Assessing Code Authorship:
The Case of the Linux Kernel

Guilherme Avelino[1,2] , Leonardo Passos[3] , Andre Hora[1] ,
and Marco Tulio Valente[1(✉)]

[1] Federal University of Minas Gerais (UFMG), Belo Horizonte, Brazil
{gaa,hora,mtov}@dcc.ufmg.br
[2] Federal University of Piaui (UFPI), Teresina, Brazil
[3] University of Waterloo, Waterloo, Canada
lpassos@gsd.uwaterloo.ca

Abstract. Code authorship is a key information in large-scale open-source systems. Among others, it allows maintainers to assess division of work and identify key collaborators. Interestingly, open-source communities lack guidelines on how to manage authorship. This could be mitigated by setting to build an empirical body of knowledge on how authorship-related measures evolve in successful open-source communities. Towards that direction, we perform a case study on the Linux kernel. Our results show that: (a) only a small portion of developers (26%) makes significant contributions to the code base; (b) the distribution of the number of files per author is highly skewed—a small group of top-authors (3%) is responsible for hundreds of files, while most authors (75%) are responsible for at most 11 files; (c) most authors (62%) have a specialist profile; (d) authors with a high number of co-authorship connections tend to collaborate with others with less connections.

Keywords: Code authorship · Linux kernel · Developer networks

1 Introduction

Collaborative work and modularization are key players in software development, specially in the context of open-source systems [14,23,27]. In a collaborative setup imposed by open-source development, code authorship allows developers to identify which project members to contact upon maintaining existing parts of the code base. Additionally, authorship information allows maintainers to assess overall division of work among project members (e.g., to seek better working balance) and identify profiles within the team (e.g., specialists versus generalists).

Our notion of authorship is broader than the English definition of the word. In the context of code, authorship relates to those who make significant changes to a target file. This may include the original file creator, as well as those who subsequently change it. Hence, different from authorship in books and scientific papers, code authorship is inherently dynamic as a software evolves.

Problem Statement. Currently, open-source communities lack guidance on how to organize and manage code authorship among its contributors.

Research Goal. We argue that the stated problem could be mitigated by setting to build an empirical body of knowledge on how authorship-related measures evolve in successful open-source communities. In that direction, we investigate authorship in a large and long-lived system—the Linux kernel. Our goal is to identify authorship parameters from the Linux kernel evolution history, as well as interpret why they appear as such. We also check whether those parameters apply to the subsystem level, allowing us to assess their generality across different parts of the kernel. Our analysis accounts for 56 stable releases (v2.6.12–v4.7), spanning a period of over 11 years of development (June, 2005–July, 2016).

Research Questions. When investigating the Linux kernel authorship history, we follow three research questions:

RQ1: What is the distribution of the number of files per author?

Motivation: Answering such a question provides us with a measure of the work overload and the concentration of knowledge within team members, as well as how that evolves over time.

RQ2: How specialized is the work of Linux authors?

Motivation: Following the Linux kernel architectural decomposition, we seek to understand the proportion of developers who have a narrower understanding of the system (specialists), versus those with a broader knowledge (generalists). Specialist developers author files in a single subsystem; generalists, in turn, author files in different subsystems. Answering our research question seeks to assess how effective the Linux kernel architectural decomposition is in fostering specialized work, a benefit usually expected from a good modularization design [3,30].

RQ3: What are the properties of the Linux co-authorship network?

Motivation: The authorship metric we use enables identifying multiple authors per file, evidencing a co-authorship collaboration among developers [20]. Such collaborations form a network—vertices denote authors and edges connect authors sharing common authored files. This question seeks to identify collaboration properties in the Linux kernel co-authorship network.

Contributions. From the investigation we conduct, we claim two major contributions: (a) an in-depth investigation of authorship in a large, successful, and long-lived open-source community, backed up by several authorship measures when answering each of our research questions. The findings we report also serve researchers, allowing them to contrast the authorship in the Linux kernel with those of other communities; (b) the definition of several authorship-centric concepts, such as authors and specialists/generalists, that others may use as a common ground to study the social organization of software systems.

In Sect. 2, we provide a description of our study design. Section 3 details our results. Sections 4 and 5 discuss threats to validity and related work, respectively. Section 6 concludes the paper, also outlining future work.

2 Study Design

2.1 Author Identification

At the core of our study lies the ability to identify and quantify authorship at the source code level. To identify file authors, as required by our three research questions, we employ a normalized version of the *degree-of-authorship* (DOA) metric [9,10]. The metric is originally defined in absolute terms:

$$DOA_A(d, f) = 3.293 + 1.098 * FA + 0.164 * DL - 0.321 * \ln(1 + AC)$$

From the provided formula, the absolute degree of authorship of a developer d in a file f depends on three factors: first authorship (FA), number of deliveries (DL), and number of acceptances (AC). If d is the creator of f, *FA* is 1; otherwise it is 0; *DL* is the number of changes in f made by d; and *AC* is the number of changes in f made by other developers. DOA_A assumes that FA is by far the strongest predictor of file authorship. Further changes by d (DL) also contributes positively to his authorship, but with less importance. Finally, changes by other developers (AC) contribute to decrease someone's DOA_A, but at a slower rate. The weights we choose stem from an experiment with professional Java developers [9]. We reuse such thresholds without further modification.

The normalized DOA (DOA_N) is as given in [2]:

$$DOA_N(d, f) = DOA_A(d, f)/max(\{DOA_A(d', f) \mid d' \in changed(f)\})$$

In the above equation, $changed(f)$ denotes the set of developers who edited a file f up to a snapshot of interest (e.g., release). This includes the developer who creates f, as well as all those who later modify the file. $DOA_N \in [0..1]$: 1 is granted to the developer(s) with the highest absolute DOA among those changing f; in other cases, DOA_N is less than one.

Lastly, the set of authors of a file f is given by:

$$authors(f) = \cup\{d \mid d \in changed(f) \wedge DOA_N(d, f) > 0.75 \wedge DOA_A(d, f) \geq 3.293\}$$

The authors identification depends on specific thresholds—0.75 and 3.293. Those stem from a calibration setup when applying DOA_N to a large corpus of open-source systems. For full details, we refer readers to [2].

2.2 Linux Kernel Architectural Decomposition

Investigating authorship at the subsystem level requires a reference architecture of the Linux kernel. Structurally, the Linux kernel architectural decomposition comprises seven major subsystems [8]: *Arch* (architecture dependent code), *Core*

Table 1. Linux subsystems size and authors proportion

Subsystem	Files		Authors proportion			
			Last release (v4.7)			All releases
	#	%	Developers	Authors	Proportion	Avg ± Std Dev
Driver	22,943	42%	10,771	2,604	24%	25.00 ± 0.80%
Arch	17,069	32%	3,613	1,145	32%	33.10 ± 1.28%
Misc	6,621	12%	644	78	12%	14.85 ± 2.69%
Core	3,840	7%	4,165	1,083	26%	25.77 ± 1.56%
Net	1,957	4%	2,161	269	13%	13.63 ± 0.90%
Fs	1,809	3%	1,777	175	10%	12.61 ± 1.95%
Firmware	151	0%	–	–	–	–
All	54,400	100%	13,436	3,459	26%	26.86 ± 0.83%

(scheduler, IPC, memory management, etc.), *Driver* (device drivers), *Firmware* (firmware required by device drivers), *Fs* (file systems), *Net* (network stack implementation), and *Misc* (miscellaneous files, including documentation, samples, scripts, etc.). To map files in each subsystem, we rely on mapping rules set by G. Kroah-Hartman, one of the main Linux kernel developers.[1] Table 1 shows the number of files in each kernel subsystem as mapped by using the expert rules.

2.3   Data Collection

We study 56 stable releases of the Linux kernel, obtained from LINUS/TORVALDS GitHub repository. A stable release is any named tag snapshot whose identifier does not have a $-rc$ suffix. To define the *authors* set of a file f in a given release r, we calculate DOA_N from the first commit up to r. It happens, however, that the Linux kernel history is not fully stored under Git, as explained by Linus Torvalds in the first commit message.[2] Therefore, we use `git graft` to join the history of all releases prior to v2.6.12 (the first release recorded in Git) with those already controlled by Git. After join, we increment the Linux kernel Git history with 64,468 additional commits.

Given the entire development history, we check out each stable release at a time, listing its files, and calculating their DOA_N. In the latter case, we rely on `git log --no-merges` to discard merge commits and retrieve all the changes to a given file prior to the release under investigation. To compute the DOA_N, we only consider the author of each commit, not its committer (Git repositories store both) [6]. It is worth noting that prior to calculate DOA_N, we map possible aliases among developers, as well as eliminate unrelated source code files. As

---

[1] https://github.com/gregkh/kernel-history/blob/master/scripts/stats.pl.
[2] https://github.com/torvalds/linux/commit/1da177e4c3f41524e886b7f1b8a0c1fc73 21cac2.

example, the *Firmware* subsystem was removed because most of its files are binary blobs. To perform these steps, we adopt the procedures described in [2].

Table 1 shows the proportion of authors in each Linux subsystem. In the last release, Linux kernel has 13,436 developers, but only 3,459 (26%) are authors of at least one file. Throughout the kernel development, the proportion of authors is nearly constant (*Std dev* = ±0.83%). Thus, the heavy-load Linux kernel maintenance has been kept in the hands of less than a third of all developers.

Using custom-made scripts, we fully automate authorship identification, as well as the collection of supporting data for the claims we make. Our infrastructure is publicly available on GitHub.[3] We encourage others to use it as means to independently replicate/validate our results.

3   Results

(RQ.1) Distribution of the Number of Files per Author

What is the distribution of the number of files per author?

The number of files per author is highly skewed. Figure 1 presents the boxplots of files per author across the Linux kernel releases (adjusted for skewness—see [15]). To simplify the visualization of the results, we present the boxplots at each two releases. With exception of one release (v2.6.24), 50% of the authors responds to at most three files (median); for 75% of the authors, the number of files ranges from 11 to 16. Outliers follow from the skewed distribution. Still, the number of authors with more than 100 files is always lower than 7% of the authors, ranging from 7% in the first release to 3% in the last one. Similar behavior is observed at the subsystem level. In the last release (v4.7), for instance, the number of files per author up to the 75% percentile in *Fs*, *Arch*, and *Driver* closely resemble one-another and the global distribution as a whole—all share the same median (three). *Core* and *Misc*, however, have less variability than the other subsystems, as well as lower median values (two and one, respectively).
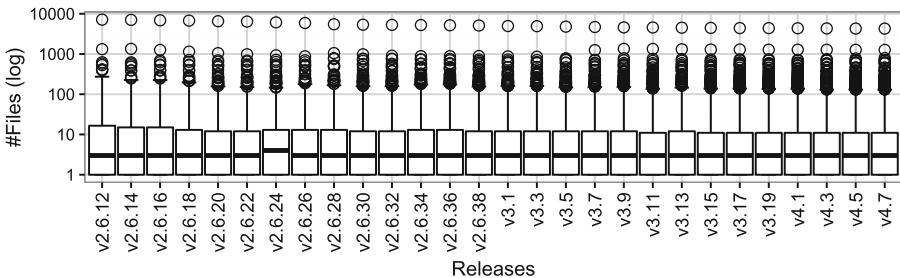

Fig. 1. Distribution of the number of files per author in each release

---

It is interesting to note that file authorship follows a pyramid-like shape of increasing authority; at the top, Linus Torvalds acts as a "dictator", centralizing authorship of most of the files (after all, he did create the kernel!). Bellow him lies his hand-picked "lieutenants", often chosen on the basis of merit. Such organization directly reflects the Linux kernel contribution dynamics, which is itself a pyramid [4]. However, as the kernel evolves, we see that Torvalds is becoming more "benevolent". As Fig. 2 shows, the percentage of files authored by him has reduced from 45% (first release) to 10% in v4.7. Currently, he spends more time verifying and integrating patches than writing code [7]. Similar behavior is observed downwards the authorship pyramid. The percentage of files in the hand of the next top-9 Linux kernel authors (bars) is consistently decreasing. This suggests that authorship is increasing at lower levels of the pyramid, becoming more decentralized. This is indeed expected and to an extent required to allow the Linux kernel evolves at the pace it does.
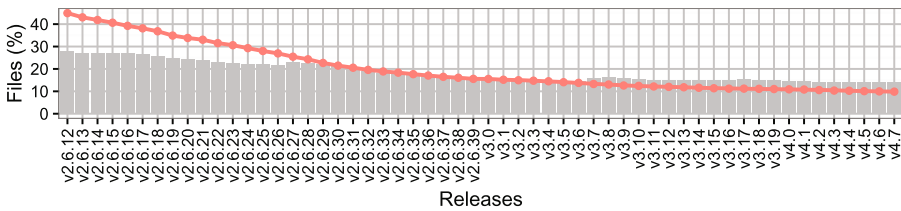
Fig. 2. Percentage of files authored by the top-10 authors over time. The line represents Linus Torvalds (top-1) and the bars represent the accumulated number of files of the next top-9 authors

We also apply the Gini coefficients [12] to analyze the distribution of the number of files per author (Fig. 3). In all releases, the coefficient is high, confirming skewness. However, we notice a decreasing trend, ranging from 0.88 in the first release to 0.78 (v4.7). Such a trend further strengthens our notion that authorship in the Linux kernel is becoming less centralized.
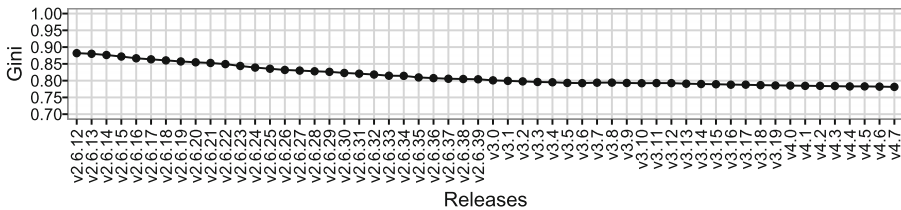
Fig. 3. Gini coefficients. It ranges from 0 (perfect equality) to 1 (perfect inequality).

(RQ.2) Work Specialization

How specialized is the work of Linux authors?

To assess work specialization, we introduce two author profiles. We call authors *specialists* if they author files in a single subsystem. *Generalists*, in turn, author files in at least two subsystems. As Fig. 4 shows, the number of specialists dominates the amount of generalists. In the Linux kernel (All), any given release has at least 61% of specialist authors, with a maximum of 64%; at all times, 39% of the authors are generalists. Moreover, the proportion of generalists and specialists appears to be fairly stable across the entire kernel (All) and its constituent subsystems (except for *Misc*).
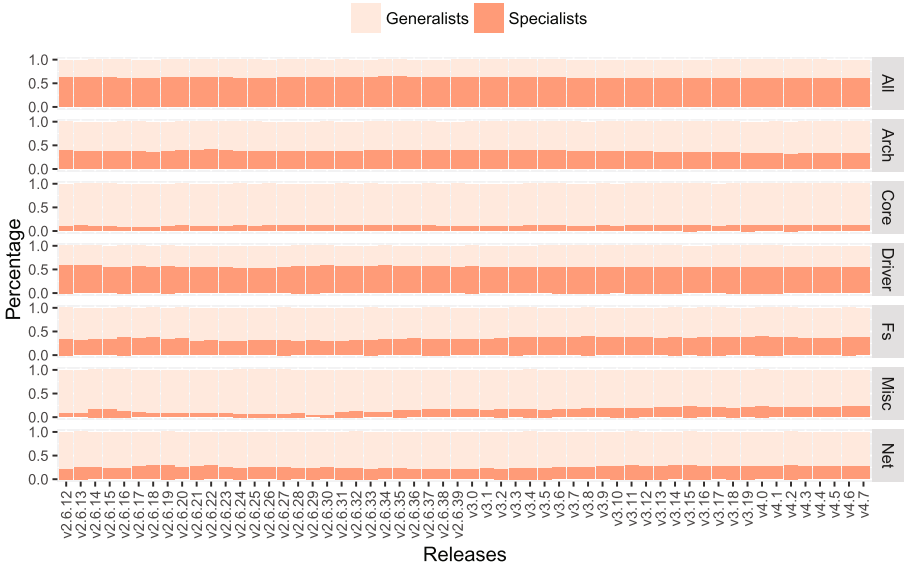

Fig. 4. Percentage of specialists and generalists

Looking at the work specialization in each subsystem also provides a means to assess how much the Linux kernel architectural decomposition fosters specialized work. The architectural decomposition plays a key role in fostering specialists inside the *Driver* subsystem (more than 50% of specialists), but less so elsewhere. The reason it occurs so extensively inside *Driver* follows from the plug-in interface of the latter and its relative high independence to other subsystems [8,28]. In contrast, *Core* and *Misc* have the lowest percentage of specialized workers. More than 75% of their authors own files in more than one subsystem. Specifically, *Core* is the subsystem with the lowest percentage of specialized workers (13%). This is also expected since *Core* developers tend to have expertise on Linux's central features, which allows them to also work on other subsystems.

(RQ.3) Co-authorship Properties

What are the properties of the Linux co-authorship network?

Many files in the Linux kernel result from the work of different authors. As such, we set to investigate such collaboration by means of the properties of the Linux kernel *co-authorship network*. We model the latter as follows: vertices stand for Linux kernel authors; an edge connects two authors v_i and v_j if $\exists f$ such that $\{v_i, v_j\} \subseteq authors(f)$. In other words, an edge represents a *collaboration*.

To answer our research question, we initially analyze the latest co-authorship network, as given in release v4.7 (Table 2).[4] The number of vertices (authors) determines the size of a co-authorship network. The mean degree network, in turn, inspects the number of co-authors that a given author connects to. In the system level (All), the mean vertex degree is 3.64, i.e., on average, a Linux author collaborates with 3.64 other authors. At the subsystem level, *Driver* forms the largest network (2,604 authors, 75%), whereas *Misc* results in the smallest one (78 authors, 2%). *Arch* has the highest mean degree (3.14 collaborators per author); *Misc* has the lowest (0.79). Linus Torvalds has connections with 215 other authors. His collaborations spread over all subsystems and range from 92 collaborations in *Driver* to five in *Misc*. Excluded Torvalds, the top-2 and top-3 authors with more collaborators have 156 and 118 collaborators, respectively.

Table 2. Co-authorship network properties (release v4.7)

	All	Driver	Arch	Core	Net	Fs	Misc
Number of vertices	3,459	2,604	1,145	1,083	269	175	78
Mean degree	3.64	2.74	3.14	1.67	2.57	2.59	0.79
Clustering coefficient	0.080	0.074	0.128	0.074	0.205	0.175	0.188
Assortativity coefficient	−0.070	−0.115	−0.060	−0.072	−0.003	−0.146	−0.062

The third property, *clustering coefficient*, reveals the degree to which adjacent vertices of a given vertex tend to be connected [31]. In a co-authorship network, the coefficient gives the probability that two authors who have a co-author in common are also co-authors themselves. A high coefficient indicates that the vertices tend to form high density clusters. The clustering coefficient of the Linux kernel is small (0.080). Nonetheless, *Net*, *Misc*, and *Fs* exhibit a higher tendency to form density clusters (0.205, 0.188, and 0.175, respectively) in comparison to other subsystems. The three subsystems are the smallest we analyze, a factor that influences the development of collaboration clusters [1].

Last, but not least, we compute the *assortativity coefficient*, which correlates the number of co-authors of an author (i.e., its vertex degree) with the number of co-authors of the authors it is connected to [26]. Ranging from −1 to 1, the coefficient shows whether authors with many co-authors tend to collaborate with

---

[4] We use the R *igraph* (version 1.0.1) to calculate all measures.

other highly-connected authors (positive correlation). In v4.7, all subsystems
have negative assortativity coefficients, ranging from −0.134 in *Fs* to −0.029 in
Net subsystem. This result diverges from the one commonly observed in scientific
communities [25]. Essentially, this suggests that Linux kernel developers often
divide work among experts who help less expert ones. These experts (i.e., highly-
connected vertices), in turn, usually do not collaborate among themselves (i.e.,
the networks have negative *assortative coefficients*).

We identify in the co-authorship networks a relevant amount of *solitary
authors*—authors that do not have co-authorship with any other developer. In
total, 20% (699) of Linux kernel developers are solitary. Although there is a
high percentage of solitary authors, only 9% of them have more than three files.
Additionally, 66% of them work in the *Driver* subsystem. The latter is likely to
follow from the high proportion of specialists within that subsystem (see RQ.2).

Evolution of Co-authorship network properties. We set to investigate how
the co-authorship properties evolved to those in release v4.7. Figure 5 displays
the corresponding graphics. Although we can observe a small decrease in some
intermediate releases, by looking at the first and last releases, the mean degree
has little variation, ranging from 3.61 to 3.64. Clustering coefficient, in turn,
varies from 0.099 (first release) to 0.080 (v4.7). Since the mean degree does
not vary considerably, we interpret such decrease as an effect of the growth of
the number of authors (network vertices). The latter creates new opportunities
of collaboration, but these new connections do not increase the density of the
already existing clusters. A similar behavior is common in other networks, as
described by Albert and Barabási [1]. Finally, we observe a relevant variation
in the evolution of assortativity coefficients. Measurements range from −0.25 in
the first release to −0.07 in v4.7. Such a trend aligns with the decrease of the
percentage of files authored by Linus Torvalds and the other top authors (refer
to RQ.1). With less files, these authors are missing some of their connections
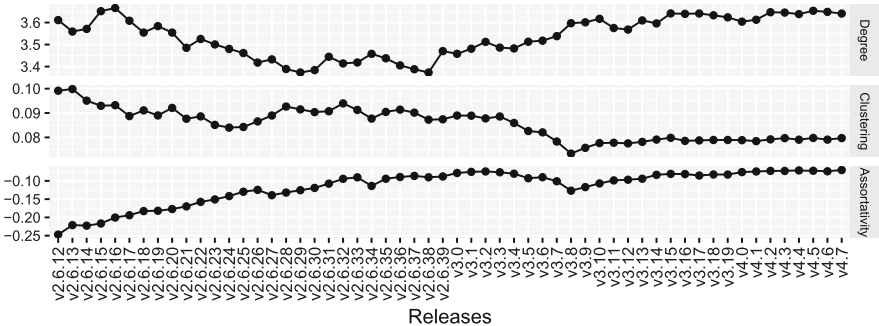and becoming more similar (in terms of vertex degree) to their co-authors.

Fig. 5. Co-authorship network properties over time

4   Threats to Validity

Construct Validity. Our results depend on the accuracy of DOA calculations. Currently, we compute DOA values using weights from the analysis of other systems [9,10]. Although the authors of the absolute DOA claim their weights are general, we cannot fully eliminate the threat that the choice of weights pose to our results. Still, we have previously applied them when analyzing different open-source systems, obtaining positive feedback from developers [2].

Internal Validity. We measure authorship considering only the commit history of the official Linux kernel Git repository. Hence, we do not consider forks that are not merged into the mainstream development. Although these changes might be relevant to some (e.g., studies about integration activities [11]), they are not relevant when measuring authorship of the official Linux kernel codebase. We also consider that all commits have the same importance. As such, we do not account for the granularity of changes (number of lines of code affected) nor their semantics (e.g., bug fixes, new features, refactoring, etc.).

External Validity. The metrics we use can be applied to any software repository under a version control system. Still, our findings are very specific to the Linux kernel development. Thus, we cannot assume that the findings about workload, specialization, and collaboration are general. Nonetheless, we pave the road for further studies to validate our findings in the context of other systems.

5   Related Work

Code Authorship. McDonald and Ackerman propose the "Line 10 Rule", one of the first and most used heuristics for expertise recommendation [19]. The heuristic considers that the last person who changes a file is most likely to be "the" expert. Expertise Browser [24] and Emergent Expertise Locator [22] are alternative implementations to the "Line 10 Rule". The former uses the concept of experience atoms (EA) to give the value for each developer's activity and takes into account the amount of EAs to quantify expertise. The latter refines the Expertise Browser approach by considering the relationship between files that change together. Fine-grained algorithms that assign expertise based on the percentage of lines a developer has last touched are used by Girba et al. [13] and by Rahman and Devanbu [29].

Social Network Analysis (SNA). Research in this area use information from source code repositories to build a social network, adopting different strategies to create the links between developers. Fernández et al. [18] apply SNA, linking developers that perform commits to the same module, to study their relationship and collaboration patterns. Others rely on fine-grained relations, building networks connecting developers that change the same file [5,16,21,32]. Joblin et al. [17] propose an even more fine-grained approach, connecting developers that change the same function in a source code. They claim that file-based links result in dense

networks, which obscures important network properties. Our approach, although centered on file-level information, does not produce dense networks, as authorship requires that developers make significant contributions to a file.

6   Conclusion

In this paper, we extract and analyze authorship parameters from a successful case: the Linux kernel. By mining over 11 years of the Linux kernel commit history, we investigate how authorship changes over time, deriving measures that other communities mirroring the Linux kernel evolution could directly replicate. Moreover, our study provides the grounds for further analyses—we define authorship concepts setting basic terminology and operationalization, in addition to providing a dataset of a large case study that others may use as a comparison baseline. As future work, we seek to validate our findings directly with Linux kernel developers. Moreover, we plan to study authorship in other systems.

References

1. Albert, R., Barabási, A.L.: Statistical mechanics of complex networks. Rev. Modern Phys. **74**, 47–97 (2002)
2. Avelino, G., Passos, L., Hora, A.C., Valente, M.T.: A novel approach for estimating truck factors. In: 24th International Conference on Program Comprehension (ICPC), pp. 1–10 (2016)
3. Baldwin, C.Y., Clark, K.B.: Design Rules: The Power of Modularity. MIT Press, Cambridge (1999)
4. Bettenburg, N., Hassan, A.E., Adams, B., German, D.M.: Management of community contributions. Empir. Softw. Eng. **20**(1), 252–289 (2015)
5. Bird, C., Nagappan, N., Murphy, B., Gall, H., Devanbu, P.: Don't touch my code! In: 19th International Symposium on Foundations of Software Engineering (FSE), pp. 4–14 (2011)
6. Chacon, S., Straub, B.: Pro Git. Expert's voice in software development, 2nd edn. Apress, New York (2014)
7. Corbet, J., Kroah-Hartman, G., McPherson, A.: Who writes Linux: Linux kernel development: how fast it is going, who is doing it, what they are doing, and who is sponsoring it. Technical report, Linux Foundation (2013). http://www.linuxfoundation.org/publications/linux-foundation/who-writes-linux-2013
8. Corbet, J., Rubini, A., Kroah-Hartman, G.: Linux Device Drivers, 3rd edn. O'Reilly, Sebastopol (2005)
9. Fritz, T., Murphy, G.C., Murphy-Hill, E., Ou, J., Hill, E.: Degree-of-knowledge: modeling a developer's knowledge of code. ACM Trans. Softw. Eng. Methodol. **23**(2), 14:1–14:42 (2014)
10. Fritz, T., Ou, J., Murphy, G.C., Murphy-Hill, E.: A degree-of-knowledge model to capture source code familiarity. In: 32nd International Conference on Software Engineering (ICSE), pp. 385–394 (2010)

11. German, D.M., Adams, B., Hassan, A.E.: Continuously mining distributed version control systems: an empirical study of how Linux uses Git. Empir. Softw. Eng. **21**, 260–299 (2015)

12. Gini, C.: Measurement of inequality of incomes. Econ. J. **31**(121), 124–126 (1921)

13. Girba, T., Kuhn, A., Seeberger, M., Ducasse, S.: How developers drive software evolution. In: 8th International Workshop on Principles of Software Evolution (IWPSE), pp. 113–122 (2005)

14. Herbsleb, J.D.: Global software engineering: the future of socio-technical coordination. In: 2007 Future of Software Engineering (FOSE), pp. 188–198 (2007)

15. Hubert, M., Vandervieren, E.: An adjusted boxplot for skewed distributions. Comput. Stat. Data Anal. **52**(12), 5186–5201 (2008)

16. Jermakovics, A., Sillitti, A., Succi, G.: Mining and visualizing developer networks from version control systems. In: 4th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), pp. 24–31 (2011)

17. Joblin, M., Mauerer, W., Apel, S., Siegmund, J., Riehle, D.: From developer networks to verified communities: a fine-grained approach. In: 37th International Conference on Software Engineering (ICSE), pp. 563–573 (2015)

18. López-Fernández, L., Robles, G., Gonzalez-Barahona, J.M., Herraiz, I.: Applying social network analysis techniques to community-driven libre software projects. Int. J. Inf. Technol. Web Eng. **1**(3), 27–48 (2006)

19. McDonald, D.W., Ackerman, M.S.: Expertise recommender: a flexible recommendation system and architecture. In: Conference on Computer Supported Cooperative Work (CSCW), pp. 231–240 (2000)

20. Meneely, A., Williams, L.: Socio-technical developer networks: should we trust our measurements? In: 33rd International Conference on Software Engineering (ICSE), pp. 281–290 (2011)

21. Meneely, A., Williams, L., Snipes, W., Osborne, J.: Predicting failures with developer networks and social network analysis. In: 16th International Symposium on Foundations of Software Engineering (FSE), pp. 13–23 (2008)

22. Minto, S., Murphy, G.C.: Recommending emergent teams. In: 4th Workshop on Mining Software Repositories (MSR), p. 5 (2007)

23. Mistrík, I., van der Hoek, A., Whitehead, J.: Collaborative software engineering: challenges and prospects. In: Mistrik, I., Grundy, J., Hoek, A., Whitehead, J. (eds.) Collaborative Software Engineering, 4th edn, pp. 389–403. Springer, Heidelberg (2010)

24. Mockus, A., Herbsleb, J.D.: Expertise browser: a quantitative approach to identifying expertise. In: 24th International Conference on Software Engineering (ICSE), pp. 503–512 (2002)

25. Newman, M.E.J.: Coauthorship networks and patterns of scientific collaboration. Proc. Natl. Acad. Sci. **101**, 5200–5205 (2004)

26. Newman, M.E.J.: Mixing patterns in networks. Phys. Rev. E **67**, 026126 (2003)

27. Parnas, D.L.: On the criteria to be used in decomposing systems into modules. Commun. ACM **15**(12), 1053–1058 (1972)

28. Passos, L., Padilla, J., Berger, T., Apel, S., Czarnecki, K., Valente, M.T.: Feature scattering in the large: a longitudinal study of Linux kernel device drivers. In: 14th International Conference on Modularity, pp. 81–92 (2015)

29. Rahman, F., Devanbu, P.: Ownership, experience and defects. In: 33rd International Conference on Software Engineering (ICSE), pp. 491–500 (2011)

30. Sullivan, K.J., Griswold, W.G., Cai, Y., Hallen, B.: The structure and value of modularity in software design. In: 9th International Symposium on Foundations of Software Engineering (FSE), pp. 99–108 (2001)
31. Watts, D.J., Strogatz, S.H.: Collective dynamics of small-world networks. Nature **393**, 440–442 (1998)
32. Yang, X.: Social network analysis in open source software peer review. In: 22nd International Symposium on Foundations of Software Engineering (FSE), pp. 820–822 (2014)