing these addresses, we were able to match 98.5% of email addresses with real names. Based on real names, 115 individuals appear to use multiple addresses or have switched providers over the 8 years of our data. These addresses were consolidated to single entities except when they involved multiple corporations.

[1] http://lkml.indiana.edu/hypermail/linux/kernel/index.html

C. Code Classification

In order to answer RQ2 we needed to have some understanding of the purpose or class of code commits. Code commits can be broadly categorized into one of three categories: bug fixes and improvements (modifying existing code), new features or functionality (adding new code), and "other" – commits that didn't belong to either of the previous categories (documentation, test code etc.). Two key problems are that it is not always trivial to determine what category a commit falls under, and that the Linux kernel sees a huge amount of activity. Manual classification was therefore not an option, and we decided to use machine learning techniques for this purpose.

As a first step to this we manually labeled commits made in response to a specific or implied bug report as bug fixes. Some keywords indicating bug fixes were "Fix", "Bug", and "Resolves" along with their derivatives. Improvements were manually identified based on the following keywords: "Cleanup", "Optimize", and "Simplify" or their derivatives. Commits were tagged as New Features by the keywords: "Add", "Introduce". Also, the number of lines modified was compared with the lines added. Those commits with more lines added than modified were considered more likely to be associated with new features. Anything that did not fit into this pattern we marked as "Other". 10.3% of the commits studied fell into this latter category.

1) Training the Commit Classifier

To generate the training data for the machine learning classifiers we randomly selected 800 commits from the 8 corporations (100 from each corporation) as representative of the work they did for the Kernel. We also randomly selected 300 commits from the control group as training data for the machine learning classifiers.

Two evaluators worked independently to classify the commits. Their datasets had a 33% overlap, which we used to calculate the inter-rater reliability. This gave us a Cohen's Kappa of 0.49. Our dataset has two qualities that make a lower kappa acceptable according to Bakeman et al., which are few codes and codes that are not equiprobable [1]. In simulations, assuming equiprobable codes, raters with 85% agreement would have 0.49 and 0.60 kappa values for code sets of size 2 and 3 respectively. In our training dataset, the distribution of Bug fix was 65%, new feature was 25% and 10% of the commits belonged to the other category.

2) Commit Classification

We used the manually categorized data set as our training data for the machine learning classifier. We identified the unique words in manually classified code commits and used Porter's stemming algorithm [29] for suffix stripping and removed all the words belonging to the standard stop word list[2]. We ended up with two different vocabularies; one for the control group and one for our test group. The vocabulary from the corporate contributors was reasonably similar across all corporations, but significantly different from that of the control group. We measured the similarity subjectively by looking at the comments associated with contributions. We found that corporate and unaffiliated developers would use different terminologies to describe a bug fix, what issues said code fixed, and the level of details given. We therefore decided to treat the two sets separately.

We used the vocabulary data as training data for classification of the entire set of commits for the selected companies and control group. We used a decision tree (J.48) and a naïve Bayes classifier (NB). We used 10-fold cross validation while training the classifier. We used the Weka [18] platform, and Table I has quality indicators.

TABLE I. CLASSIFIER MODEL'S QUALITY MEASUREMENTS

Classifier	Precision	Recall	ROC Area	F-Value
J.48 (Company)	0.658	0.655	0.713	0.657
J.48 (Control)	0.589	0.633	0.664	0.587
NB (Company)	0.587	0.510	0.639	0.528
NB (Control)	0.604	0.617	0.763	0.607

Our goal was to achieve high precision and recall, so we used the F-value to measure and compare the performance of the models. F-value considers precision and recall by taking their harmonic mean. The J.48 classifier out-performed the NB for the company case, and performed close to the NB in the case of the control group. We also used the area under the ROC curve to evaluate the performance of the classifiers. ROC represents how well the test separates the categories and our goal was to achieve the best possible accuracy in distinguishing between bug fixes and improvements, new features or functionality, and "other" category. The J.48 classifier out-performed the NB for the company case using these criteria as well.

IV. RESULTS

We started by re-examining the key findings of Forrest et al., as one criticism of their finding was that they used the number of contributors in their analysis rather than the number of contributions [13]. We found that while the number of contributors may not be directly interchangeable with the number of contributions, they do follow the same trends for each of the corporations studied, as shown in tables II and III.

TABLE II. CONTRIBUTORS AND CONTRIBUTIONS (CODE)

Company	Contributions	Contributors	Avg
intel.com	42,854	256	167.4
fujitsu.com	6,272	58	108.1
amd.com	3,360	43	78.1
samsung.com	1,200	34	35.3
oracle.com	14,942	35	426.9
redhat.com	48,052	213	225.6
ibm.com	29,567	346	85.5
google.com	2,316	83	27.9
gmail.com	49,693	1,621	30.4

[2] http://dev.mysql.com/doc/refman/5.1/en/fulltext-stopwords.html

TABLE III. CONTRIBUTORS AND CONTRIBUTIONS (BUG)

Company	Contributions	Contributors	Avg
intel.com	1,378	47	29.3
fujitsu.com	13	10	1.3
amd.com	37	9	4.1
samsung.com	1	1	1
oracle.com	35	6	5.8
redhat.com	174	31	5.6
ibm.com	866	112	7.7
google.com	7	5	1.4
gmail.com	3,816	1,509	2.5

A. *Answering RQ1*

Our first research question was whether corporate contributors show a preference for code development over other contributions or interactions (e.g bug contribution and mailing list participation).

1) Corporate Participation in Bug Contribution

First, we checked whether the control group (gmail) and other groups had equal mean in terms of bug contribution, and found strong evidence against this (Welch two-sample t-test, t=-3.7552, df =227.119, p=0.0002). We also checked whether software and hardware companies had equal means in terms bug contribution. This also turned out to be statistically significant (Welch two-sample t-test, t=2.0909, df =73.639, p=0.0399).

2) Corporate Participation in Mailing List

We also gathered statistics on mailing list participation, adding a new dimension to the study of this community. We looked at both the number of participants and the number of posts they made to the list (see Table IV).

TABLE IV. CONTRIBUTORS AND CONTRIBUTIONS (LKML)

Company	Contributions	Contributors	Avg
intel.com	18,570	61	304.4
fujitsu.com	1,086	16	67.87
amd.com	3,189	11	289.9
samsung.com	56	3	18.7
oracle.com	11,804	17	694.35
redhat.com	40,285	116	347.28
ibm.com	22,611	173	130.7
google.com	13,239	35	378.25
gmail.com	12,362	472	26.2

The control group (gmail) showed a statistically significant difference in mean mailing list contribution compared to the other groups (Welch two-sample t-test, t=-4.6009, df=460.314, p=5.447e-06). We also checked whether software and hardware companies had equal means and found no difference (Welch two-sample t-test, t=-0.1621, df=213.754, p=0.87).

3) Corporate Participation in Code Contribution

Our numbers are generally lower than those reported by Forrest et al. because we removed obfuscated email addresses. We re-ran our numbers without removing these emails to confirm this. Despite the discrepancy, the same general skew is still seen in favor of code contributions by corporate contributors (see Table V).

TABLE V. RATIO OF CODE CONTRIBUTORS TO BUG CONTRIBUTORS

Company	Forrest et al	Our Data
intel.com	12.1	5.4
fujitsu.com	53.1	5.8
amd.com	14.4	4. 8
samsung.com	88	34
oracle.com	13	5.8
redhat.com	12.8	6. 9
ibm.com	6.3	3.1
google.com	45.6	16.6
gmail.com	1.6	1.1

Next, we looked at overlap in participation. In an ideal world, we expect coders to participate in bug reporting and triage (addressing community needs), and mailing list discussions (project governance). From Figure 1 we see this is not the case. The majority of corporate developers only contribute code. A good number of corporate developers' code and participate on the mailing list, but a minority in bug reporting or triage. Because we did not have email addresses from the LKML, we do not know the number of participants who only participate on the mailing list.

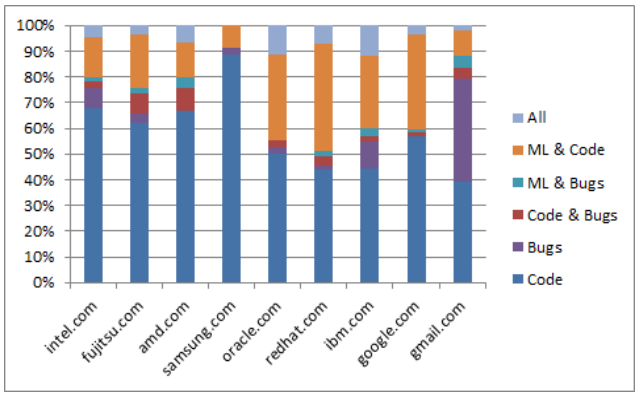

Fig. 1. Distribution of contributors between bug, code and mailing list repositories. (Without LKML only data due to missing contributor affiliations.)

B. *Answering RQ2*

Having seen the skew in favor of code and the magnitude of corporate involvement, we turn our attention to examining what types of code contributions these developers make. We classified the 198,256 contributions made (see Table II for breakdown), and show the results in Figure 2.

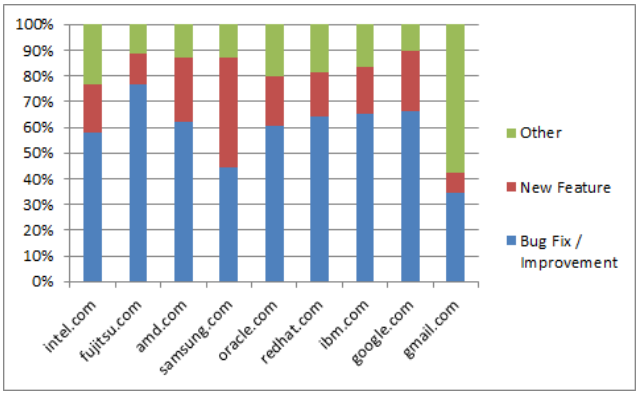

Fig. 2. Contribution type by company

In Figure 2 we see that a majority of code commits were classified as improvements or bug fixes. There were two outliers here, Samsung and the gmail control group, the first skewed in the direction of new features and functionality, and the second in the direction of "other". The median contributions of bug fix/improvement between software and hardware companies were not statistically significant (Wilcoxon rank-sum test with continuity correction, w=99789.5, p= 0.9638). There is no evidence that the number of new feature contribution is different between software and hardware companies (Wilcoxon rank sum test with continuity correction, w=113628.5, p= 0.05636)

Apart from the 2 research questions we wanted to understand the types of commits performed by corporate participants going into the "Other" category. We manually classified 126 randomly selected commits for this purpose (see Table VI for breakdown). These numbers are only meant as guides rather than being representative for all "Other" code contributions.

TABLE VI. BREAKDOWN OF "OTHER" CATEGORY

Category	Freq.
Embed discussion/design	32.5%
Change configuration	28.6%
Merge branches	10.3%
Move functions	7.9%
Add/update documentation file	6.3%
Add/delete comments	3.2%
Rename function/variables	3.2%
Prepare for future changes	1.6%
Remove Dead code	1.6%
Remove debugging code	1.6%
Remove unused variables	1.6%
Data type change/add	0.8%
Remove orphaned Email	0.8%

We also wanted to examine the makeup of "core" contributors, a very nebulous term. We decided to set a bar of 50 accepted contributions, and found that 216 contributors met this criterion, 8% of the population in our sample. Next, we looked at their affiliation, and found that around 75% of these are associated with one of the corporation in our sample (Table VII). Given that corporate participants made up 45% of developers in our sample, we find that corporate developers are overrepresented among core developers.

TABLE VII. DISTRIBUTION OF CORPORATIONS WITHIN THE HIGH FREQUENCY CONTRIBUTORS

Company	Freq.
intel.com	23.61%
fujitsu.com	4.17%
amd.com	1.85%
samsung.com	0.93%
oracle.com	5.56%
redhat.com	18.98%
ibm.com	18.52%
google.com	0.93%
gmail.com	25.46%

V. DISCUSSION

This study focuses on corporate participation in the Linux Kernel, but more broadly tries to explore corporate participation behavior in any open source project. Open source development is a significant focal area for many technology companies, either directly (they depend on or use open source software themselves), or indirectly (the practices and tools pioneered in the production of open source are being more broadly adopted as standard practices by all developers). Understanding the challenges of working with others, and how to provide more effective governance over such multi-corporation efforts is therefore crucial to fostering more such efforts in the future.

A. The State of Corporate Participation in the Linux Kernel

Looking at the basic data we see confirmation of the fact that corporations do contribute a lot to the Linux Kernel. Second, that these developers are in most cases contributing more code per person than our control group (gmail users), and therefore have more influence over the project. What isn't clear is whether these people are effective because they receive sponsorship, or whether corporations hire or sponsor people who have proven themselves effective. The truth probably lies somewhere in-between.

We found many of the same trends identified by Forrest et al. [13] including that many corporations focus on code more than other forms of participation. One interesting finding is that while almost half of developers with software companies were active on the mailing list, less than a quarter of developers from hardware companies were similarly engaged. Fewer than 10% of corporate developers participated fully in the community (bug reporting, code development, and community discussions).

Contrary to the findings of Forrest et al., we found the participation of Fujitsu no longer as heavily skewed towards code. Forrest et al found a ratio of over 50:1 for code developers to bug repository participants [13]. After accounting for the obfuscated email addresses, we found a ratio of 5:1; much more in line with the rest.

We were sensitive to the idea that individual corporate developers might hide behind group aliases for things like reporting bugs (for instance using bugs@corp.com). In looking through the data we found no evidence of such practices.

We found references in corporate contributions to other bug repositories, most likely company internal. There are a number of reasons a company might want to keep bugs hidden, including security and PR. Both of these reasons are legitimate in a private project, but touch potential nerves in an open project like the Linux Kernel. Not sharing known bugs could lead to delays in addressing issues, or even suboptimal fixes. Perhaps a different approach should be explored, with an open bug repository and a sensitive repository with a more limited membership.

B. The Impact of Corporations on The Linux Kernel

Turning to our research questions, we do see clear evidence that corporate developers are more prolific than unaffiliated

developers (gmail). It would have been interesting to see if corporate developers were more successful (% acceptance) than unaffiliated developers, but the data on rejected patches is not available.

In response to RQ1 we found that there is a clear bias towards coding, with approximately 50% of corporate contributors never participating in any other way. There was some nuance here though. IBM for instance had a relatively high level of participation in the bug repository, and almost half of the developers from software companies also participated in the mailing list. This is important, because participation in discussions and other forms of governance and community life ensures coordination and builds trust.

The low participation in bug reporting and triaging was surprising. The bug repository serves as an indicator for community involvement; it tracks the level of coordination and collaboration around shared priorities and problems. While this initially paints a dim picture of corporate participation, the analysis summarized in Figure 2 adds more nuance, and we find that the majority of coding done by corporate developers (in terms of commit count) is focused on fixing (unreported) bugs and improving code rather than injecting new functionality. With one exception, our companies have fairly consistent contribution ratios: 60%-70% of code contributions are bug fixes or improvements, and 10-20% of their contributions are new features.

In this paper, we look at code not tied to bug reports, community discussions, or bug fixes and optimizations as potentially "selfish." This does not imply worthlessness to the community, but rather that it potentially "jumped the line" in terms of community priority. This behavior is not unique to corporate developers; a lot of individuals contribute to "scratch their own itch". That said, corporations on the whole appear to be good, if somewhat distant citizens of the Linux Kernel.

One can hardly fault a corporation for acting in its own best interest by following a self-focused development philosophy. However, if a corporation forges ahead without coordinating with the community, any contributions they make become "take-it or leave it" propositions. The benefit to the company is clear, but the benefit to the project may be questionable. The fact that changes are accepted into the Kernel shows that the project leadership considers them to at least have some value. If data on rejected code were available, it would be interesting to see how often code is rejected due to a mismatch of priorities, potentially a risk for corporations that do not engage in broader community discussions and governance.

C. *Broader picture*

Based on what we have found we posit that corporate involvement is strongly beneficial overall for the Linux Kernel community. The eight companies studied here alone contribute a large amount of code, and over 75% of all kernel development is done by developers who are being paid for their work [6]. Without this support, it is unimaginable that the project would be as robust or feature rich. That said, corporate participation is subtly changing the culture, with less bugs being shared, and more work being done in isolation. Project leadership needs to be aware of this, and make sure that these contributions align with the needs and desires of the community as a whole. Given what previous research has shown [10], the lack of communication could, intentionally or unintentionally, lead to the "hijacking" of a project.

In the case of the Linux community, there are several large corporations that can keep each other in check, and a vigilant management team, but other projects may not be as organized. In the case of the Linux community, the greater danger would likely be that corporations as a group overrule smaller developers and hobbyists, leading to a Linux only suited for the server room for instance. For smaller projects, or projects with fewer dominant players, we advocate that project leadership take an active role in monitoring and communicating with corporate participants to make sure they remain aligned with the interests of the community, and thus help build transparency and trust.

Many of these inequalities or skews could be successfully addressed through social protocols imposed by community leaders. Setting up a limited-access bug repository for sensitive bugs might address some of the reluctance to share bugs. Requiring all patches and contributions to reference a mailing list discussion or bug report, or subjecting unsolicited patches and features to go through an extended vetting process would likely be effective forcing functions to promote wider participation. Though this might have a chilling effect on code contributions, it is unlikely to affect corporate sponsored developers much, as they are externally motivated to contribute.

VI. THREATS TO VALIDITY

With any study of socio-technical artifacts there are limitations. The date range chosen for the study was large. In this nearly eight-year period, the priorities and activities of the project and the companies studied may have shifted. While this may have had an effect on the results, we chose these companies in part because they are substantial and long-term contributors to the kernel. We also wished to obtain a sense of long-term trends. In this way, we consider the benefits of the long timeframe to outweigh the potential drawbacks in the data gathered.

Generic email addresses that we could not map to real names were removed from our data set. This included any addresses from email providers such as yahoo.com, hotmail.com or yandex.ru. While there is substantial anecdotal evidence of sponsored developers using generic emails, without thoroughly investigating each contributor, connecting a generic email address to a specific individual and sponsor would be nearly impossible.

The use of gmail accounts as a control group has difficulties of its own. gmail did not become available until 2004 and we chose to stick with the time period used by Forrest et. al.. To compensate, we mapped individuals who used multiple addresses or switched providers over the 8 years of our study to single entities, except when they involved multiple corporations. We know corporate-sponsored contributors use generic email addresses, including gmail, but were not able to systematically identify and remove them. These individuals chose to contribute from an address that

would not tie them to a corporation. Thus, this may have skewed our control, but is likely a minor factor, is unavoidable limitation to our method, and would in any case skew the data to our disadvantage. It is also possible that some members of the community have participated only in the LKML. We were unable to identify these individuals. While this was not unfortunate, the lack of data available made a mapping impossible.

Some individuals changed employers during the timeframe studied. In these cases, we found the corresponding code commits or bug repository contributions that bounded the change in employer and assigned the average of the two dates as the cut-over date. This may not be accurate, but it is an unavoidable limitation. Since there were only 15 such cases, this did not have a significant impact on our results.

The removal of obfuscated email addresses affected some corporations more than others. While all but Samsung had some obfuscated email addresses, some saw up to a 71% reduction in the number of email addresses. While this may have impacted the totals, it is important to note that all of these addresses refer to a single commit. Additionally, these are not tied to a single individual; they are references to LKML messages to assist in understanding the chain of decision-making. For this reason, we felt removing them would not impact our analysis.

While we did classify the commits of each corporation we did not attempt to judge the value of commits or bug reports. Our data is limited to just a number of commits and our classification. We have also given the same weight to any individual whose name appears on a commit or bug. While this is overly simplistic, it does provide a starting point and without a system for rating these commits and activities it would be difficult to remain objective.

While classifying the commits, we used machine learning classifier to determine whether a commit was a bug-fix. Since our analysis relies on relative count of bug fixes, as long as we do not systematically undercount bug fixes, our results are valid.

VII. Conclusion

In this study, we were able to replicate and confirm the findings presented by Forrest et al. [13]. By adding data on mailing list participation, and an analysis of the code contributions, we were able to significantly extend the analysis they did and document several new dynamics and behaviors.

We found that corporate contributors do contribute more code to the Linux kernel than unaffiliated developers, which agrees with the findings of Corbet et al. We also found that corporate contributors are less likely to participate in non-coding facets of the project, such as bug reporting and triaging, or discussions on the mailing list. This finding agrees with Forrest et al. However, after looking more closely at the data and removing non-personal accounts we found that the disparity is not as large as reported.

When activity within the LKML is considered, most corporations do have a presence in community discussions. However, this presence is smaller than the number of individuals contributing to code development. Based on this we take a more cautious stance, but still affirm that corporations are being strategic in where they assign their resources.

In studying the types of code contributions made by corporations we find that bug fixes or improvements dominate. This is an encouraging sign, in that a bug fix or optimization benefits the whole community. However, we do not know whether the bug fixes and improvements are biased towards code originally contributed by the same corporation, or whether they address general issues.

More study is needed to determine how corporate sponsorship impacts FOSS communities, thought the overall effect seems to be to significantly help improve the project. Understanding the social dynamics is important for maintaining the openness and trust within the community, especially with non-sponsored contributors.

Acknowledgment

We would like to thank the Oregon State University HCI group for their input and feedback on the research.

References

[1] Bakeman, R., McArthur, D., Quera, V., & Robinson, B. F. (1997). “Detecting sequential patterns and determining their reliability with fallible observers.” *Psychological Methods*, *2*(4), (pp.357-370).

[2] Bergquist, M., & Ljungberg, J. (2001). “The power of gifts: organizing social relationships in open source communities”. *Information Systems Journal*, *11*(4), (pp.305-320).

[3] Bettenburg, N., Premraj, R., Zimmermann, T., & Kim, S. (2008). “Duplicate bug reports considered harmful… really?.” In *International Conference on Software Maintenance, ICSM 2008. IEEE* (pp. 337-345).

[4] Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., & Zimmermann, T. (2008). “What makes a good bug report?”. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering* (pp. 308-318).

[5] Bonaccorsi, A., & Rossi, C. (2003). “Why open source software can succeed”. *Research policy*, *32*(7), (pp.1243-1258).

[6] Corbet, J., Kroah-Hartman, G. & McPherson,A. (2015) “Linux Kernel Development: How Fast It Is Going, Who Is Doing It, What They Are Doing, and Who Is Sponsoring It.” *The Linux Foundation. Available at: https://www.linux.com/publications/linux-kernel-development-how-fast-it-going-who-doing-it-what-they-are-doing-and-who.*

[7] Crowston, K., & B. Scozzi.(2012) “Open Source Software Projects as Virtual Organisations: Competency Rallying for Software Development.” *Software, IEEE Proceedings* - 149.1 (2002): 3–17.

[8] Crowston, K., & Howison, J. “The Social Structure of Open Source Software Development Teams.” *First Monday, 10(2), February 2005.* .

[9] De Souza, C., Froehlich, J., & Dourish, P. (2005). “Seeking the source: software source code as a social and technical artifact.” In *Proceedings of the 2005 international ACM SIGGROUP conference on Supporting Group Work* (pp. 197-206).

[10] Duc, A. N., Cruzes, D. S., Ayala, C., & Conradi, R. (2011). “Impact of stakeholder type and collaboration on issue resolution time in OSS Projects.” In *Open Source Systems: Grounding Research* (pp. 1-16). Springer Berlin Heidelberg.

[11] Easterbrook, S. M., Beck, E. E., Goodlet, J. S., Plowman, L., Sharples, M., & Wood, C. C. (1993). “A survey of empirical studies of conflict”. In *CSCW: Cooperation or Conflict?* (pp. 1-68). Springer London.

[12] Elliott, M. S.,& Scacchi,W.(2003) “Free Software Developers as an Occupational Community: Resolving Conflicts and Fostering

Collaboration." *Proceedings of the 2003 International ACM SIGGROUP Conference on Supporting Group Work*. (pp.21–30).

[13] Forrest, D., Jensen, C., Mohan, N., & Davidson, J. (2012). "Exploring the Role of Outside Organizations in Free / Open Source Software Projects." In *Open Source Systems: Long-Term Sustainability*. Springer Berlin Heidelberg. (pp. 201-215).

[14] German, D. M. (2006). "An Empirical Study of Fine-grained Software Modifications." *Empirical Software Engineering* 11. (pp. 369–393).

[15] Ghosh, Rishab A. et al. *Free/Libre and Open Source Software: Survey and Study*. The Netherlands: International Institute of Infonomics University of Maastricht, 2002.

[16] Gyimothy, T., R. Ferenc, & I. Siket. (2005) "Empirical Validation of Object-oriented Metrics on Open Source Software for Fault Prediction." *IEEE Transactions on Software Engineering* 31.10. (pp. 897–910).

[17] Hars, A., & S. Ou.(2001) "Working for Free? - Motivations of Participating in Open Source Projects." *Hawaii International Conference on System Sciences*. Vol. 7. Los Alamitos, CA, USA: IEEE Computer Society. 7014.

[18] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). "The WEKA data mining software: an update." *ACM SIGKDD Explorations Newsletter*, *11*(1), (pp.10-18).

[19] Herraiz, I., Robles, G., Amor, J. J., Romera, T., & González Barahona, J. M. (2006). "The Processes of Joining in Global Distributed Software Projects." In *Proceedings of the 2006 International Workshop on Global Software Development for the Practitioner*. New York, NY, USA. (pp. 27–33).

[20] Hertel, G., Sven, N., & Stefanie, H. (2003). "Motivation of Software Developers in Open Source Projects: An Internet-based Survey of Contributors to the Linux Kernel." *Research Policy* 32.7. (pp.1159–1177).

[21] Hofstede, G., Hofstede, G. J., & Minkov, M. (1991). *Cultures and organizations: Software of the mind* (Vol. 2). London: McGraw-Hill.

[22] Jensen, C., & W. Scacchi.(2007). "Role Migration and Advancement Processes in OSSD Projects: A Comparative Case Study." *29th International Conference on Software Engineering. (*pp. 364–374).

[23] Jensen, C., Scott, K., & Victor, K. (2011). "Joining Free/Open Source Software Communities: An Analysis of Newbies' First Interactions on Project Mailing Lists." *Proceedings of the 2011 44th Hawaii International Conference on System Sciences*. Washington, DC, USA: IEEE Computer Society. (pp. 1–10).

[24] Jergensen, C., Sarma, A., & Wagstrom, P. (2011) "The Onion Patch: Migration in Open Source Ecosystems." *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*. New York, NY, USA. (pp. 70–80).

[25] Ko, A. J., & Chilana, P. K. (2010). "How Power Users Help and Hinder Open Bug Reporting." *Proceedings of the 28th International Conference on Human Factors in Computing Systems*. New York, NY, USA. (pp. 1665–1674).

[26] Kogut, B., & Anca M. (2001) "Open-Source Software Development and Distributed Innovation." *Oxford Review of Economic Policy* 17.2. (pp. 248 –264).

[27] Lakhani, K., & Wolf, R. (2003). "Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects*"*. Rochester, NY: Social Science Research Network.

[28] Mockus, A., Fielding, R. T., & Herbsleb, J. D. (2002). "Two case studies of open source software development: Apache and Mozilla." In *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(3). (pp. 309-346).

[29] Porter, M. F. (1980) "An algorithm for suffix stripping." In *Program: electronic library and information systems* 14.3. (pp.130-137).

[30] Raymond, E. (1999). "The cathedral and the bazaar". Knowledge, Technology & Policy, 12(3). (pp.23-49).

[31] Robles, G., & Gonzalez-Barahona, J. M. (2006). "Contributor turnover in libre software projects". In *Open Source Systems* (pp. 273-286). Springer US.

[32] Sandusky, R. J., & Gasser, L. (2005). "Negotiation and the Coordination of Information and Activity in Distributed Software Problem Management." *Proceedings of the 2005 International ACM SIGGROUP Conference on Supporting Group Work*. New York, NY, USA. (pp. 187-196).

[33] Scacchi, W. (2002). "Understanding the Requirements for Developing Open Source Software Systems." *Software, IEE Proceedings* - 149.1. (pp. 24–39).

[34] Stewart, K. J., & Gosain,S.(2006) "The Impact of Ideology on Effectiveness in Open Source Software Development Teams." *MIS Quarterly* 30.2. (pp. 291–314).

[35] Ye, Y., & Kishida, K. (2003). "Toward an understanding of the motivation of open source software developers." In *25th International Conference on Software Engineering, 2003*. Proceedings. (pp. 419-429).