Do All Software Projects Die When Not Maintained? Analyzing Developer Maintenance to Predict OSS Usage

Emily Nguyen
emilyngn@ucla.edu
University of California at Los Angeles
USA

ABSTRACT

Past research suggests software should be continuously maintained in order to remain useful in our digital society. To determine whether these studies on software evolution are supported in modern-day software libraries, we conduct a natural experiment on 26,050 GitHub repositories, statistically modeling library usage based on their package-level downloads against different factors related to project maintenance.

CCS CONCEPTS

• **Software and its engineering** → **Software libraries and repositories**.

KEYWORDS

Open Source, Open Source Sustainability, Survival Analysis

ACM Reference Format:
Emily Nguyen. 2023. Do All Software Projects Die When Not Maintained? Analyzing Developer Maintenance to Predict OSS Usage. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '23), December 3–9, 2023, San Francisco, CA, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3611643.3617849

1 INTRODUCTION

Open-source software (OSS) provides widely reusable infrastructure that our digital society relies on [9], but most of these systems rely on the execution of "mundane but necessary" tasks [16], generally defined as software maintenance [2, 3]. Much open-source maintenance is done by volunteers, and the process can be daunting, repetitively time-consuming, and insufficiently funded to take on [7, 25]. Research on open-source sustainability [5, 18, 24] addresses these factors, and we begin to see more projects are left unmaintained and abandoned by users and maintainers over time [7].

To preserve our modern software ecosystems, we aim to see how open-source users react to projects that are less maintained over time. Lehman's laws of software evolution suggests the idea of Continuing Change [12], arguing that a software will progressively become less satisfying to its users over time unless it is continually adapted to meet new needs [13, 19, 27]. This was supported in industrial agile products [15, 26] and open source Java programs [1] through empirical analysis of program evolution. In other studies, users admitted switching to different libraries after maintainers stopped updating project dependencies as well [17, 28].

While project maintenance seems intuitively vital, some dependencies exist that do not require continuous maintenance because the software is "feature-complete" [20].

To understand the dynamics between open-source maintenance and project usage, we ask: **Is there a relationship between a project's maintenance characteristics and the downstream usage of that project?** We perform a large-scale quantitative analysis of 26,050 popular GitHub repositories to identify the degree maintainer disengagement associates with low project usage. We obtain data on package-level downloads from the npm registry as a way to measure project usage. In addition to project download data we assembled a longitudinal data set of their activity metrics, developed an automated heuristic to detect decreased usage in open-source projects, and estimated Cox proportional hazards survival regressions to model what factors of maintenance affect packages' chances of losing users.

2 RESEARCH METHODS

To answer our research question, we use Cox proportional hazards survival analysis [10] to model the dependent outcome of whether project usage declines based on independent variables related to maintenance and other project characteristics. Cox proportional hazards is an extension of original survival analysis methods such as Kaplan-Meier survival curves [21] and log-rank tests [4]. While Kaplan-Meier models the survival probability past a certain time, and log-rank tests assess statistical significance without capturing effect size [23], Cox proportional hazards models a hazard function to assess the effects of multiple quantitative risk factors on the survival time before an event of interest [11]. In our case the event is the date when the package began to significantly lose downloads, with risks of low maintenance under consideration. To encode our outcome of project usage we built a human-validated automated heuristic to detect projects that experienced a *peak* in longitudinal download counts followed by a significant decline, which we refer to as *peaked projects*.

Data set. We restricted our analysis to packages that were popular at least once in their history (i.e. packages that received at least 10,000 downloads and received at least 1 commit or opened/closed issue during any month in our 2015-01 to 2020-12 observation window). From that collection of 37,984 packages we identified 13,378 (35.22%) that *peaked*. To establish a control group of projects with similar popularity we used the matching-pairs technique [6] to identify a matching package that did not *peak* for each of the 13,378 projects. Each *peaked project* is paired with a *non-peaked project*

ESEC/FSE '23, December 3–9, 2023, San Francisco, CA, USA

ACM ISBN 979-8-4007-0327-0/23/12.
https://doi.org/10.1145/3611643.3617849

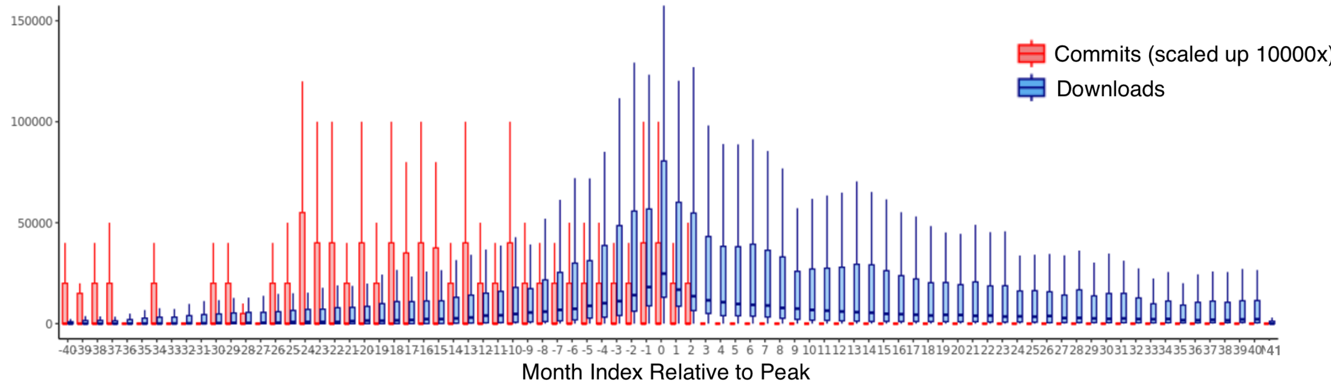

Figure 1: Distribution of package downloads and commits for *peaked projects* during every month relative to *peak*.

that contained between 0.75- to 1.25-times the number of downloads during the month of the *peaked project's* maximum downloads. Then, because we want to study how various maintenance factors contribute to explaining the variability in package usage over time, we excluded packages that *peaked* within the first early 6 months of the observation window and their matched-set pairs from our sample. Thus, our final sample contains 26,050 total projects (13,025 *peaked* and 13,025 *non-peaked*) to be used in our Cox model.

Operationalization of Peaked Usage for Model. Before creating and encoding our heuristic, we established a common-ground understanding of what represented *peaked* usage in downloads. Across 3 rounds 4-5 researchers were prompted to individually evaluate whether or not each package's download trends in a random sample of 100 projects showed a generally declining trend in downloads following a higher point in the past (i.e. *peaked*). To measure their degree of agreement, we then conducted inter-rater reliability tests using the Fleiss' kappa [14], confirming a *strong* level of agreement [8] with an average Fleiss score of 0.855.

We then designed a heuristic to automatically encapsulate the generalized patterns observed by humans. The heuristic checks for a 20% decrease from the maximum number of downloads to the number of downloads during the last month in our observation window to classify projects as *peaked*.

Collecting Independent Factors. As independent variables we operationalized maintenance as the number of commits, number of opened issues, number of closed issues, number of contributors, and if the repository was archived as read-only. These activity metrics gauge the package's overall maintainer and user engagement. We additionally added the package size in kilobytes as a variable to explore any relationship between small packages and widely-used feature-complete software. Details on how these metrics were obtained is in http://github.com/Enemily/Open-Source-Downloads.

3 RESULTS

The risk of packages severely losing downloads (i.e. *peaking*) decreases when they are more consistently maintained. Figure 1 shows relatively stable maintenance in commits leading up to the peak, followed by downloads peaking at the same time commits begin to drastically decrease. From the regression coefficients of our Cox model, the number of commits by contributors had a coefficient of -0.04, giving a hazard ratio [22] of $e^{-0.04} = 0.96$ and implying for every 1-unit increase in the number of contributor commits, the risk of downloading *peaking* decreases by 4%. Maintainer engagement through closing issues also decreases the risk of *peaking* by 6%, and archived repositories that become read-only increase the risks of *peaking* by 37%. Additionally, the effects of no maintenance, particularly read-only archived projects, on package downloads decreases for smaller projects whose size in kilobytes during the *peak date* (or last month of observation window for *non-peaked* projects) is less than the median size of all packages in our sample during their respective *peak date* (or last month for *non-peaked*).

Discussion. To some extent, the results from our longitudinal analyses support Lehman's lemma that projects progressively become less satisfactory unless continually adapted. While most maintenance factors agreed with Lehman, the relationship between software maintenance and usage levels was not entirely supported for smaller packages, as expected of small and widely-reused projects to be feature-complete and less dependent on maintainers to thrive.

The relevance of a project may influence maintenance levels, as developers prefer investing maintenance into popular projects that attract more clients. Hence, a project's declining popularity could *cause* a decline in maintenance activity and should be considered in addition to our results. Additionally, some studies define dormant repositories as those with less than 1 average commit every 4 consecutive quarters. Different operationalizations between other studies and ours could suggest bias and serve a threat to our results.

4 CONTRIBUTIONS

From our natural experiment on 26,050 packages, we can support conjectures of open-source-sustainability research that maintenance is important, but there is also nuance in that not all projects need to be consistently maintained and a lack of maintenance is not automatically a problem. We hope our empirical findings help practitioners and researchers better understand the components that drive open-source usability and sustainability as a whole.

ACKNOWLEDGMENTS

This work was supported through Carnegie Mellon University's REU in Software Engineering (NSF Award 1901311).

REFERENCES

[1] Mamdouh Alenezi and Khaled Almustafa. 2015. Empirical analysis of the complexity evolution in open-source software systems. *International Journal of Hybrid Information Technology* 8, 2 (2015), 257–266. https://doi.org/10.14257/ijhit.2015.8.2.24

[2] Lowell Jay Arthur. 1988. *Software evolution: the software maintenance challenge.* Wiley-Interscience.

[3] Keith H Bennett and Václav T Rajlich. 2000. Software maintenance and evolution: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering.* 73–87. https://doi.org/10.1145/336512.336534

[4] J Martin Bland and Douglas G Altman. 2004. The logrank test. *Bmj* 328, 7447 (2004), 1073. https://doi.org/10.1136/bmj.328.7447.1073

[5] InduShobha Chengalur-Smith, Anna Sidorova, and Sherae L Daniel. 2010. Sustainability of free/libre open source projects: A longitudinal study. *Journal of the Association for Information Systems* 11, 11 (2010), 5. https://doi.org/10.17705/1jais.00244

[6] Peter Christen and Peter Christen. 2012. *The data matching process.* Springer. https://doi.org/10.1007/978-3-642-31164-2_2

[7] Jailton Coelho and Marco Tulio Valente. 2017. Why modern open source projects fail. In *Proceedings of the 2017 11th Joint meeting on foundations of software engineering.* 186–196. https://doi.org/10.1145/3106237.3106246

[8] Kevin Crowston and James Howison. 2016. FLOSS Project Effectiveness Measures1. *Successful OSS project design and implementation: requirements, tools, social designs and reward structures* (2016), 149. https://doi.org/10.1108/14684521211240207

[9] Nadia Eghbal. 2016. Roads and bridges. *The Unseen labor behind our digital infrastructure* (2016). https://doi.org/10.48558/9qrg-bh51

[10] John Fox and Sanford Weisberg. 2002. Cox proportional-hazards regression for survival data. *An R and S-PLUS companion to applied regression* 2002 (2002).

[11] Brandon George, Samantha Seals, and Inmaculada Aban. 2014. Survival analysis and regression models. *Journal of nuclear cardiology* 21 (2014), 686–694. https://doi.org/10.1007/s12350-014-9908-2

[12] Michael W Godfrey and Daniel M German. 2014. On the evolution of Lehman's Laws. *Journal of Software: Evolution and Process* 26, 7 (2014), 613–619. https://doi.org/10.1002/smr.1636

[13] Jesus M Gonzalez-Barahona, Gregorio Robles, Israel Herraiz, and Felipe Ortega. 2014. Studying the laws of software evolution in a long-lived FLOSS project. *Journal of Software: Evolution and Process* 26, 7 (2014), 589–612. https://doi.org/0.1002/smr.1615

[14] Menelaos Konstantinidis, Lisa W Le, and Xin Gao. 2022. An empirical comparative assessment of inter-rater agreement of binary outcomes and multiple raters. *Symmetry* 14, 2 (2022), 262. https://doi.org/10.3390/sym14020262

[15] Gurpreet Kour and Paramvir Singh. 2016. Using Lehman's laws to validate the software evolution of agile projects. In *2016 International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT).* IEEE, 90–96. https://doi.org/10.1109/ICCTICT.2016.7514558

[16] Karim R Lakhani and Eric Von Hippel. 2004. How open source software works:"free" user-to-user assistance. *Produktentwicklung mit virtuellen Communities: Kundenwünsche erfahren und Innovationen realisieren* (2004), 303–339. https://doi.org/10.1007/978-3-322-84540-5_13

[17] Courtney Miller, Christian Kästner, and Bogdan Vasilescu. 2023. "We Feel Like We're Winging It:" A Study on Navigating Open-Source Dependency Abandonment. In *Proceedings of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)* (San Francisco, CA). ACM Press, New York, NY.

[18] Courtney Miller, Paige Rodeghero, Margaret-Anne Storey, Denae Ford, and Thomas Zimmermann. 2021. " how was your weekend?" software development teams working from home during covid-19. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE).* IEEE, 624–636. https://doi.org/10.1109/ICSE43902.2021.00064

[19] Ramin Moazeni, Daniel Link, and Barry Boehm. 2013. Lehman's laws and the productivity of increments: Implications for productivity. In *2013 20th Asia-Pacific Software Engineering Conference (APSEC)*, Vol. 1. IEEE, 577–582. https://doi.org/10.1109/APSEC.2013.84

[20] Audris Mockus, David M Weiss, and Ping Zhang. 2003. Understanding and predicting effort in software projects. In *25th International Conference on Software Engineering, 2003. Proceedings.* IEEE, 274–284. https://doi.org/10.1109/ICSE.2003.1201207

[21] Jason T Rich, J Gail Neely, Randal C Paniello, Courtney CJ Voelker, Brian Nussenbaum, and Eric W Wang. 2010. A practical guide to understanding Kaplan-Meier curves. *Otolaryngology—Head and Neck Surgery* 143, 3 (2010), 331–336. https://doi.org/10.1016/j.otohns.2010.05.007

[22] Michael Schemper, Samo Wakounig, and Georg Heinze. 2009. The estimation of average hazard ratios by weighted Cox regression. *Statistics in medicine* 28, 19 (2009), 2473–2489. https://doi.org/10.1002/sim.3623

[23] Patrick Schober and Thomas R Vetter. 2021. Kaplan-meier curves, log-rank tests, and cox regression for time-to-event data. *Anesthesia & Analgesia* 132, 4 (2021), 969–970. https://doi.org/10.1213/ANE.0000000000005358

[24] Kimberly Truong, Courtney Miller, Bogdan Vasilescu, and Christian Kästner. 2022. The unsolvable problem or the unheard answer? a dataset of 24,669 open-source software conference talks. In *Proceedings of the 19th International Conference on Mining Software Repositories.* 348–352. https://doi.org/10.1145/3524842.3528488

[25] Qiang Tu et al. 2000. Evolution in open source software: A case study. In *Proceedings 2000 International Conference on Software Maintenance.* IEEE, 131–142. https://doi.org/10.1109/ICSM.2000.883030

[26] Jilles van Gurp, Christian Prehofer, and Jan Bosch. 2010. Comparing practices for reuse in integration-oriented software product lines and large open source software projects. *Software: Practice and Experience* 40, 4 (2010), 285–312. https://doi.org/10.1002/spe.955

[27] Liguo Yu and Alok Mishra. 2013. An empirical study of Lehman's law on software quality evolution. (2013).

[28] Markus Zimmermann, Cristian-Alexandru Staicu, Cam Tenny, and Michael Pradel. 2019. Small World with High Risks: A Study of Security Threats in the npm Ecosystem. In *USENIX security symposium*, Vol. 17.

Received 2023-06-05; accepted 2023-08-11