On the Scalability of Linux Kernel Maintainers' Work

Minghui Zhou
Peking University; Key Laboratory of High Confidence Software Technologies, Ministry of Education
Beijing, China 100871
zhmh@pku.edu.cn

Qingying Chen
Peking University; Key Laboratory of High Confidence Software Technologies, Ministry of Education
Beijing, China 100871
qychen@pku.edu.cn

Audris Mockus
University of Tennesee
Knoxville, TN, USA 37996-2250
audris@utk.edu

Fengguang Wu
Intel Opensource Technology Center
Shanghai, China 200241
fengguang.wu@intel.com

ABSTRACT

Open source software ecosystems evolve ways to balance the workload among groups of participants ranging from core groups to peripheral groups. As ecosystems grow, it is not clear whether the mechanisms that previously made them work will continue to be relevant or whether new mechanisms will need to evolve. The impact of failure for critical ecosystems such as Linux is enormous, yet the understanding of why they function and are effective is limited. We, therefore, aim to understand how the Linux kernel sustains its growth, how to characterize the workload of maintainers, and whether or not the existing mechanisms are scalable. We quantify maintainers' work through the files that are maintained, and the change activity and the numbers of contributors in those files. We find systematic differences among modules; these differences are stable over time, which suggests that certain architectural features, commercial interests, or module-specific practices lead to distinct sustainable equilibria. We find that most of the modules have not grown appreciably over the last decade; most growth has been absorbed by a few modules. We also find that the effort per maintainer does not increase, even though the community has hypothesized that required effort might increase. However, the distribution of work among maintainers is highly unbalanced, suggesting that a few maintainers may experience increasing workload. We find that the practice of assigning multiple maintainers to a file yields only a power of 1/2 increase in productivity. We expect that our proposed framework to quantify maintainer practices will help clarify the factors that allow rapidly growing ecosystems to be sustainable.

CCS CONCEPTS

• **Software and its engineering → Software evolution**; **Maintaining software**; **Open source model**; **Programming teams**;

KEYWORDS

Maintainer's workload, maintainer scalability, work distribution, software evolution, Linux kernel, open source ecosystem

ACM Reference format:
Minghui Zhou, Qingying Chen, Audris Mockus, and Fengguang Wu. 2017. On the Scalability of Linux Kernel Maintainers' Work. In *Proceedings of 2017 11th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Paderborn, Germany, September 4–8, 2017 (ESEC/FSE'17),* 11 pages.
https://doi.org/10.1145/3106237.3106287

1 INTRODUCTION

Free/libre open source (FLOSS) ecosystems, in particular large ecosystems such as the Linux kernel, OpenStack, Docker, or Android, represent critical computing infrastructure for our society. Such ecosystems involve contributions from various participants who are distributed over the world, and are diverse in skills, interests, and needs. Typically, a small core group does most of the work and coordinates a much larger group of peripheral participants [8, 23, 33]. The authority of this core group arises naturally as a result of their contributions to the project [3, 20], such as writing code themselves or reviewing patches contributed by others. The Linux kernel has a fairly sophisticated hierarchical organization within this core group of contributors referred to as maintainers. "Subsystem maintainers are responsible for collecting the accepted patches, do[ing] a final review[,] and submit[ting] them to Linus Torvalds and [the] main maintainer."[1][2] Understanding how such core groups function can shed light on Linux and other large ecosystems. In particular, we would like to develop ways to measure the work maintainers do. Even though a substantial body of literature has characterized developers' work [1, 10, 13, 23, 36, 39], what maintainers do has not been addressed. Maintainers are not regular developers; unlike core groups in smaller projects some maintainers do not write code. Consequently it is not clear how to quantify the work of Linux kernel maintainers (RQ1).

"Maintainers are like editors in the publishing industry"[1] with burdensome responsibilities, such as 487 patches in a two-week period [16] and hundreds of emails a day [27]. The situation may deteriorate because a popular project often accumulates a rapidly expanding code base and contributor community. As project grows,

[1]events.linuxfoundation.org/sites/events/files/slides/collab_linux_kernel_v2.pdf
[2]https://www.linux.com/blog/role-linux-kernel-maintainer

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ESEC/FSE'17, September 4–8, 2017, Paderborn, Germany
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-5105-8/17/09…$15.00
https://doi.org/10.1145/3106237.3106287

its ability to sustain itself may come into question. Alarms over the number of major features that will need to be integrated in the future and the concern about the potential of the current maintainer population's ability to scale with the Linux kernel have been raised.[3] This risk is exacerbated by the fact that almost everyone and every organization in the world relies on the Linux kernel. These concerns lead to more research questions. How fast does the Linux ecosystem grow (RQ2)? Has the maintainers' workload increased as the ecosystem has expanded (RQ3)? How is the work distributed among maintainers (RQ4)? In particular, does the 80/20 rule (80% of work is done by 20% of participants [23]) apply within the core groups of Linux? If so, the Linux ecosystem exhibits self-similarity and the same phenomenon is repeated at different scales [28].

Finally, we inquire: how well does the Linux ecosystem scale (RQ5)? More specifically, to what degree does adding development resources to a large project decrease productivity and cause diminishing returns to overall output [6].

According to its creator, the key to the success of the Linux is its modularity [31]. Inside the system, the modules are arranged in a structured hierarchy of dependence relations, but modules at the same level can be developed independently from each other [3]. We, therefore, assume that the coping mechanisms that make ecosystem effective allow for adaptations that both are caused by and affect software architecture (represented by module structure), change over time, and react to commercial involvement strategies. Observing differences among Linux kernel modules and the changes that occur over time may lead to insights regarding what mechanisms may be at play.

To investigate our research questions we obtained the maintainer history for each file of the Linux kernel and quantified each maintainer's workload by the number of files under maintenance, the numbers of commits in the maintained files, and the number of authors the maintainer was obligated to deal with. We find that the amount of work had stabilized for the core modules but continued to grow in the periphery. The number of maintainers grows faster than the amount of work, thus the providing evidence against the hypothesized risk that maintainers would be overwhelmed by work. The distribution of work among maintainers showed that a few maintainers accomplish most of the work for most modules, confirming this aspect of self-similarity. Finally, the investigation of scalability showed that the practice of assigning multiple maintainers to a file yields only a power of 1/2 increase in productivity (e.g., four parallel maintainers are needed to double the overall output).

In the remainder of the paper Section 2 presents the methodology we used, Section 3 summarizes our findings, Section 5 outlines limitations, Section 6 discusses related work. The conclusions are presented in Section 7.

2 METHODOLOGY

We applied a mixed-method approach [7] to understand and quantify how the maintainers of the Linux kernel, the core of the Linux operating system, work. We used qualitative methods to understand maintainer behavior and to design suitable measures for maintainers' work (RQ1). Maintainer activity data were then used to quantify the growth of the system (RQ2), the growth of maintainers' workloads (RQ3), the work distribution among maintainers (RQ4), and the scalability of maintainers' work (RQ5).

2.1 Qualitative Investigation

The Linux kernel has been extensively studied in the past; we compiled further evidence from recorded artifacts in the version control system (Git), project web pages, and relevant websites. We interviewed maintainers to address questions with no coverage in existing sources.

More specifically, we searched for and read digital records, and communicated with Linux maintainers to understand project context and practices. We designed metrics and validated results by combining and triangulating information from disparate sources. In particular, we went through the following procedure: 1) Read the existing literature, particularly on the Linux kernel, e.g., [2, 14, 17, 18, 20, 21, 31], to understand the project context and the studied practices. 2) Inspect Linux web sites looking for project-related information, such as the standard development process and the role of maintainers. Examine various blogs, forums, webzines (like lwn.net), and news websites (like linux.slashdot.org). Search for relevant information, such as the practices of maintainers in different subsystems and the trajectories of known maintainers. 3) Target four maintainers and conduct interviews to understand how they do their work and the details of maintainer mechanisms that are difficult to obtain from artifacts, such as whether the tasks and contributors in different modules vary from each other and how they vary. Also interview maintainers to validate the findings.

Based on the information we gathered from various sources, the following quote pithily summarized the maintainers' work: "Being a maintainer means you read patches from submitters, handle questions from both developers and users about things related to the subsystem (usually bug reports). If a patch looks acceptable, you test it if possible, and apply it to the relevant git tree and push it publicly, and notify the author that it was accepted. Every weekday, these git trees get merged together in the linux-next release, and inevitably, problems are reported and it is up to the maintainer to fix them when they affect their portion of the kernel."[4] It suggests three measurable quantities that should affect maintainer effort.

1. The more files a maintainer oversees, the more time and effort she will need to devote. Each file under maintenance may need to be considered when fixing a bug even if it is ultimately not changed.
2. More commits in the maintained files imply more effort for a maintainer (all other things being equal). In particular, a maintainer is likely to review and approve changes made to the files she maintains. She might also make the changes herself.
3. A maintainer's effort is likely to increase with the number of authors contributing to the maintained files. A need for increased maintainer effort may be caused by "too many cooks spoil the broth" — effect or the need to learn the different personalities of contributors and their contribution styles.

We, therefore, use the number of files maintained, the number of commits committed to the maintained files, and the number of authors in these commits during a time period (month) to characterize the amount of work for a maintainer. This answers RQ1.

[3]https://lwn.net/Articles/703005/

[4]http://www.kroah.com/log/linux/what_greg_does.html

2.2 Quantitative Analysis

For our quantitative analysis we obtained and prepared data from the mainline Git repository as described in Section 2.2.1. We selected seven modules of the Linux kernel that play distinct roles in the architecture. Section 2.2.2 discusses the differences among these modules. We define the primary module for a maintainer (used in several analyses) in Section 2.2.3. The number of commits, authors, files, maintainers, and new joiners are used to quantify the growth of the Linux kernel (RQ2). New joiners are identified in Section 2.2.4. The growth of each maintainer's workload (RQ3) is calculated in Section 2.2.5 based on the aforementioned metrics. To understand the scalability of the maintainers' work (RQ5), we fit regression models as introduced in Section 2.2.6.

2.2.1 Data Preparation. We cloned the mainline repository of Linux kernel maintained by Linus Torvalds[5] in Dec 2015 and Jan 2017 respectively. We used the 2015 data set for exploration and present our results based on the 2017 clone. The Linux kernel moved to Git in 2005; the present study omitted the pre-2005 history of Linux. We took steps to clean and standardize the raw data for further analysis [41]. Each observation is a change (a commit may contain a group of related changes) submitted to the mainline repository. The repository records who authored the code, when the code was committed, and the name of the file involved, such as "drivers/pci/iova.c."

We also obtained the maintainer of the file as well as the module for that file. From April, 2009, the Linux kernel has included a file named MAINTAINERS that contains information needed to discover the maintainer for each file. The file records the name of the subsystem (e.g., UFS FILESYSTEM), people who maintain it, and the files associated with this subsystem, and the status of this subsystem, such as "maintained" (meaning that someone actually looks after the file) or "supported" (meaning someone is actually paid to look after the file). In addition, a perl program get_maintainer.pl uses the MAINTAINERS file to compute a file-to-maintainer mapping. Using these tools we obtained maintainers for every file in the Linux kernel for every month since April, 2009.

2.2.2 Modules Used for this Study. Modularity suits the characteristics of the open source production process [3]. Linux kernel has a folder structure that partially reflects the architecture of its modules. In particular, 22 folders exist at the first level; some of these folders, such as "include/documentation," do not represent a module; other folders, such as "tools/scripts," do not contain code that is a part of a running system. We, therefore, focus on seven folders that implement the main features, have the most changes, and may be considered as modules, as described in following passages.[6]

arch - Each supported processor architecture is in the corresponding folder. For example, the source code for Alpha processors is maintained in the "alpha" folder.

drivers - This directory contains the code for the drivers. Each folder is named after each piece or type of hardware. For example, the "bluetooth" folder holds the code for Bluetooth drivers.

fs - Inside this folder, each file system's code is in its own folder. For instance, the ext4 file system's code is in the "ext4" folder.

kernel - The code in this folder controls the kernel itself. For instance, if a debugger needed to trace an issue, the kernel would use code that originated from source files in this folder to inform the debugger of all of the actions that the kernel performs.

mm - The memory management folder contains the code for managing the memory.

net - The network folder contains the code for network protocols. This includes code for IPv6 and Appletalk as well as protocols for Ethernet, wifi, bluetooth, and other related functions.

sound - This directory has sound driver code for audio cards.

2.2.3 Defining Maintainer's Primary Module. To investigate phenomena across modules we must define which module a maintainer works on. As explained in Section 2.2.1, we obtained maintainers for every file and every month since April 2009. Two complications arose: 1) Some files, such as all files in the "Documentation" or "include" folders, do not belong to any module; 2) maintainers often work on multiple modules, for example, drivers and arch.

We used two approaches to address this problem. First, we report the statistics of the maintained or changed files based on the actual folders they were located in. We thus referred to the module as the original module, mod_0. In the second approach, we defined a primary module for a maintainer in order to calculate the average maintainer's workload for different modules. We assume that the files may be modularized better not by actual folder structure but by the sets of files a maintainer is maintaining [26]. This definition is distinct from the architectural definitions derived from call flows or data flow graphs. It is also distinct from the co-change definition used to define modules [24] in which multiple files are said to have the co-change relationship if all of those files are changed together in the same commit. We, therefore, first obtained the module for each maintainer-month which had the most files maintained by that maintainer. The module $mod_M(m) = \text{argmax}_{mod_0} \sum_{f \in mod_0} I(f, M, m)$ is the primary module for maintainer M at month m; where f is a file, $I(f, M, m)$ is the indicator function for M being a maintainer of f during month m.

For each file we, therefore, can uniquely obtain mod_0 based on its folder and for each maintainer M of the file we define mod_M as the primary module of M. Notably, both modules may vary over time: mod_0 may change if the file is moved to another folder and mod_M may change if the maintainer M changes her set of maintained files.

2.2.4 Identifying Joiners and One-time Contributors. For every author, we define the first time she authored code in the commit history as her joining time. We define any author who stayed with the project for less than one month (from her joining day) as a one-time contributor (OTC). Our latest data was retrieved in Jan 2017. We exclude joiners in the most recent year for the purpose of calculating OTCs. We thus avoid classifying developers who might commit within one year again as OTCs (see Figure 13).

2.2.5 Calculating Workload. The workloads (or productivity) of individual maintainers is complicated because over 75% of the files in the Linux kernel have more than one maintainer. To adjust individual maintainer output we subdivided effort equally among all maintainers of a file. To accomplish that we produce a weight for each file-maintainer pair. For example, suppose maintainer M maintains f_1 with one other maintainer and f_2 with two other maintainers during month m. Suppose, that f_1 is modified twice

[5] git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
[6] www.linux.org/threads/the-linux-kernel-the-source-code.4204/

Table 1: Average and median monthly workload of a maintainer

	drivers	arch	fs	net	sound	kernel	mm
#files	25.4	80.3	21	20.1	47	2.8	0.9
	3.3	6.5	10	7	1	1	0.5
#authors	1.6	2.1	2	1.9	1.5	1.5	0.9
	0.3	0.4	1	0.5	0.2	0.7	0.7
#commits	4.9	5.7	5.8	5.9	5.9	3.5	1.8
	0.4	0.5	1	1	0.3	1	0.8

by one author and f_2 is modified six times by six different authors. Then maintainer M's *adjusted effort* during month *m* is $1/2 + 1/3 = 5/6$ files, $1/2 * 2 + 1/3 * 6 = 3$ commits and $1/2 * 1 + 1/3 * 6 = 2.5$ authors. This measure preserves overall effort; if we add all the observations for a month we obtain the total number of maintained files, commits, and authors for that month. It also allows us to distribute effort among multiple modules by adding the relevant quantity for files in that module. More formally, the adjustment weight for a file f is obtained as:

$$w_m(f) = \frac{1}{\|\{M : M \text{ is a maintainer for } f \text{ during month } m\}\|} \quad (1)$$

Table 1 presents the monthly workload of a maintainer in different modules (mod_M). Each cell includes two values, one is the average adjusted workload and the other is median adjusted workload. Unadjusted workload ignores the fact that multiple maintainers are working on the same file, and is much higher. For example, for drivers, the average adjusted number of files that a maintainer maintains is 25.4 and unadjusted is 112.8. We, therefore, only consider adjusted workload in the remainder of the paper. Table 1 shows substantial differences (statistically significant) between average and median effort and among modules. These differences, which are explored in more detail in Section 3, may reflect how the ecosystem adapts to the variations in architecture, commercial involvement, and growing workloads.

2.2.6 Regression for Productivity Scaling. To understand the scalability of Linux ecosystem, we investigate how maintainer productivity scales when more maintainers are added to a set of maintained files. To model maintainer productivity we used multiple regression on logarithm-transformed data. The predictor and the response have much less skewed distribution after the transformation. Furthermore, the model: $\ln Output = \alpha * \ln NMaintainers + \dots$ has a very simple interpretation with $Output \sim NMaintainers^{\alpha}$, namely, α represents power, by which the productivity scales as maintainers are added to a file. We also included other major predictors that are likely to affect productivity: identity of the maintainer, and identity of the module. Whereas the individuals and modules greatly affect productivity, we are not concerned with the estimates for these predictors; they are nuisance parameters from our perspective and we adjust for that variation by including them as independent variables in our model.

3 RESULTS

This section presents the results for RQ2-5. We use mod_M when it is necessary to associate a maintainer with a single module, otherwise we use mod_0 (mod_M and mod_0 are defined in Section 2.2.3).

RQ2: How fast does the Linux ecosystem grow?

The Linux kernel has grown from 10.2 thousand lines of code in 1991 (version 0.01) to 22.3 million lines of code in 2016 (version 4.9). In recent years the rise in popularity of the Android operating system, which includes the Linux kernel, has made the kernel the most popular choice for mobile devices, rivaling the installed base of all other operating systems. We explore how the Linux ecosystem, particularly the central parts (represented by the seven modules defined in Section 2.2.2), grows over time in terms of commits, authors, files, maintainers, and new joiners.

Figure 1 presents the number of commits over time for different modules (mod_0). Modules drivers, arch, net, and sound have increasing numbers of commits. The other three modules appear to have a decreasing trend in the last few years. Modules mm and kernel have the fewest commits for the entire period. Please note that different scales are used to emphasize similarities in trends among modules. For example, numbers for the drivers module are divided by 10, but the numbers for the arch module are divided by three.

In Figure 2, the number of authors shows almost linear growth for drivers, arch, and net, whereas the growth for the arch module slows in the final years. Modules kernel and mm do not seem to have an increase, similar to their numbers of commits.

The number of maintained files, shown in Figure 3, increases almost constantly for almost all modules. The number of maintainers, shown in Figure 4, also demonstrates a rapid increase for almost all modules. The drivers module has particularly high increase in the number of files, which may be explained by the popularity of Linux with hardware vendors.

However, the inflow of joiners appears to drop, as shown in Figure 5. In particular, it shows a constant stream of joiners of approximately 1000/year for drivers, and an obvious decrease for almost all the other modules. Furthermore, the rate of joiners, even though it is declining, appears to exceed that of leavers as modules drivers, arch, and sound show linear growth in the number of authors (see Figure 2). In some modules, however, leavers appear to be balanced by joiners, resulting in a constant (or even slightly declining in net) number of authors contributing per year.

Figure 6 depicts the overall growth of the Linux kernel. The numbers of commits, authors, files, and maintainers appear to grow over time. However, the number of joiners does not appear to grow.

In summary, the amount of work measured by commits and authors appears to have stabilized for the core modules (e.g., kernel and mm) but continues to grow for the periphery (e.g., drivers). Some modules, particularly drivers, appear to contain most of the changes of the system. For example, in the most recent Linux kernel 4.9 (released in December 2016), "two-thirds of the bulk of changes are drivers[7]". The number of maintainers and the number of maintained files grow faster than the number of commits or the number of authors. Because the number of files is the easiest to measure, it may be the cause of concern expressed in public discussions. The remaining two measures do not appear to be exploding and appear to expand much less than the number of maintainers. It may be

[7] https://www.linux.com/news/linux-kernel-49-here-and-its-largest-release-ever

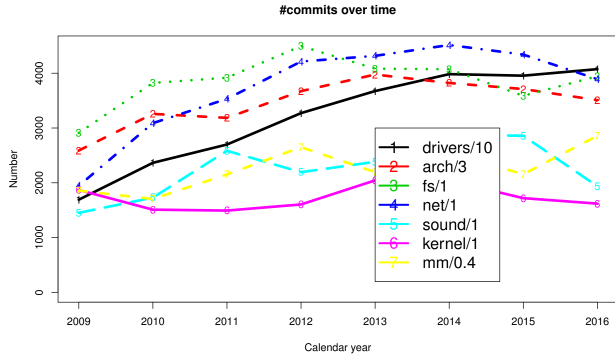


Figure 1: Numbers of commits in different modules

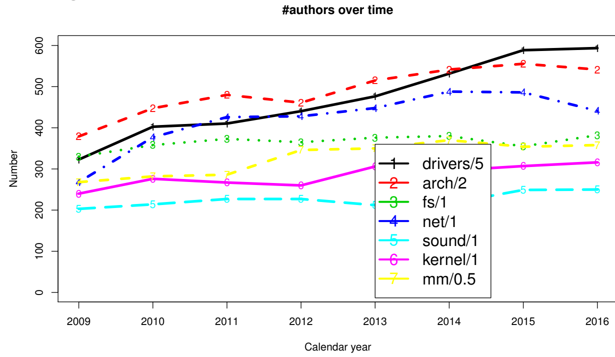


Figure 2: Numbers of authors in different modules

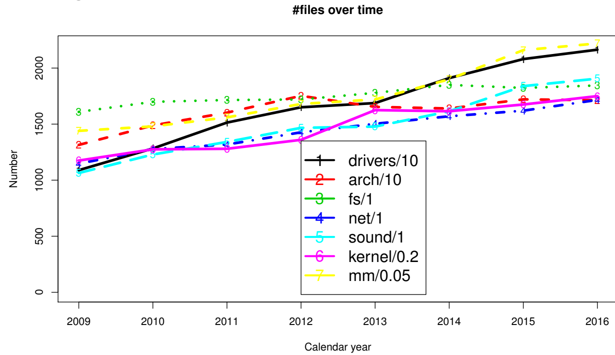


Figure 3: Numbers of maintained files in different modules

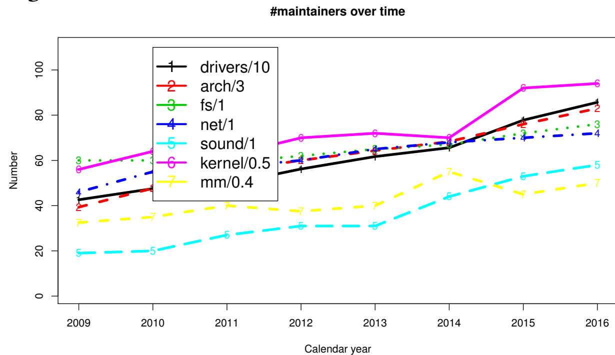


Figure 4: Numbers of maintainers in different modules

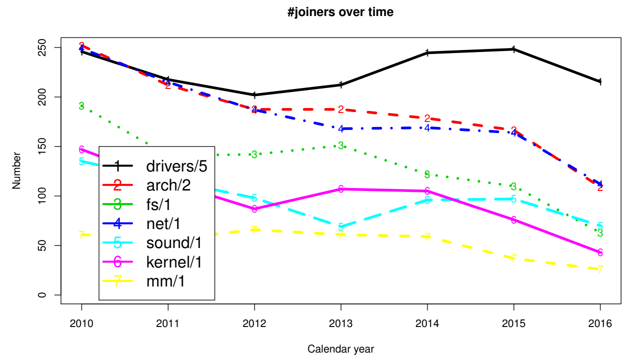


Figure 5: Numbers of joiners in different modules

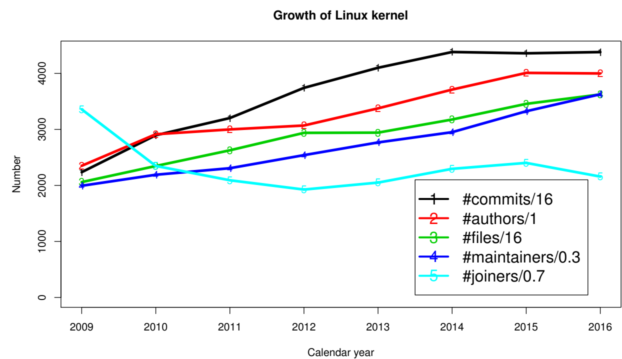


Figure 6: Overall growth of the Linux kernel

troubling to observe that the number of new joiners has been decreasing in recent years, but the total number of authors has been stable or is still expanding, at least for drivers.

RQ3: Has the maintainers' workload increased as the ecosystem has expanded?

Whereas the expansion as measured by commits and authors is contained within a few modules, the Linux kernel has continued to expand since 2009 both in terms of files and maintainers. Because both amount of work and number of workers have increased, it is not clear whether the workload per maintainer has increased. We calculated both average and median workload per maintainer on their primary module (mod_M) for each year (we also obtained monthly data, which has similar trends but is more noisy) to determine whether the maintainers' work is evenly distributed among maintainers.

Files. As shown in Figure 7, the number of files does not appear to be increasing for a median maintainer with one to two files in kernel, less than one in mm, and up to 12 for fs and arch. The drivers module appears to be the only module with a slightly increasing number of files per maintainer. Modules arch and sound appear to be decreasing all the time. The average number is more sensitive to extreme maintainers; arch and sound are at the top with 40 or more files per maintainer (and decreasing) and the rest of the modules have less than 30 files per maintainer. In particular, kernel and mm are at the bottom with less than 10 (and that number seems to hold constant). By comparing average and median workload

per maintainer among the seven modules, we can see that sound is at the bottom for a median maintainer but at the top for an average maintainer. A further investigation shows that the sound module had 19 maintainers in 2009 and 58 maintainers in 2016. A few maintainers are much more productive than others; the most productive, Jaroslav Kysela and Takashi Iwai, are the maintainers of the whole sound subsystem. This implies that the work is not balanced among maintainers.

Commits. As shown in Figure 8, the average number of commits per maintainer per month appears to be stable over time for most modules (mm, kernel, drivers, net, and fs). However, arch and sound have a clear decreasing trend (ranging from 10 to 4), similar to the situation for the number of files. Again, the mm module is at the bottom with two commits. The median number of commits per maintainer per month is less than 1.5 for all modules (not shown). This suggests what appears to be a rather moderate workload for a typical maintainer.

Authors. The yearly counts shown in Figure 2 show a rapid growth in the number of authors for drivers, arch, and net, with the remaining modules stabilizing. For example, yearly numbers for drivers increased from 1500 in 2009 to 3000 in 2016. Increase in maintainer numbers for drivers appears to be rapid also, resulting in a flat curve of authors per maintainer in this module, as shown in Figure 9. However, the numbers for sound and arch seem to be decreasing in a manner similar to the inflow of commits.

Author Churn. Whereas the number of authors a maintainer must handle over a one-year period stays relatively constant over time, the effort may increase if each month brings new authors or the effort may decrease if the same authors contribute over long periods of time. The number of authors per maintainer encountered over the last year appears to be much more stable (five for drivers to seven for mm and fs; 10 for arch and kernel; approximately 15 for net and sound) as shown in Figure 11 (to reduce noise, the monthly numbers were smoothed using a moving average with a window length of four). The monthly numbers of authors per maintainer were several times lower: 4ÃŮ for drivers and arch, 5ÃŮ for fs, net, and sound and 7ÃŮ lower for kernel and mm. The high ratios suggest that author churn from month to month in mm and kernel may make the work of mm and kernel maintainers more difficult than the work in the modules with more stable groups of authors, like drivers.

Figure 10 depicts the average workload of a maintainer in the Linux kernel. The average numbers of commits, authors, and files per maintainer appear to decline.

In summary, the maintainers in different modules differ in their workloads. Most importantly, a maintainer's workload does not seem to grow over time; on the contrary, it tends to decrease, particularly in modules arch and sound. The average values tend to be much higher than median values, suggesting highly uneven distribution of the work investigated in RQ4.

RQ4: How is the work distributed among Linux maintainers?

An average maintainer in the Linux kernel does not appear to have an increasing workload despite rapid expansion of the ecosystem. However, the difference between the median and average workload suggests that a small team of maintainers does most of the work. A similar relationship was observed between the core and the periphery of project contributors measured by lines added, issues fixed, and commits [23]. It is not clear whether the same relationships would apply to the maintainers at the center of the project, if their work were measured by number of files maintained, and numbers of authors and commits for the maintained files. If that relationship does apply, it may suggest that the Linux kernel ecosystem has self-similarity [28], i.e., the system has invariants that are preserved under a scale transformation.

Figure 12 shows the fraction of maintainers who are responsible for 80% of work (#files, #commits and #authors) plotted for each primary mod_M. Module drivers has the smallest core team of approximately 10% of maintainers handling files for up to 80% of the authors. At the other extreme, module mm deploys 80% of the maintainers to do 80% of the work. The other modules have their own "core teams" containing between 20% and 40% of all maintainers. This appears to support the conjecture of self-similarity, but the variations among modules, especially the outlier represented by mm, suggests that the self-similarity may not be completely uniform and warrants a closer investigation.

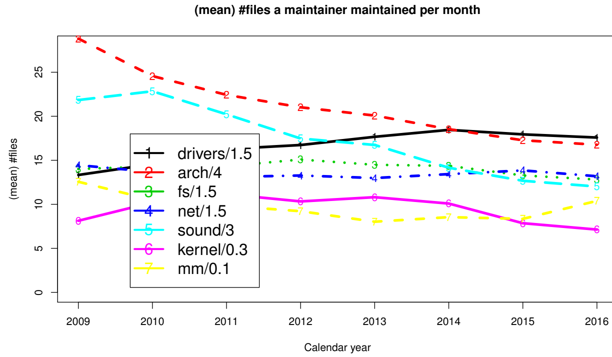
In summary, the distribution of work among maintainers appears to depend on the module. In particular, modules like drivers with well-modularized maintainers follow the 20/80 rule (i.e., 20% of the people accomplish 80% of the work), but modules where maintainers are often also involved in other modules, like mm, have a much more even distribution of work. Such self-similarity, as in biological ecosystems, would suggest that similar mechanisms must be at play in the context of Linux kernel maintainers as at the larger scale of core/peripheral contributors. Perhaps, to be effective, the community requires this rather extreme distribution of participants so as to be both predictable at delivering major features on time and to be able to incorporate a rich variety of small inputs from a much wider community.

RQ5: How well does the Linux ecosystem scale?

The growth of the features and the participants may challenge the core team of the Linux kernel. Discussions swirl around in the forums and conferences about how to expand the maintainer model, because, as one critic complained, "As the workload has increased, it has come to feel like things are getting much worse."[8] At the 2015 Kernel Summit, Linus Torvalds said that he has come to like the group maintainer model, where more than one person takes responsibility for a given subsystem. However, numerous developers were skeptical of the idea.[8]

To understand whether or not maintainers' work could scale by assigning more maintainers to the same files we fit models of maintainer productivity. Each observation represents a month (denoted as m) for a maintainer (denoted as M) and module (denoted as mod_0). The response variable was operationalized in three ways according to our three primary measures: files, authors, and commits. Table 2 presents the attributes of an observation used for the model. To model number of files (or authors, or commits) per maintainer we used the average number of maintainers over all maintained files as an independent variable, as Equation (2) shows. Note that the adjusted numbers reflect the adjusted contribution of a maintainer.

[8] https://lwn.net/Articles/703005/



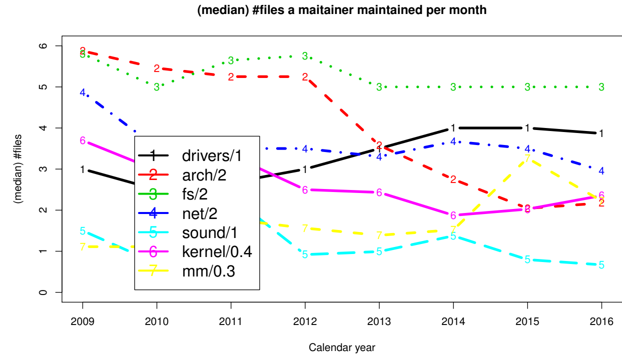


Figure 7: Average and median numbers of files maintained per maintainer

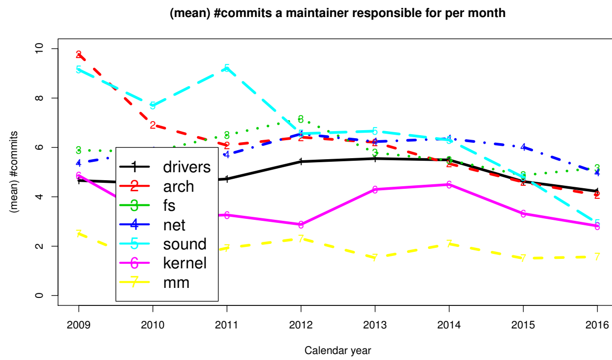


Figure 8: Average number of commits per maintainer

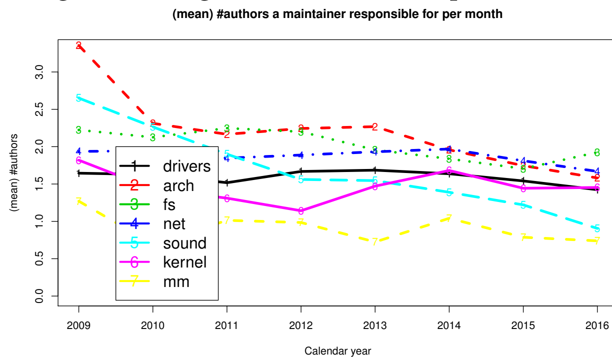


Figure 9: Average number of authors per maintainer

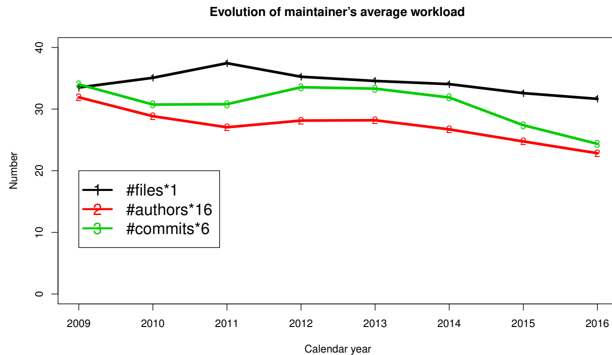


Figure 10: Average workload per maintainer

Table 2: An observation in the model

nF	#files maintained by M in module *mod* during month m
nfAdj	#files adjusted for the number of maintainers (adjustment is done using weights defined in Equation 1)
nAuth	#authors for the maintained files of M in module *mod* during month m adjusted for the number of maintainers
nCmt	#commits on the maintained files of M in module *mod* during month m adjusted for the number of maintainers
nMtr	#maintainers added over the files maintained by M in module *mod* during month m

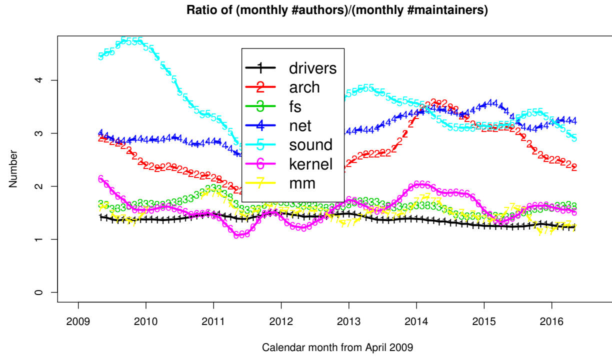
If it does not depend on the number of co-maintainers, that implies the effort scales linearly: each added maintainer contributes as much as if she were the sole maintainer. If adding additional maintainers to the same file diminishes productivity we would expect a negative exponent (see Section 2.2.6).

$$\log(nfAdj|nAuth|nCmt) \sim \log\frac{nMtr}{nF} + mod_0 + M \qquad (2)$$

The results are presented in Table 3. In all cases adjusted R^2 is fairly high, including 0.92, 0.65 and 0.73. The coefficient of interest α is close to $-.5$ and is statistically significant with an extremely small p-value. Notably, the arch module, in contrast to the other modules, has the most files and the mm has the fewest (see Table 3). This means that maintainers for arch were more productive in terms of maintained files. Maintainers for drivers supervised the most commits and authors and maintainers for mm supervised the fewest. A dual interpretation would be that the drivers module is the easiest to maintain and the mm module is the hardest.

The coefficients for $\log\frac{nMtr}{nF}$ mean the power at which maintainer productivity increases as more maintainers are added to a file. In particular, α for the number of files is -0.4, α for the number of commits is -0.54, and α for the number of authors is -0.59. Individual productivity decreases approximately by $\sqrt{\frac{nMtr}{nF}}$ but the number of maintainers goes up, so two maintainers can handle $\frac{2}{\sqrt{2}}$ more files (or commits, or authors) than one maintainer. Four maintainers can handle twice as much as one maintainer.

In summary, adding more maintainers to a file yields only a power of 1/2 increase in productivity, thus, four parallel maintainers



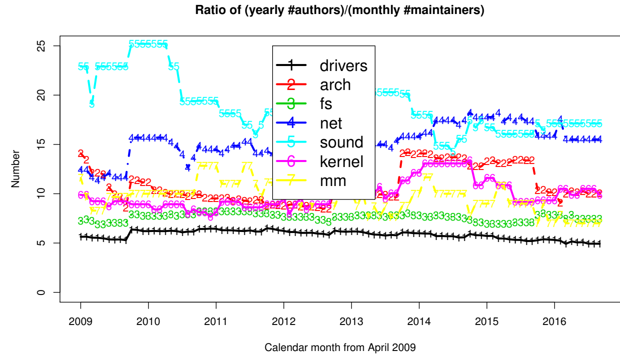
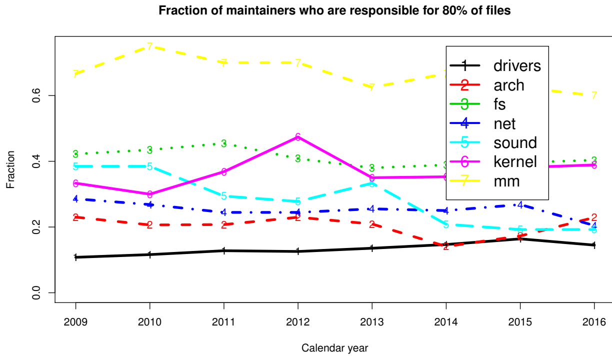
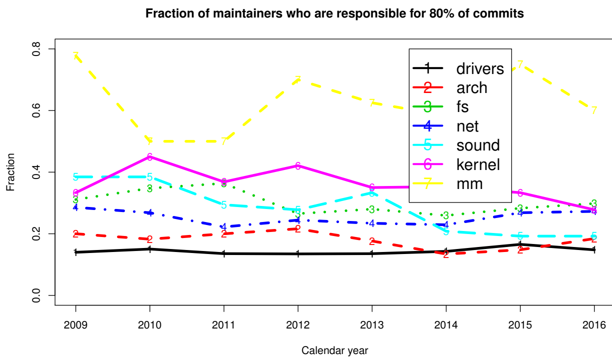


Figure 11: Ratio of number of authors to number of maintainers





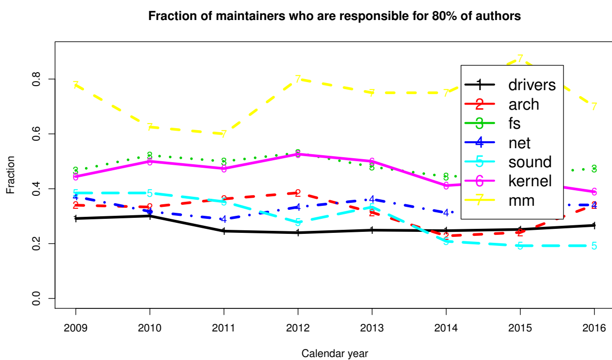


Figure 12: Fraction of maintainers who are responsible for 80% of files

Table 3: Results of modeling nfAdj/nAuth/nCmt: adjusted R^2: 0.92/0.65/0.73 (* indicates p-value $< .001$)

	nfAdj	Std. Error	nAuth	Std. Error	nCmt	Std. Error
(Intercept)	2.93*	0.08	−0.47	0.16	-0.21	0.24
$\log \frac{nMtr}{nF}$	−0.41*	0.01	−0.59*	-0.01	−0.54*	0.02
drivers	−0.12*	0.02	0.27*	0.02	0.25*	0.04
fs	−0.34*	0.04	−0.19*	0.05	−0.17	0.08
kernel	−1.43*	0.06	−0.34*	0.08	−0.39*	0.11
mm	−2.53*	0.06	−0.68*	0.07	−0.75*	0.10
net	−0.22*	0.04	−0.13	0.05	−0.26*	0.07
sound	−0.41*	0.06	−0.29*	0.08	-0.18	0.11

are needed to double the overall output. This suggests limits to the scalability that can be achieved by adding multiple maintainers to the same files.

4 MECHANISMS DIFFERENTIATING MODULES

Different maintainers' contributions on the seven central modules of the Linux kernel appear to differ in a consistent way. A review of various artifacts and interviews with the maintainers of Linux kernel suggests three mechanisms at play: architectural features, commercial support, and maintainer's skill sets.

Architectural features. The seven modules under study were chosen for distinct roles they play in the architecture of the Linux kernel. We also use the core-peripheral classification; for example, mm and kernel are core, drivers and arch are peripheral. As we have observed, after more than two decades of evolution, the core modules appear to have become mature and have less development activity and therefore fewer people looking after them. However, maintainers' work on these modules tends to be widely distributed; for example, 80% of the work is accomplished by 80% of the maintainers in mm. At the other extreme, the peripheral modules like drivers keep growing to satisfy various needs of hardware manufacturers and, therefore, involve large numbers of commits, authors, and maintainers. In drivers, 80% of the work is accomplished by only 10% of the maintainers. This highly uneven distribution of work suggests that some drivers maintainers may experience increasing workloads.

Commercial support. Most Linux code is currently developed by well-paid engineers.[9] Approximately 80% of kernel developers

[9]https://s3.amazonaws.com/storage.pardot.com/6342/120970/lf_pub_whowriteslinux2015.pdf

Table 4: Commercial support measures

	drivers	arch	fs	net	sound	kernel	mm
nF	6215	5463	212	332	114	65	0
$\frac{nF}{allF}$	.28	.32	.11	.19	.05	.18	0

are paid according to a senior Linux maintainer [27]. The most obvious and compelling reason is commercial interest of large (and rich) companies in the continued robust health of Linux. Twenty years ago, Linux was the plaything of hobbyists and supercomputer makers – today, it powers everything from smartphones (Android) to wireless routers to set-top boxes.

This commercial interest may not be uniform for different parts of the Linux kernel. The drivers module, for example, has strong support from hardware manufacturers interested in increasing the market for their products. However, modules like mm do not appear to have features that would represent a business opportunity for numerous companies. Table 4 lists the two measures that are likely to reflect commercial interests in the actual modules of the Linux kernel (mod_0): the number of supported files (nF) and the fraction among all files ($\frac{nF}{allF}$). The drivers module has the largest number of supported files, followed by arch. The mm module has none. The biggest fraction of supported files is in arch, followed by drivers. This may partially explain the differences of growth in different modules. In particular, modules like mm do not have commercial backing and may have to seek contributions from the community.

Breadth of maintainers' reach. "At the functional level, different modules do vastly different things, and require vastly different skill sets to be able to contribute in a meaningful manner" as one interviewee noted. Presumably, most maintainers specialize in a single area. This could be illustrated with two measures: the fraction of maintainers who exclusively maintain their primary module mod_M and the fraction of all maintainers of mod_0 for whom $mod_0 = mod_M$, as shown in Table 5. Let $I(M, mod_0) = 1 \iff \exists m, f \in mod_0 : I(f, M, m) = 1$ and 0 otherwise where $I(f, M, m)$ is defined in Section 2.2.3. The first fraction is then $\frac{\sum_{M:(mod_M=mod_0) \quad (M \; maintains \; only \; mod_M)} I(M, mod_0)}{\sum_{M:mod_M=mod_0} I(M, mod_0)}$. The second fraction is $\frac{\sum_{M:mod_M=mod_0} I(M, mod_0)}{\sum_M I(M, mod_0)}$.

For modules sound, kernel, and mm, no more than half of the maintainers primarily work on these modules, whereas more than 80% of the drivers maintainers work either primarily or exclusively on drivers. A driver implementation is relatively self-contained.[6] A single developer can add a new device driver, and that addition requires minimal interaction with other kernel developers. However, "the skill level required to work on kernel/mm versus fixing up spelling or whitespace changes in drivers/staging is vastly different" according to another respondent. The fraction of maintainers for whom kernel and mm are primary modules and who work exclusively on kernel and mm (the first row of Table 5) may be relatively high because so few maintainers are capable of maintaining these modules and the ones that do, may not have time for other parts of the kernel. For example, "mm is a very tiny subsystem, yet a very core one, so the people working on it are much more specialized and experienced than âĂŸnormal' driver developers." Meanwhile, the tasks of driver development are often considered to represent

Table 5: Multi-module maintainers

	drivers	arch	fs	net	sound	kernel	mm
Exclusive	0.89	0.35	0.68	0.54	0.57	0.59	0.77
Primary	0.84	0.70	0.70	0.64	0.43	0.50	0.48

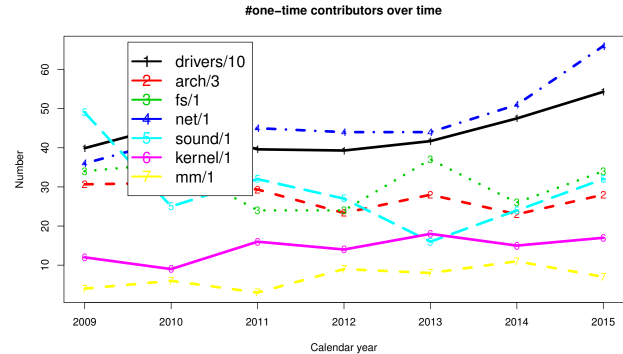


Figure 13: Number of OTCs in different modules

lower entry barrier for newcomers.[10] Our models of productivity in RQ5 support the view that the drivers module is the easiest from the maintainers' perspective as well: it is the easiest in terms of commits and authors and the second easiest (after arch) in terms of files maintained. Figure 13 shows that the mm module has the fewest OTCs; each year it has fewer than ten (with kernel having fewer than twenty), whereas the drivers module has 400-600 OTCs each year (mod_0).

In summary, each module implements different technical features and may attract different groups of contributors with distinct profiles, unique skill sets, and particular commercial interests. The ecosystem uses community support for central tasks and central code, but the modules of the architectural periphery are most effectively supported by commercial entities (except for the kernel module). The ecosystem grows primarily because commercial entities add code to the drivers module; the community resolves conflicts in core modules such as mm. Other systems may learn from these strategies to adapt and scale their efforts, e.g., to distribute their resources based on architectural features and to utilize commercial support and community strength on different parts of the system.

5 LIMITATIONS

We used the Git repository of the Linux kernel to reconstruct past versions of the MAINTAINERS file and associated scripts; we obtained a list of maintainers for each file. Only individuals explicitly labeled as "maintainer" were considered. People who do not want their names to appear in the MAINTAINERS file are excluded. Further, we unified multiple identities for many of the maintainers and authors based on email, login, and full name; some of these identifications may have been erroneous. Our approach may not capture all ways in which maintainers may spend their effort. For example, the participation in discussion groups was not considered. Because we considered only the mainline repository of the Linux kernel, a large number of patches that did not gain acceptance into the mainline repository were excluded [12]. Reviewing and rejecting

[10] https://www.linux.com/news/software/linux-kernel/804403-three-ways-for-beginners-to-contribute-to-the-linux-kernel/eudyptula-challenge.org

these patches consume maintainers' effort even though they are not accepted into the mainline repository. However, according to one interviewee, "It's easier for the maintainer to not accept your code at all. To accept it, it takes time to review it, apply it, send it on up the development chain, handle any problems that might happen with the patch, accept responsibility for the patch, possibly fix any problems that happen later on when you disappear, and maintain it for the next 20 years." Therefore, it is not unreasonable to assume that the accepted commits can represent the bulk of the maintainers' efforts.

To increase internal validity we interviewed Linux kernel maintainers and inspected various online resources to validate our findings. We checked the assumptions for the regression model and log-transformed the predictors to make the model coefficients more interpretable and to reduce the influence of the potential outliers.

The uniqueness of the Linux kernel limits external validity. As one of the largest cooperative software projects ever attempted [17], the Linux kernel may have unique practices. Despite that, other projects might benefit from the practices of Linux kernel; as one interviewee said: "I've been spending lots of time helping other projects scale that are having problems. ... we work together to see if they can adapt things that we do, or I help them identify the pain points they have, they try a change, iterate, and see if it's better or not." Yin [37] wrote that the question of how to generalize from a single case is challenging for case studies. The short answer is that case studies are generalizable to theoretical propositions and not to populations or universes. The theoretical propositions in this study are the quantification of effort spent by the maintainers in the project and the quantification of work across maintainers.

6 RELATED WORK

The division of labor and distribution of tasks is a common theme in the FLOSS literature because FLOSS projects rely on volunteers who are dispersed across organizational and geographical boundaries. Lee and Cole [18] reported that the Linux community has a project leader, several hundred maintainers, and thousands of developers; the patterns of Linux resemble the patterns of community organization revealed in other studies. For example, Ducheneaut [8] presented a pattern with core developers in the center, surrounded by the maintainers, patchers (who fix bugs), bug reporters, documenters, and, finally, the users of the software. They characterized a community as a series of concentric circles; each circle is occupied by people playing a particular role in the development process. In this study we focus on the central circle of the Linux kernel ecosystem: maintainers.

How developer communities evolve and sustain has been subject to numerous investigations. Two constructs are considered crucial to an FLOSS team's input effectiveness [30]: the number of developers that have been attracted and retained to work on the team (team size) and the amount of effort those developers have devoted. Two factors shape the lifecycle of a successful FLOSS project [3]: a widely accepted leadership setting the project guidelines and driving the decision process, and an effective coordination mechanism among the developers based on shared communication protocols. Studies of the progressive integration of new members [8, 29, 32, 40] and the evolution of sustainable groups [15, 25] are common. For example, Zhou and Mockus [40] found that individuals' initial willingness and environment affect their chances of staying long term with the FLOSS project.

The laws of software evolution were stated by Lehman [19] and have been widely observed by others [22, 35, 38]. Evolutionary studies of long-lived, large-scale FLOSS ecosystems have attracted some attention [5, 11]. Wermelinger et al. [34, 35] found that the Eclipse architecture is always growing but components on a layered plugin-architecture dependency graph exhibit different evolution patterns. Fortuna et al. [9] found that the modularization of the Debian network over time in various operating system installations often parallels ecological relationships between interacting species.

Building on prior work we investigated Linux maintainers to quantify the types of tasks they do and the relationships between their work patterns and architectural and governance aspects of the contributors they work with. Whereas developers' work has been extensively studied [4, 10, 23, 36, 39], we adopted existing metrics, and also added new metrics for maintainers.

7 CONCLUSIONS

We investigated maintainer activities in the Linux ecosystem and quantified maintainers' work based on the files they maintain, the change activity in the maintained files, and the number and churn of external contributors they must deal with. We found that most of the modules did not grow appreciably over the last decade; the majority of code was added to three modules: drivers, architecture, and net. However, the number of files has grown for all modules, which may be the cause of concern expressed in public discussions. We found systematic yet stable differences among modules suggesting that the architectural features, commercial interests, or module-specific practices led to distinct sustainable maintenance equilibria. We also found that the workloads of the average maintainer and the median maintainer do not appear to increase, thus some risks hypothesized in the community are not evident. However, the distribution of work among maintainers showed that 20/80 rule applies for most modules, suggesting that a few maintainers may bear the brunt of the increased workload.The practice of assigning multiple maintainers to a file yielded only a power of 1/2 increase in productivity.

Our proposed framework to quantify maintainer practices and productivity scaling may lead to a better understanding of the factors that allow rapidly growing projects to be sustainable and to practices that reduce risk of project failures. The mechanisms underlying such large-scale complicated production remain to be explored because the ecosystem never stops evolving. In the words of one maintainer: "5 years from now it [Linux kernel] will look different depending on the people involved, the external forces happening, and what the needs of the project is."

ACKNOWLEDGMENTS

This work is supported by the National Basic Research Program of China Grant 2015CB352200, the National Natural Science Foundation of China Grants 61432001, 61421091, and 61690200 and the National Science Foundation Award 1633437.

To facilitate replications or other types of future work, we provide data and scripts used in this study online:
https://github.com/minghuizhou/maintainerAnalysis.

REFERENCES

[1] A.J. Albrecht and Jr. Gaffney, J.E. 1983. Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. *IEEE Transactions on Software Engineering* SE-9, 6 (November 1983), 639–648.

[2] P. Anbalagan and M. Vouk. 2009. On predicting the time taken to correct bug reports in open source projects. In *IEEE International Conference on Software Maintenance, 2009.* 523–526.

[3] Cristina Rossi Andrea Bonaccorsi. 2003. Why Open Source software can succeed. *Research Policy* 32 (2003) (2003), 1243–1258.

[4] G. Avelino, L. Passos, A. Hora, and M. T. Valente. May 2016. A novel approach for estimating truck factor. In *IEEE 24th International Conference on Program Comprehension (ICPC), 2016.* 1–10.

[5] Christopher Bogart, Christian Kästner, James Herbsleb, and Ferdian Thung. 2016. How to break an API: Cost negotiation and community values in three software ecosystems. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering.* ACM, 109–120.

[6] Frederick P. Jr. Brooks. 1975. *he Mythical Man-Month.* Addison-Wesley.

[7] J. W. Creswell. 2009. *Research design: Qualitative, quantitative, and mixed methods approaches* (3rd edition ed.). Sage Publications.

[8] Nicolas Ducheneaut. 2005. Socialization in an Open Source Software Community: A Socio-Technical Analysis. *Computer Supported Cooperative Work (CSCW)* 14, 4 (2005), 323–368. http://dx.doi.org/10.1007/s10606-005-9000-1

[9] M. A. Fortuna, J. A. Bonachela, and S. A. Levin. 2011. Evolution of a modular software network. *Proceedings of the National Academy of Sciences of the United States of America* 108, 50 (2011), 19985–9.

[10] Daniel M German. 2003. The GNOME project: a case study of open source, global software development. *Software Process: Improvement and Practice* 8, 4 (2003), 201–215.

[11] Daniel M German, Bram Adams, and Ahmed E Hassan. 2013. The evolution of the R software ecosystem. In *17th European Conference on Software Maintenance and Reengineering (CSMR), 2013.* IEEE, 243–252.

[12] Daniel M. German, Bram Adams, and Ahmed E. Hassan. 2016. Continuously Mining Distributed Version Control Systems: An Empirical Study of How Linux Uses Git. *Empirical Softw. Engg.* 21, 1 (Feb. 2016), 260–299. DOI:http://dx.doi.org/10.1007/s10664-014-9356-2

[13] T. Graves and A. Mockus. 2001. Identifying Productivity Drivers by Modeling Work Units Using Partial Data. *Technometrics* 43, 2 (May 2001), 168–179.

[14] Guido Hertel, Sven Niedner, and Stefanie Herrmann. 2003. Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel. *Research Policy* 32, 7 (2003), 1159 – 1177. http://www.sciencedirect.com/science/article/pii/S0048733303000477

[15] R. E. Kraut and P. Resnick. 2012. *Building successful online communities: Evidence-based social design.* Cambridge, MA: MIT Press.

[16] Greg Kroah-Hartman. 2013. I don't want your code: Linux Kernel Maintainers, why are they so grumpy?, In https://github.com/gregkh/presentation-linux-maintainer/blob/master/maintainer.pdf. (2013).

[17] Greg Kroah-Hartman, Jonathan Corbet, and Amanda McPherson. March 2008. Linux Kernel Development. *The Linux Symposium* (March 2008).

[18] Gwendolyn K. Lee and Robert E. Cole. 2003. From a Firm-Based to a Community-Based Model of Knowledge Creation: The Case of the Linux Kernel Development. *Organization Science* 14, 6 (2003), 633–649. http://dx.doi.org/10.1287/orsc.14.6.633.24866

[19] Meir M Lehman, Juan F Ramil, Paul D Wernick, Dewayne E Perry, and Wladyslaw M Turski. 1997. Metrics and laws of software evolution-the nineties view. In *Software Metrics Symposium, 1997. Proceedings., Fourth International.* IEEE, 20–32.

[20] Fink M. 2003. *The business and economics of Linux and open source.* Prentice Hall Professional.

[21] A. Meneely and L. A. Williams. 2009. Secure open source collaboration: an empirical study of linus' law. In *Proceedings of the ACM 2009 Conference on Computer and Communications Security.*

[22] Tom Mens, Maálick Claes, Philippe Grosjean, and Alexander Serebrenik. 2014. Studying evolving software ecosystems based on ecological models. In *Evolving Software Systems.* Springer, 297–326.

[23] A. Mockus, R. F. Fielding, and J. Herbsleb. 2000. A Case Study of Open Source Development: The Apache Server. In *22nd International Conference on Software Engineering.* Limerick, Ireland, 263–272. http://dl.acm.org/authorize?2580

[24] Audris Mockus and David M. Weiss. 2000. Predicting Risk of Software Changes. *Bell Labs Technical Journal* 5, 2 (April–June 2000), 169–180.

[25] SiobhÃąn O'Mahony and Fabrizio Ferraro. 2007. The Emergence of Governance in an Open Source Community. *Academy of Management Journal* 50, 5 (Oct 1 2007), 1079–1106.

[26] D. L. Parnas. 1972. On the Criteria to be Used in Decomposing Systems into Modules. *Commun. ACM* 15, 12 (1972), 1053–1058.

[27] Sarah Sharp. December, 2014. Linux Kernel Introduction. In *https://www.slideshare.net/saharabeara/linux-kernel-introduction.*

[28] Makse H A. Song C, Havlin S. 2005. Self-similarity of complex networks[J]. *Nature* 7024 (2005), 392–395.

[29] Igor Steinmacher, Tayana Conte, Marco Aurélio Gerosa, and David Redmiles. 2015. Social Barriers Faced by Newcomers Placing Their First Contribution in Open Source Software Projects. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing (CSCW '15).* ACM, New York, NY, USA, 1379–1392.

[30] Katherine J. Stewart and Sanjay Gosain. 2006. The Impact of Ideology on Effectiveness in Open Source Software Development Teams. *MIS Quarterly* 30, 2 (2006), pp. 291–314.

[31] Linus Torvalds. 1999. The Linux Edge. *Commun. ACM* 42, 4 (April 1999), 38–39. DOI:http://dx.doi.org/10.1145/299157.299165

[32] Georg von Krogh, Sebastian Spaeth, and Karim R. Lakhani. 2003. Community, joining, and specialization in open source software innovation: a case study. *Research Policy* 32, 7 (July 2003), 1217–1241.

[33] Georg von Krogh and Eric von Hippel. 2003. Special issue on open source software development. *Research Policy* 32, 7 (2003), 1149 – 1157. http://www.sciencedirect.com/science/article/pii/S0048733303000544

[34] Michel Wermelinger and Yijun Yu. 2008. Analyzing the Evolution of Eclipse Plugins. In *Proceedings of the 2008 International Working Conference on Mining Software Repositories (MSR '08).* ACM, New York, NY, USA, 133–136. http://doi.acm.org/10.1145/1370750.1370783

[35] Michel Wermelinger, Yijun Yu, and Angela Lozano. 2008. Design principles in architectural evolution: a case study. In *IEEE International Conference on Software Maintenance, 2008.* IEEE, 396–405.

[36] Jialiang Xie, Minghui Zhou, and Audris Mockus. 2013. Impact of Triage: a Study of Mozilla and Gnome. In *ESEM 2013.* Baltimore, Maryland, USA, 247–250.

[37] Robert K. Yin. 2009. *Case Study Research: Design and Methods. Fourth Edition.* SAGE Publications, California.

[38] Liguo Yu and Alok Mishra. 2013. An empirical study of LehmanâĂŹs law on software quality evolution. *International Journal of Software & Informatics* 7, 3 (2013), 469–481.

[39] Minghui Zhou and Audris Mockus. 2010. Developer Fluency: Achieving True Mastery in Software Projects. In *ACM SIGSOFT / FSE.* Santa Fe, New Mexico, 137–146. http://dl.acm.org/authorize?309273

[40] Minghui Zhou and Audris Mockus. 2015. Who Will Stay in the FLOSS Community? Modeling Participant's Initial Behavior. *Software Engineering, IEEE Transactions on* 41, 1 (Jan 2015), 82–99. DOI:http://dx.doi.org/10.1109/TSE.2014.2349496

[41] Jiaxin Zhu, Minghui Zhou, and Hong Mei. 2016. Multi-extract and Multi-level Dataset of Mozilla Issue Tracking History. In *Proceedings of the 13th International Conference on Mining Software Repositories (MSR '16).* ACM, New York, NY, USA, 472–475. DOI:http://dx.doi.org/10.1145/2901739.2903502