Snapshot Metrics Are Not Enough: Analyzing Software Repositories with Longitudinal Metrics

Nicholas Synovic
Loyola University Chicago
Chicago, IL, USA

Matt Hyatt
Loyola University Chicago
Chicago, IL, USA

Rohan Sethi
Loyola University Chicago
Chicago, IL, USA

Sohini Thota
Loyola University Chicago
Chicago, IL, USA

Shilpika
University of California at Davis
Davis, CA, USA

Allan J. Miller
Loyola University Chicago
Chicago, IL, USA

Wenxin Jiang
Purdue University
West Lafayette, IN, USA

Emmanuel S. Amobi
Loyola University Chicago
Chicago, IL, USA

Austin Pinderski
Loyola University Chicago
Chicago, IL, USA

Konstantin Läufer
Loyola University Chicago
Chicago, IL, USA

Nicholas J. Hayward
Loyola University Chicago
Chicago, IL, USA

Neil Klingensmith
Loyola University Chicago
Chicago, IL, USA

James C. Davis
Purdue University
West Lafayette, IN, USA

George K. Thiruvathukal
Loyola University Chicago
Chicago, IL, USA

ABSTRACT

Software metrics capture information about software development processes and products. These metrics support decision-making, e.g., in team management or dependency selection. However, existing metrics tools measure only a snapshot of a software project. Little attention has been given to enabling engineers to reason about metric trends over time—longitudinal metrics that give insight about process, not just product. In this work, we present *PRIME* (PRocess MEtrics), a tool to compute and visualize process metrics. The currently-supported metrics include productivity, issue density, issue spoilage, and bus factor. We illustrate the value of longitudinal data and conclude with a research agenda. The tool's demo video can be watched at https://bit.ly/ase2022-prime. Source code can be found at https://github.com/SoftwareSystemsLaboratory/prime.

CCS CONCEPTS

• **Software and its engineering**; • **General and reference** → **Metrics**;

KEYWORDS

Software metrics; Empirical software engineering

ASE '22, October 10–14, 2022, Rochester, MI, USA

ACM ISBN 978-1-4503-9475-8/22/10…$15.00
https://doi.org/10.1145/3551349.3559517

ACM Reference Format:
Nicholas Synovic, Matt Hyatt, Rohan Sethi, Sohini Thota, Shilpika, Allan J. Miller, Wenxin Jiang, Emmanuel S. Amobi, Austin Pinderski, Konstantin Läufer, Nicholas J. Hayward, Neil Klingensmith, James C. Davis, and George K. Thiruvathukal. 2022. Snapshot Metrics Are Not Enough: Analyzing Software Repositories with Longitudinal Metrics. In *37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22), October 10–14, 2022, Rochester, MI, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3551349.3559517

1 INTRODUCTION

An effective software engineering process is correlated with high software quality [18]. Measurements of software processes therefore give engineers insight into software quality [7]. Software metrics characterize the software engineering process (*e.g.,* time to fix a defect) and the engineered product (*e.g.,* cyclomatic complexity). Using software metrics, engineers and managers may improve products and assess the risks of external software dependencies.

Tools for software metrics typically provide metrics on the current project state, or "snapshot metrics," rather than longitudinal metrics (§2). While a snapshot can be useful—for example, it can quickly reveal if a project has no test suite—it does not provide a full picture of the longitudinal evolution of a software project. *We conjecture that engineers will make different decisions when presented with snapshot metrics compared to longitudinal metrics (§5).*

To evaluate a development process, one needs to measure the history of the code. The classic Fenton & Bieman reference on software metrics [7] establishes that measurement needs to be related to a time range and scale for a meaningful longitudinal assessment of software quality. Tools that measure quality need to calculate both direct measurements and derived calculations at consistent intervals to evaluate the process properly. Trends in metrics can quantify software engineering decisions.

To support our investigation of this research question, we present *PRIME* [12] (PRocess MEtrics): an open-source tool that enables engineers and researchers to analyze software projects with longitudinal metrics. *PRIME* uses a modular Extract-Transform-Load (ETL) pipeline architecture for ease of adoption and extension (§3), *PRIME* currently supports the following metrics: code size, productivity, bus factor, issue count, issue spoilage, and issue density (§4).

We close by proposing three studies facilitated by *PRIME* (§6): (1) exploring engineers' use of longitudinal metrics when assessing their products; (2) exploring their use of longitudinal metrics during dependency selection; and (3) analyzing the software supply chain to identify potential weak links.

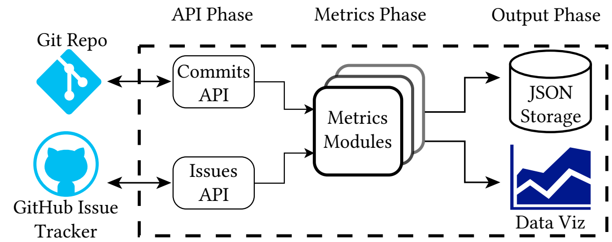

Figure 1: System architecture of *PRIME*.

2 BACKGROUND AND RELATED WORK

Process metrics are critical for improving software quality as agile repositories may eventually become more established and require regular maintenance. Although numerous efforts have focused on mining open-source repositories, the current support for process metrics—and visualizing them longitudinally—is mixed. In our survey of related efforts, we identified various tool types, including scorecards, frameworks, dashboards, and platform monitors.

Scorecards assign a risk score for open source projects to assess security risks and project health [3]. However, they are computed as a snapshot metric and cannot easily express longitudinal effects.

Frameworks simplify the process of developing tools for mining software repositories (MSR). Frameworks are typically libraries and domain-specific languages (DSL) that researchers and engineers integrate into their tools. The ISHEPARD/PYDRILLER [19] library and the BOA [5] DSL meet this criterion. These frameworks are not ready-to-use MSR tools but provide building blocks for developing new MSR tools for the analysis of version control systems (VCS).

Dashboards are built into online VCS platforms and visualize repository and issue tracker trends. GitHub Insights [2] and GitLab Insights [8] provide longitudinal metrics for hosted projects. However, these tools provide limited insights when it comes to process metrics but can be expanded upon by the community [9].

Platform monitors are third-party analysis tools that compute metrics for hosted packages. NPM [14] provides the NPM Search [15] analyzer for JavaScript packages, which tracks process metrics regarding issue trackers. The GoReportCard [10] is a monitor for Go projects hosted on GitHub, which tracks code metrics. Aside from dashboards, these tools compute process metrics as snapshots and do not make longitudinal and trends visualization easy for users.

3 ARCHITECTURE

PRIME follows an Extract, Transform, Load (ETL) architecture (Figure 1). The ETL phases of the pipeline are each module or collection of modules. In addition, the extraction and transformation stages of the pipeline store data in text-encoded JSON files. By storing measurements in a file rather than in memory during pipeline execution, *PRIME* can be integrated with existing tools and pipelines.

PRIME extracts base measurements from a project's version control system (VCS) and issue tracker during the **API Phase**. Here, using the external CLOC [1] and SLOCCOUNT [17] utilities, *PRIME* measures each commit of a repository and measures the size of the repository in lines of code (LOC), thousands of lines of code (KLOC), and the size difference between each sequential commit as the delta thousands of lines of code (DKLOC). *PRIME* also extracts issue report metadata by utilizing the REST API of a repository's host issue tracker.

PRIME transforms the extracted base measurements into derived metrics during its **Metrics Phase**. At the moment, *PRIME* can compute the following metrics: *issue spoilage*, *issue/defect density*, *productivity*, and *bus factor*, which we will define below. Each metric module takes in a text-encoded JSON file containing the base measurements for commits, issues, or both.

After both the API and Metrics phases, data is loaded into either text-encoded JSON files or visualized with MatPlotLib [11] in the **Output Phase**. *PRIME* can export the JSON and visualization files to integrate with other pipelines. Additionally, the visualizations can be customized using style sheets, thereby allowing engineering teams to implement style standards for their visualizations.

The ETL architecture allows engineers to use individual *PRIME* modules for the metrics of interest. Furthermore, each phase of the pipeline is configurable, reducing the time engineering teams need to post-process the data to match their specific needs. Finally, *PRIME* can be run on private repositories without exposing any data or metrics charts for any given project.

4 METRICS IMPLEMENTED

To address the limitations of existing tools, *PRIME* computes *longitudinal process metrics*. We chose the current set of metrics by their ability to provide insights into the development process as well as their ability to compute derived metrics. A prior survey informs our choice of these metrics [6], where research software engineers indicated that process metrics can be helpful. *PRIME* computes two types of software metrics: (1) *Direct metrics*, which are measurements of internal attributes of the process, and (2) *derived metrics*, which are computed metrics from two or more direct metrics.

4.1 Direct Metrics

Direct metrics are measurements of a particular attribute of the process involving no other attribute [7]. These measurements are the foundation for the more complex metrics that *PRIME* computes.

1. Code Size: *PRIME* measures the size of a repository in terms of the number of source lines of code normalized by 1000 reported as KLOC. Changes in the KLOC (DKLOC) show the growth (or shrinkage) of a repository over time.

2. Developer Count: *PRIME* measures this metric as the number of unique developers who contribute code to a repository within a time interval. By measuring developer count, engineering teams

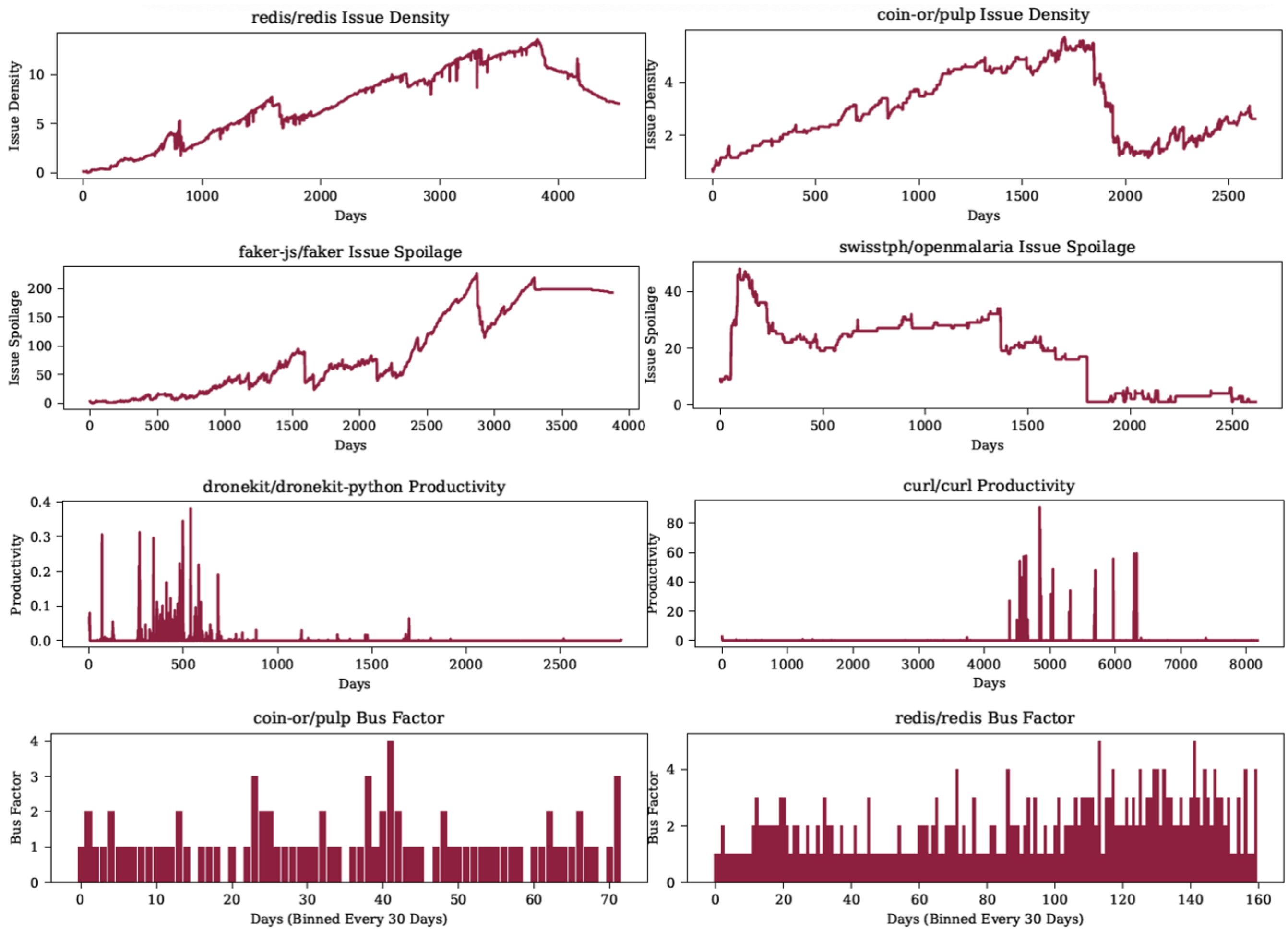

Figure 2: This figure shows the *PRIME* tool's output for each supported longitudinal derived process metric applied to several sample projects. The first pair depicts contrasting issue densities. The second pair depicts two projects with contrasting trends in resolving issues. The third pair depicts two projects with contrasting productivity trends. The fourth pair depicts two projects with contrasting bus factor binned to measure the number of core contributors each month.

can determine the amount of developer support in contributing new code, maintaining existing code, and resolving bugs.

3. Issue Count: *PRIME* measures this as the count of the number of open and closed *issues* reported in an issue tracker, including feature requests, tasks, and bug reports, in addition to potential and confirmed defects. If an online VCS has an issue tracker, this metric also reports the count of open and closed pull requests.

4.2 Derived Metrics

Derived metrics capture interactions between direct metrics [7]. *PRIME* computes derived metrics to analyze and subsequently visualize changes in the development process of a software product.

1. Issue Density: This metric tracks a project's total number of issues normalized by project size. Because we are interested in open-source repositories on GitHub, we use the more general issue density rather than defect density, which refers only to the ratio of bug count to repository size. A high issue density, regardless of confirmed defects, could signify an unhealthy repository. For example, if there are many feature requests that are never acted upon, then the development team is not implementing the features that users want. This would be a possible warning sign for poor customer support and, eventually, would lead to low customer or user satisfaction [16].

2. Issue Spoilage: Issue spoilage is the weighted average age of unresolved issues at a given time in the project timeline. With further analysis, this metric calculates the age of issues with respect to the project timeline to measure how quickly a project's team resolves issues. Issue spoilage can serve as a gauge of customer support and the efficiency of software teams in resolving issues. For instance, if issue spoilage increases in a time interval, new issues are being created faster than the team can resolve old ones. On the other hand, if the issue spoilage drops in a time interval, the team resolves previous issues faster than new ones are created.

3. Productivity: Productivity measures the rate at which a development team adds KLOC within a time interval [7]. Healthy repositories will typically have high productivity. However, low productivity is not always a sign of a lack of productiveness, as when efficient development teams are refactoring code KLOC may not change significantly.

4. Bus Factor: Bus factor [4] is the number of developers on a project team who would have to be "hit by a bus" to cause the project to fail. This metric measures the employee turnover risk of a project. However, as our work focuses on open-source projects, we propose that this is a metric of the development community's interest as well. By analyzing bus factor longitudinally, users gain insight into potential risks of the software development process. While bus factor is not a classical process metric, it is well known in the general SE literature that under-resourced projects carry a high risk of falling out of maintenance [7].

5 DEMONSTRATION

Figure 2 shows all four process metrics for several repositories over their entire project history. We chose projects from the RepoReapers/reaper data set [13] in pairs that showed contrasting trends in their process metrics to demonstrate possible insights from longitudinal analysis. We have organized this figure to demonstrate the potential for comparative analysis of process effectiveness, even among projects that have a good score using existing *scorecard* apps. The addition of process metrics clearly demonstrates that all of these otherwise good projects may benefit from further examining their development process. This examination is especially prudent when it comes to managing development while addressing issues (issue density), addressing issues (issue spoilage),ensuring appropriate resources (bus factor), or managing group priorities to avoid team burnout (productivity).

6 PLANNED STUDIES

In the first study, we pose the research question: *How do engineers use longitudinal process metrics during their development process?* We hypothesize that basic metrics are used in many open-source projects today, but the use of longitudinal metrics, particularly process metrics, is limited. To perform this study, we will measure the number of process metrics utilized and survey open-source developers on established projects about why and how they use these metrics in their development process.

In a second study, we pose the research question: *Do longitudinal metrics contribute to selecting dependencies in software composition?* Based on our survey of tools, we hypothesize that engineers take little consideration of derived longitudinal process metrics but will consider direct longitudinal process metrics as those are more prevalent when selecting dependencies for software development. To perform this study, we intend to survey the current state of software metrics tooling, and survey open-source engineers about their utilization of longitudinal process metrics for dependency selection.

In our third study, we pose the research question: *What role can longitudinal process metrics play in analyzing dependencies in open-source software?* We hypothesize that many projects are likely to depend on other projects that require process improvement, *e.g.,* a third-party library with a risky bus factor. To perform this study, we will examine the *dependencies* of well-known projects by using *PRIME* to analyze each of the dependent projects for process-related concerns. With *PRIME*, we can autonomously and automatically compute the longitudinal metrics that are of concern to our study.

7 ACKNOWLEDGMENTS

Davis acknowledges support from NSF OAC-2107230; Thiruvathukal acknowledges NSF OAC-2107020 and NSF OAC-1445347; Davis and Thiruvathukal acknowledge the Google TensorFlow Model Garden.

8 CONCLUSION

PRIME is an ongoing development effort to understand process effectiveness beyond snapshots of process metrics and support more longitudinal analysis and visualization. This paper demonstrates working software to compute four process metrics, which represent classical (*e.g.,* issue density, issue spoilage, productivity) and modern/agile (*e.g.,* bus factor) metrics. We argue for the potential of these tools to support future planned studies by showing their ability to visualize long and short-term trends via simple and intuitive charts. Future development efforts will include expanding *PRIME* with support for more process metrics, emphasizing comparative visualizations, and expanding the number of data sources. Future studies will build on this foundation to study the usage of longitudinal metrics in practice, longitudinal metrics in selecting dependencies, and the software supply chain.

REFERENCES

[1] cloc Contributors. 2021. AlDanial/cloc: 1.92. https://doi.org/10.5281/zenodo.5760077
[2] GitHub Insights Contributors. 2022. GitHub Insights for Projects. https://docs.github.com/en/issues/planning-and-tracking-with-projects/viewing-insights-from-your-project/about-insights-for-projects
[3] Scorecard Contributors. 2022. Security Scorecards. https://github.com/ossf/scorecard original-date: 2020-10-09T14:48:27Z.
[4] Valerio Cosentino, Javier Luis Cánovas Izquierdo, and Jordi Cabot. 2015. Assessing the bus factor of Git repositories. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. SANER, Unknown, 499–503. https://doi.org/10.1109/SANER.2015.7081864 ISSN: 1534-5351.
[5] Robert Dyer, Hoan Anh Nguyen, Hridesh Rajan, and Tien N. Nguyen. 2013. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In *2013 35th International Conference on Software Engineering (ICSE)*. ACM, Unknown, 422–431. https://doi.org/10.1109/ICSE.2013.6606588 ISSN: 1558-1225.
[6] Nasir U. Eisty, George K. Thiruvathukal, and Jeffrey C. Carver. 2018. A Survey of Software Metric Use in Research Software Development. In *2018 IEEE 14th International Conference on e-Science (e-Science)*. IEEE, Amsterdam, 212–222. https://doi.org/10.1109/eScience.2018.00036
[7] Norman Fenton and James Bieman. 2014. *Software Metrics: A Rigorous and Practical Approach, Third Edition* (3rd edition ed.). CRC Press, Boca Raton.
[8] GitLab. 2019. GitLab Insights Documentation. https://docs.gitlab.com/ee/user/project/insights
[9] GitLab. 2020. GitLab Insights Video. https://www.youtube.com/watch?v=OMTfPsLa98I
[10] Go Report Card Contributors. 2022. Go Report Card. https://goreportcard.com/
[11] John D. Hunter. 2007. Matplotlib: A 2D Graphics Environment. *Computing in Science Engineering* 9, 3 (May 2007), 90–95. https://doi.org/10.1109/MCSE.2007.55 Conference Name: Computing in Science Engineering.
[12] Matt Hyatt, Amy Kuhl, Jake Palmer, Rohan Sethi, Ethan Stoneman, Nicholas Synovic, Sohini Thota, and George K. Thiruvathukal. 2022. *clime-metrics*. Software and Systems Laboratory. https://doi.org/10.5281/zenodo.6587880
[13] Nuthan Munaiah, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. 2017. Curating GitHub for engineered software projects. *Empirical Software Engineering* 22, 6 (Dec. 2017), 3219–3253. https://doi.org/10.1007/s10664-017-9512-6
[14] npm contributors. 2022. npm. https://www.npmjs.com/
[15] npms.io contributors. 2018. npms. https://npms.io/
[16] William Scherkenbach. 2011. *The Deming Route to Quality and Productivity*. WWS, Inc., Unknown.
[17] SLOCCount Contributors. 2016. SLOCCount. https://dwheeler.com/sloccount/
[18] Ian Sommerville. 2015. Software engineering 10th Edition. *ISBN-10* 137035152 (2015), 18.
[19] Davide Spadini, Maurício Aniche, and Alberto Bacchelli. 2018. PyDriller: Python framework for mining software repositories. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, Lake Buena Vista FL USA, 908–911. https://doi.org/10.1145/3236024.3264598