Engineering Formality and Software Risk in Debian Python Packages

Matthew Gaughan
Northwestern University
Email: gaughan@u.northwestern.edu

Kaylea Champion
University of Washington
Email: kaylea@uw.edu

Sohyeon Hwang
Northwestern University
Email: sohyeonhwang@u.northwestern.edu

***Abstract*—While free/libre and open source software (FLOSS) is critical to global computing infrastructure, the maintenance of widely-adopted FLOSS packages is dependent on volunteer developers who select their own tasks. Risk of failure due to the misalignment of engineering supply and demand — known as underproduction — has led to code base decay and subsequent cybersecurity incidents such as the Heartbleed and Log4Shell vulnerabilities. FLOSS projects are self-organizing but can often expand into larger, more formal efforts. Although some prior work suggests that becoming a more formal organization decreases project risk, other work suggests that formalization may increase the likelihood of project abandonment. We evaluate the relationship between underproduction and formality, focusing on formal structure, developer responsibility, and work process management. We analyze 182 packages written in Python and made available via the Debian GNU/Linux distribution. We find that although more formal structures are associated with higher risk of underproduction, more elevated developer responsibility is associated with less underproduction, and the relationship between formal work process management and underproduction is not statistically significant. Our analysis suggests that a FLOSS organization's transformation into a more formal structure may face unintended consequences which must be carefully managed.**

I. Introduction

Across global computing infrastructures, free/libre open source software (FLOSS) packages underpin the successful operation of widely used technical systems [1]. Yet this crucial software is often "underproduced" — that is, not as well-maintained as we might expect, given its importance [2]. Underproduction can be consequential, leading to disruption across global networks. Two notorious such failures are the FLOSS security defects known as Heartbleed and Log4Shell. The Heartbleed vulnerability impacted the widely-used OpenSSL library. At the time the vulnerability was announced, OpenSSL was decaying from community abandonment, with no full time developers and only $2000 in annual donations. Heartbleed was eventually remediated by large-scale, formally organized developer engagement — the very kind which OpenSSL had previously lacked [3]. The Log4j library is maintained by The Apache Software Foundation and a dedicated development team. In 2021, researchers identified the zero-day Log4Shell vulnerability, which enables attackers to inject malicious code into already-running Java programs [4]. While the Log4j team was able to respond quickly with a series of patches, the long-undetected nature of this vulnerability suggests that like OpenSSL, Log4j was underproduced.

With 96% of software in the 'critical infrastructure sector' containing FLOSS code, the United States Cybersecurity and Infrastructure Security Agency (CISA) established an "Open Source Software Security Roadmap" in September 2023 [5] highlighting the need to study the social and technical antecedents of FLOSS software failure. This national attention is timely: security research firm Sonatype found in their 2023 *State of the Software Supply Chain Report* that across four major engineering ecosystems (NPM, Maven, PyPi, nugent) only 11% of projects were actively maintained [6]. Heightened scrutiny of FLOSS engineering practices as well as escalating warning signs of underproduction make examining the causes of underproduction all the more urgent.

A promising direction to diagnose both causes of and remediation strategies for underproduction is examining a structure commonly employed in FLOSS projects: commons based peer production (CBPP) [7]. Benkler defines CBPP as a production model reliant on decentralized individuals who self-select tasks, motivated by a range of factors [8]. Yet Benkler also writes that in the face of "knowledge intensive, creative, and complex" problems, projects must organize to "[elicit] diverse pro-social motivations" from contributors [9]. Said simply, although CBPP is an extremely valuable way of collaborating, FLOSS organizations may find it difficult to ameliorate underproduction because requisite yet undesirable tasks may lay neglected when contributors work according to their own interests.

In this paper, we make three contributions: we offer empirical assessments of the relationship between (1) underproduction and overall governance formality, (2) underproduction and the concentration of developer responsibility, and (3) underproduction and the management of work products. We place our work in the context of previous work in §II and describe our analysis in §III before presenting results of our analysis in §IV. We discuss the implications of this work in §V with limitations noted in §VI before concluding in §VII.

II. Background

A. Governance in CBPP

Governance refers to the totality of the systems in an organization which incorporate decision making, operational control, and incentives [10]. Thus, as an organization matures, institutionalization in governance can be understood as "the

processes by which the social processes obligations, or actualities come to take a rulelike status in social thought or action" across subunit actors (in the case of FLOSS project engineering, software developers) [11]. In CBPP governance, project institutionalization entrenches either formality or informality in decision-making processes [12].

The influential work of Ostrom on governing commons-based resources notes that communities prefer internally developed governance practices, which naturally leads to variations in the resulting governance arrangements [13]. For example, an analysis by Hwang and Shaw of CBPP governance on Wikipedia found that communities with shared goals, technical infrastructures, organizational structures, and institutional trajectories end up producing diverging rule sets and deliberating at length over shared rules [14]. At times, how CBPP communities self-govern can lead to counter-intuitive patterns of institutionalization that introduce bureaucratization and hierarchies and create later barriers to participation, following Robert Michel's "iron law of oligarchy." [15]. The full extent of the productive repercussions of the variety of approaches to CBPP remains unclear.

B. Governance of FLOSS Engineering

Similar to other CBPP communities, FLOSS projects adhere to a wide range of organizational forms; the management of functional processes (leadership elections, funding allocation, onboarding, conflict resolution) is often internally defined with governance documents such as project constitutions and charters [16], [17]. Core to FLOSS governance is the allocation of developer labor and permissions (i.e., code integration, and how work products are released). For example, in centralized projects, formal project leadership may retain engineering management power, exemplified by Linus Torvald's sole control of code merging into the Linux kernel [18].

Code merge patterns—who gets their work merged, and who decides—are a key indicators of how a project is being governed. Even in technically flat communities, Onoue et al. frame participant engagement in FLOSS projects as a hierarchy, with differences in who can enact development actions such as commits, pull requests, comments, and issue events [19]. Thus, users who have the requisite permissions to merge code from peripheral forks into the primary development branch are empowered in project governance over those who simply commit to derivative branches. Tamburri et al. and van Meijel use the hierarchy of merge permissions as evidence of project hierarchy [20]–[22]. De Stefano et al. adopt this approach to conclude that formality in internal engineering governance structures is associated with diminished community contribution to project code bases [23].

Tools like project progress trackers also give insight into FLOSS development. The management of such work products are often disputed within project communities; Crowston et al. observes that that there is no common patterns to how FLOSS projects employ public releases [1]. GitHub "milestones" are indicators to mark the current state of a project, and are used by communities to track progress on collections of tasks such as issues or pull requests[1]. While the milestone feature is platform-specific, GitHub remains the preeminent platform for FLOSS project hosting [24]. Zhang et al. found a strong positive correlation between milestone usage and the count of other engineering measures, like commits and releases, in part due to the fact that older projects are more likely to use milestones than younger projects [25].

C. Governance and Underproduction

Yin et al. observe that in canonical software engineering literature, the success of FLOSS software is primarily defined in two ways: functional development processes or community cohesion [26]. Functional development processes include the management of project risk. In these terms, underproduction is a metric which considers potential for risk (developer neglect), its impact (user adoption), and the likelihood of risk resolution (code quality, bug resolution speed.) Champion and Hill identify underproduction by evaluating whether project quality is aligned with its importance when compared to a theoretical baseline of alignment, where importance and quality are aligned if their non-parametric rankings are the same [2]. For a set of packages from the Debian GNU/Linux distribution, Champion and Hill represent project importance through installation count and project quality as the average time to bug resolution (controlling for bug severity).

Prior work examining FLOSS governance formalization vary in their measures of success and subsequent conclusions; Crowston and Howison even suggest that informality itself may be a metric of project success [27]. For critical FLOSS systems, Tamburri et al. defines success as "24/7 availability in (a) fault-tolerant system" [20]. Yin et al. uses the advancement metrics of the Apache Software Foundation's incubator program to represent project success [28]. De Stefano et al. focus on the frequency of product commits as a metric of community engagement and project success [23]. As such, Tamburri observes that as formality within software projects increases, projects may experience friction in the processes of developer engagement [20]; Yin et al. conclude that isomorphic reproduction of formal governance structure may lead to project success [28]; lastly, De Stefano et al. conclude that the more constrained a project's processes around merging are, the less engineering engagement it might receive [23].

Observations that formalizing FLOSS governance may lead to both project abandonment and project success are not contradictory but warrant further empirical study. Overall, these findings suggest that projects with higher levels of formality are more likely to be underproduced. Due to the observations of De Stefano et al. [23] and Tamburri et al. [21], we propose: **$\mathbf{H_A}$: projects with higher formality are more likely to be underproduced**. Moreover, considering the findings from prior work on merge processes, we propose that **$\mathbf{H_B}$: projects with less concentrated developer responsibility are more likely to be underproduced**. Given the argument in Yin et al.

[1]https://docs.github.com/en/issues/using-labels-and-milestones-to-track-work/about-milestones

that the governance of project work assists the development process [28], we propose **$\mathbf{H_C}$: projects with formal work process management are less likely to be underproduced.**

III. Methods

A. Empirical Setting

GNU/Linux distributions, includingDebian in particular, have a substantial history as settings for understanding software engineering communities [29], including their effective governance [30], [31] and lifecycles [32]. We examine a collection of projects packaged via the Debian GNU/Linux distribution, which is also a packaging source for Ubuntu and several other distributions. All projects were tagged by Debian developers as being implemented in the Python language, and all have repositories on GitHub. Therefore, these packages are part of one of the most widely-used GNU/Linux operating systems worldwide, with an upstream language and source code management platform in common.

B. Data

Our unit of analysis is the software project, n = 182. All projects meet three criteria: they are in the Debian GNU/Linux distribution, included in the dataset published by Champion and Hill [2], and tagged by Debian maintainers as being written in Python. Due to the manual identification of upstream repositories, our data set was constrained to Debian packages written in the Python language; further research is necessary to study the Debian package ecosystem writ large. The median project in this dataset was 13 years old, with 44 members in their developer communities.

To identify the upstream repository of each package that was the same as the one used by Debian, we first examined Debian metadata: in the Ultimate Debian Database [33], on the Debian package website[2], and inside the package as stored in Debian's GitLab instance[3]. If no upstream repository location was identified, we examined the files associated with the package (README.txt if available, and the documentation and license files if not).We then collected package commit and milestone history using the CHAOSS GrimoireLab Perceval tool [34] and the public GitHub API, respectively. We bounded our data collection within the range of 2/8/2008 - 11/09/2023; February 8, 2008 was the date of GitHub's founding.[4]

C. Measures

We operationalize project *governance formality* (H_A) using Tamburri et al.'s YOSHI formality score [20], one of six metrics addressing project governance (alongside community structure, geodispersion, longevity, engagement, and cohesion). YOSHI has been used extensively in prior empirical studies of FLOSS governance [21]–[23], [35], [36].

$$\text{Formality Score} = \frac{MMT}{MS/LS} \tag{1}$$

[2]https://packages.debian.org/

[3]https://salsa.debian.org/

[4]Our data and code are available on the Harvard Dataverse: https://doi.org/10.7910/DVN/WENTBH

This measure considers developer responsibility, work processes, and age as component variables to calculate overall project formality, shown in Equation 1. *Mean Membership Type* (MMT) represents developer responsibility. This is further explained below with Equation 3. MS refers to work processes, in our case GitHub milestone count, while LS refers to project lifespan in days. The potential range of scores is between 0 and 10,000. However, because this definition would filter out projects who do not use milestones entirely—a substantial portion of our sample— we develop an augmented calculation of the formality score where the MMT is divided by whether or not the project employs milestones (represented by a binary 1:2 classification) (MSE) per the project's age grouping (AG), which is included as a control variable given age may generally impact formalization. Projects were grouped in bins of (1) 0-9 years old (n=44), (2) 9-12 years old (n=44), (3) 12-15 years old (n=49), and (4) 15-16 years old (n=64). The breaks were selected to create bins of similar sizes. Each project was coded with a corresponding group label of one to four. This augmented formality measure is shown in Equation 2 and the resulting metric ranges between 0 and 10.

$$\text{Augmented Formality Score} = \frac{MMT}{MSE/AG} \tag{2}$$

To evaluate *concentration of developer responsibility* (H_B), we draw on a measure used in the overall formality score mentioned above: MMT, which represents the share of project developers who hold more responsibilities via their privileged roles in the engineering process, such as merge permissions. As seen in Equation 3, this metric is a weighted average of collaborators in a given FLOSS community, with the community defined as all individuals whose committed edits to project files have been accepted in the main branch [20], [21]. Collaborators are developers who have authored a merge into the primary code branch and thus have the requisite permissions to do so, contributors are commit authors who have never merged into the primary code branch [22]. Higher MMT averages may suggest flatter organizational structure, since it indicates widely diffused merge activities [37].

$$MMT = \frac{1}{|M|} \sum_{m \in M} \begin{cases} 2 & \text{If } m \text{ is a collaborator.} \\ 1 & \text{If } m \text{ is a contributor.} \end{cases} \tag{3}$$

We measure *formal work process management* (H_C) via projects' use of GitHub milestones [20], again drawing on the formality score. However, some projects in our dataset are hosted on platforms which lack the milestone metric or do not use it entirely, with prior work indicating only around 20% of projects on GitHub used the milestone tool [25]. Thus, we employed two metrics of milestone usage. One was the numeric count of milestones that a given project was using; the other was a binary classification of whether or not a project used milestones. Around 25% of packages studied in this data set used milestones. To measure the *underproduction factor* of a FLOSS project, we use the mean underproduction factor estimates published in Champion and

Hill [2]. As previously described in II, the underproduction metric measures a project's engineering activity against the project's importance.

D. Analytic Plan

Linear regression is an established approach of identifying associative relationships between two continuous variables. In our evaluation of the continuous measures of $\mathbf{H_A}$ (formality score), $\mathbf{H_B}$ (responsibility concentration), $\mathbf{H_C}$(work processes), and project age, we used linear regression models to evaluate the relationship with the project's underproduction factor and evaluate significance at the $p < .05$ level.

Given our constrained data sample, we also employed a statistical power analysis to evaluate the impact of data sample size on our results when relevant (H_A, H_C).

IV. Results

TABLE I
THIS TABLE DISPLAYS THE RELATIONSHIPS BETWEEN UNDERPRODUCTION AND THREE PROJECT METRICS: FORMALITY, MMT, AND MILESTONES. MODELS FOR H_B AND H_C INCLUDE A CONTROL FOR AGE WITH FACTOR VARIABLES; 0 TO 9 YEARS OLD IS THE BASELINE CATEGORY.

	(H_A) formality	(H_B) MMT	(H_C) milestones
(Intercept)	-0.78^*	1.65^*	-0.89^*
	$[-1.27; -0.29]$	$[0.06; 3.25]$	$[-1.32; -0.45]$
Augmented formality	0.17^*		
	$[0.05; 0.29]$		
MMT		-1.38^*	
		$[-2.21; -0.54]$	
Age 9-12y		0.07	0.27
		$[-0.53; 0.67]$	$[-0.33; 0.88]$
Age 12-15y		0.60^*	0.79^*
		$[0.01; 1.18]$	$[0.20; 1.38]$
Age 15-16y		1.15^*	1.53^*
		$[0.55; 1.74]$	$[0.96; 2.11]$
Milestone count			0.01
			$[-0.13; 0.15]$
R^2	0.04	0.22	0.17
Adj. R^2	0.04	0.20	0.15
Num. obs.	182	182	182

* Null hypothesis value outside the confidence interval.

A. H_A: Formality Score

We found a statistically significant relationship between a project's score and underproduction factor ($p < .005$). However, the relationship between formality score and underproduction was relatively small, where a unit increase in formality score was associated with only a 0.17 increase in underproduction factor. This result is evidence in favor of our hypothesis H_A, higher formality is associated with increased risk of underproduction.

The analysis reported in Table I uses the augmented formality calculation in Equation 2, which enabled us to include projects who did not use GitHub milestones, giving us a larger sample size (n=182).

Using the original formulation of the formality score in Equation 1 would have limited our analysis to projects whose milestone counts are greater than zero; the sample size for this model was 44. Testing H_A using this alternate metric found no statistically significant relationship between the score and a project's underproduction factor. Given our small sample, we conducted a power analysis to assess the impact of sample size in detecting an effect size of at least β = 0.00017 (this is the significant effect size of the augmented formality score, but scaled commensurately with the original score's range.) To simulate our data, we assumed that the formality score encapsulated other metrics such as MMT, milestones, and age and that formality score data follow a beta(α:1, β:3) distribution. After running 1000 simulations on data sets of size 75, we are able to reject the null hypothesis. The power analysis provides evidence that, to the extent that this simulated pilot data is representative of the population, our sample size is insufficient to conclude the original formality score's relationship to underproduction, and lends validity to our decision to use the augmented formality calculation displayed in Equation 2.

B. H_B: Concentration of Developer Responsibility

A linear regression model fit with MMT values indicated strong statistical significance for a negative relationship between MMT and mean underproduction values, suggesting that as the share of project developers who assume privileged roles (MMT) increases, the factor of underproduction decreases. The results are statistically significant ($p < .002$), where a unit increase in MMT is associated with a 1.38 point decrease in underproduction risk. This result is contrary to our hypothesis H_B, and instead provides evidence that increasing dispersion of responsibility is associated with lower underproduction risk, rather than higher. The effect size (-1.38) is relatively large, given that underproduction scores for our data only range between -5.05 and 2.81, suggesting that this association is also practically significant. Figure 1 displays the relationship between projects' MMT and mean underproduction factor.

C. H_C: Formal Work Process Management

A linear regression model fit with projects' mean underproduction scores and project usage of GitHub milestones did not point to a statistically significant relationship between the two measures ($p = 0.09$). However, as with formality score, a lack of significance may be due to our relatively small sample size (n = 182). Thus we conducted a power analysis simulation to assess whether we had sufficient observations to detect an effect size of at least β = 0.40; a 5% impact in underproduction factor within our data set. To simulate our data, we transposed MMT into a 0-1 range, modeled the measure's established relationship of -1.38 units of underproduction factor, and assumed MMT data follow a beta (α:5, β:1) distribution. From initial analysis of collected data, we also assumed that milestone data follow a binomial distribution with probability 0.247. Running 1000 simulations of data sets of n=300 we were able to reject the null hypothesis. This provides evidence that, to the extent that our pilot data and simulation are representative of the population, our sample

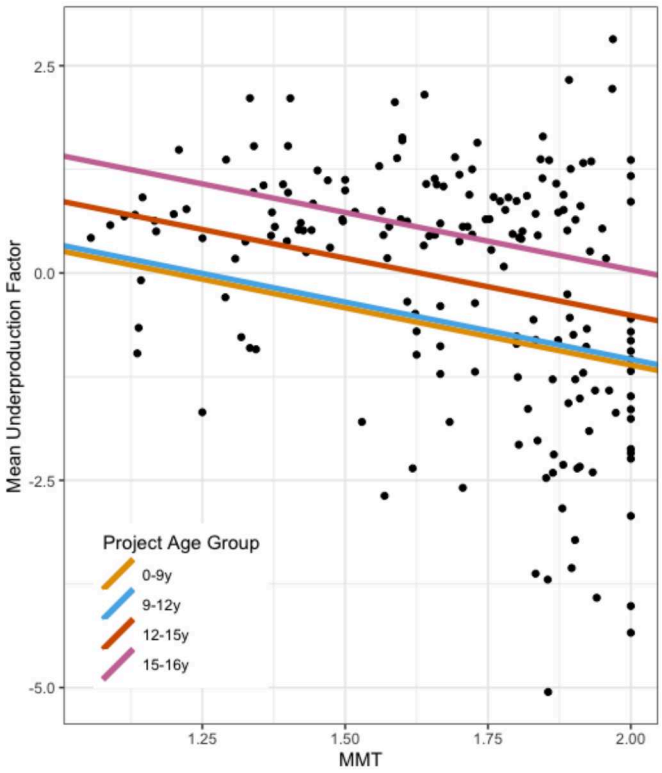

Fig. 1. Plot showing the relationship between projects' MMT and mean underproduction factor. Plotted lines are drawn from the results of our model for H_B.

size is insufficient to conclude there is no meaningful relationship between formal work process management (GitHub milestones) and underproduction.

V. Discussion

Overall, our results paint a nuanced picture of the relationship between formalization and underproduction in FLOSS projects. Our analysis of formal structure (H_A) using the augmented formality score showed that overall, more formal structures were associated with a minor increase in underproduction risk. These results support arguments from prior work from Tamburri et al. [20] and De Stefano et al. [23] that formal governance concentration is related to increased project risk.

However, our analysis of the distribution of privileged roles across developers (H_B, using MMT) showed that an increase in project developers with privileged roles is associated with decreased underproduction risk. Examining the relationship in Figure 1 suggests that this effect occurs when MMT is especially high, given the relatively flat relationship when MMT ranges between 0-1.75. One reason higher MMT may be associated with lower underproduction risk may be that the diffusion of privileged roles across more developers may indicate higher commitment to the project overall. For example, cursory analysis of project age shows a small positive relationship between age and a project's risk of underproduction. As a package ages, not only is there a higher likelihood of software adoption, there is also a higher likelihood of community abandonment. While we cannot make any causal claims, our results suggest the diffusion of project responsibility may help reduce risk of community abandonment and thus, risk of underproduction.

Finally, our analysis of formal work processes as captured by project milestone usage did not yield statistically significant results. However, our power analysis suggests that this was due to sample size, and we believe this is a valuable direction future work to better understand the effects of formalization.

VI. Limitations and Future Work

In this project, we only examine Python projects which are within the Debian distribution and for which an underproduction factor measure was available. Our analysis was further limited to projects which are hosted on platforms supporting our data collection approach (GitHub/GitLab); while these platforms are widely used hosting platforms for FLOSS projects, these limitations restricted our sample size (n=182). Although our work offers insight into risk around the widely-used Debian distribution, expanding this sample is an important direction of future work. Moreover, the metrics we employ do not account for longitudinal changes in project governance; this remains a topic of further research.

VII. Conclusion

FLOSS organizations face numerous challenges as they seek to mobilize volunteers to produce secure, high-quality software. We examined three hypotheses about the impact of formality on underproduction risk, finding that project formality may be a relevant indicator of higher software risk, that the diffusion of engineering responsibility was associated with lower risk, and that work product management is an uncertain predictor. Taken together, these results suggest governance informality and broader sharing of responsibility are beneficial to FLOSS projects. Given the importance of FLOSS across software ecosystems, unlocking the complex relationship between FLOSS governance and underproduction offers a new avenue for addressing the cybersecurity risks facing digital infrastructure.

Acknowledgment

This work is indebted to the volunteer developers producing FLOSS who have made their work available for inspection. We also gratefully acknowledge support from the Sloan Foundation through the Ford/Sloan Digital Infrastructure Initiative (Sloan Award 2018-113560 and the National Science Foundation (Grant IIS-2045055). This work was conducted using research computing resources at Northwestern University.

References

[1] K. Crowston, K. Wei, J. Howison, and A. Wiggins, "Free/Libre open-source software development: What we know and what we do not know," *ACM Computing Surveys*, vol. 44, no. 2, pp. 1–35, Feb. 2012.

[2] K. Champion and B. M. Hill, "Underproduction: An approach for measuring risk in open source software," in *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. Honolulu, HI, USA: IEEE, Mar. 2021, pp. 388–399. [Online]. Available: https://ieeexplore.ieee.org/document/9426043/

[3] J. Walden, "The Impact of a Major Security Event on an Open Source Project: The Case of OpenSSL," in *17th International Conference on Mining Software Repositories*, ser. MSR '20. New York, NY, USA: Association for Computing Machinery, Jun. 2020, pp. 409–419. [Online]. Available: http://doi.org/10.1145/3379597.3387465

[4] "Log4Shell 10 days later: Enterprises halfway through patching," Dec. 2021, publication Title: wiz.io. [Online]. Available: https://www.wiz.io/blog/10-days-later-enterprises-halfway-through-patching-log4shell

[5] "CISA Open Source Software Security Roadmap," Sep. 2023. [Online]. Available: https://www.cisa.gov/resources-tools/resources/cisa-open-source-software-security-roadmap

[6] P. Krill, "Report finds few open source projects actively maintained," *InfoWorld*, Oct. 2023. [Online]. Available: https://www.infoworld.com/article/3708630/report-finds-few-open-source-projects-actively-maintained.html

[7] Y. Benkler, *The wealth of networks: How social production transforms markets and freedom*. New Haven, CT: Yale University Press, 2006.

[8] ——, "Coase's penguin, or, Linux and the nature of the firm," *Yale Law Journal*, vol. 112, no. 3, pp. 369–446, 2002. [Online]. Available: http://www.jstor.org/stable/1562247

[9] ——, "Peer production, the commons, and the future of the firm," *Strategic Organization*, vol. 15, no. 2, pp. 264–274, May 2017. [Online]. Available: https://doi.org/10.1177/1476127016652606

[10] X. Yin and E. J. Zajac, "The strategy/governance structure fit relationship: theory and evidence in franchising arrangements," *Strategic Management Journal*, vol. 25, no. 4, pp. 365–383, 2004. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/smj.389

[11] J. W. Meyer and B. Rowan, "Institutionalized organizations: Formal structure as myth and ceremony," *American Journal of Sociology*, vol. 83, no. 2, pp. 340–363, Sep. 1977. [Online]. Available: http://www.jstor.org/stable/2778293

[12] K. Healy and A. Schussman, "The Ecology of Open-Source Software Development," 2003. [Online]. Available: https://www.semanticscholar.org/paper/The-Ecology-of-Open-Source-Software-Development-Healy-Schussman/9abecf8dfccfa21b20bb931fc63c161752b50181

[13] E. Ostrom, *Governing the Commons: The Evolution of Institutions for Collective Action*, ser. Canto Classics. Cambridge: Cambridge University Press, 2015. [Online]. Available: https://www.cambridge.org/core/books/governing-the-commons/A8BB63BC4A1433A50A3FB92EDBBB97D5

[14] S. Hwang and A. Shaw, "Rules and Rule-Making in the Five Largest Wikipedias," *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 16, pp. 347–357, May 2022. [Online]. Available: https://ojs.aaai.org/index.php/ICWSM/article/view/19297

[15] A. Shaw and B. M. Hill, "Laboratories of oligarchy? How the iron law extends to peer production," *Journal of Communication*, vol. 64, no. 2, pp. 215–238, 2014. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1111/jcom.12082/abstract

[16] P. Tourani, B. Adams, and A. Serebrenik, "Code of conduct in open source projects," in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Feb. 2017, pp. 24–33. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7884606

[17] P. B. de Laat, "Governance of open source software: state of the art," *Journal of Management & Governance*, vol. 11, no. 2, pp. 165–177, May 2007, tex.ids= noauthor_notitle_nodate-1. [Online]. Available: https://doi.org/10.1007/s10997-007-9022-9

[18] Y. Jiang, B. Adams, and D. M. German, "Will my patch make it? And how fast? Case study on the Linux kernel," in *2013 10th Working Conference on Mining Software Repositories (MSR)*, May 2013, pp. 101–110, iSSN: 2160-1860. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6624016

[19] S. Onoue, H. Hata, and K. Matsumoto, "Software population pyramids: the current and the future of OSS development communities," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '14. New York, NY, USA: Association for Computing Machinery, Sep. 2014, pp. 1–4. [Online]. Available: https://doi.org/10.1145/2652524.2652565

[20] D. A. Tamburri, P. Lago, and H. v. Vliet, "Organizational social structures for software engineering," *ACM Computing Surveys*, vol. 46, no. 1, pp. 3:1–3:35, Jul. 2013. [Online]. Available: https://dl.acm.org/doi/10.1145/2522968.2522971

[21] D. A. A. Tamburri, F. Palomba, and R. Kazman, "Exploring Community Smells in Open-Source: An Automated Approach," *IEEE Transactions on Software Engineering*, pp. 1–1, 2019.

[22] J. van Meijel, "On the Relations Between Community Patterns and Smells in Open-Source: A Taxonomic and Empirical Analysis," Master's thesis, Eindhoven University of Technology, Eindhoven, NL, Oct. 2021.

[23] M. De Stefano, E. Iannone, F. Pecorelli, and D. A. Tamburri, "Impacts of software community patterns on process and product: An empirical study," *Science of Computer Programming*, vol. 214, p. 102731, Feb. 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167642321001246

[24] K. Finley, "For Open Source, It's All About GitHub Now," *Wired*, Apr. 2019. [Online]. Available: https://www.wired.com/story/open-source-all-about-github-now/

[25] Y. Zhang, H. Wang, Y. Wu, D. Hu, and T. Wang, "GitHub's milestone tool: A mixed-methods analysis on its use," *Journal of Software: Evolution and Process*, vol. 32, no. 4, p. e2229, 2020. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.2229

[26] L. Yin, Z. Chen, Q. Xuan, and V. Filkov, "Sustainability forecasting for Apache incubator projects," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2021. New York, NY, USA: Association for Computing Machinery, Aug. 2021, pp. 1056–1067. [Online]. Available: https://dl.acm.org/doi/10.1145/3468264.3468563

[27] K. Crowston and J. Howison, "Assessing the health of open source communities," *IEEE Computer*, vol. 39, no. 5, pp. 89–91, 2006.

[28] L. Yin, M. Chakraborti, Y. Yan, C. Schweik, S. Frey, and V. Filkov, "Open Source Software Sustainability: Combining Institutional Analysis and Socio-Technical Networks," *Proceedings of the ACM on Human-Computer Interaction*, vol. 6, no. CSCW2, pp. 1–23, Nov. 2022. [Online]. Available: https://dl.acm.org/doi/10.1145/3555129

[29] S. Spaeth, M. Stuermer, S. Haefliger, and G. von Krogh, "Sampling in Open Source Software Development: The Case for Using the Debian GNU/Linux Distribution," in *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*. Waikoloa, HI: IEEE, Jan. 2007.

[30] M. O'Neil, *Cyberchiefs: autonomy and authority in online tribes*. London ; New York : New York: Pluto Press ; Distributed in the United States of America exclusively by Palgrave Macmillan, 2009.

[31] B. M. Sadowski, G. Sadowski-Rasters, and G. Duysters, "Transition of governance in a mature open software source community: Evidence from the Debian case," *Information Economics and Policy*, vol. 20, no. 4, pp. 323–332, Dec. 2008.

[32] R. Nguyen and R. Holt, "Life and death of software packages: An evolutionary study of Debian," in *Proceedings of the 2012 Conference of the Center for Advanced Studies on Collaborative Research*, ser. CASCON '12. Toronto, Ontario, Canada: IBM Corp., Nov. 2012, pp. 192–204.

[33] L. Nussbaum and S. Zacchiroli, "The Ultimate Debian Database: Consolidating bazaar metadata for Quality Assurance and data mining," in *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. Cape Town, South Africa: IEEE, May 2010, pp. 52–61.

[34] S. Dueñas, V. Cosentino, G. Robles, and J. M. Gonzalez-Barahona, "Perceval: software project data at your will," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. Gothenburg Sweden: ACM, May 2018, pp. 1–4. [Online]. Available: https://dl.acm.org/doi/10.1145/3183440.3183475

[35] G. Catolino, F. Palomba, and D. A. Tamburri, "The Secret Life of Software Communities: What we know and What we Don't know," *Proceedings of the 18th Belgium-Netherlands Software Evolution Workshop,*, 2019.

[36] W. Mauerer, M. Joblin, D. A. Tamburri, C. Paradis, R. Kazman, and S. Apel, "In Search of Socio-Technical Congruence: A Large-Scale Longitudinal Study," *IEEE Transactions on Software Engineering*, vol. 48, no. 8, pp. 3159–3184, Aug. 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9436025/

[37] G. Gousios, M. Pinzger, and A. v. Deursen, "An exploratory study of the pull-based software development model," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: Association for Computing Machinery, May 2014, pp. 345–355. [Online]. Available: https://dl.acm.org/doi/10.1145/2568225.2568260