GitRank: A Framework to Rank GitHub Repositories

Niranjan Hasabnis
Intel Labs
Santa Clara, California, USA
niranjan.hasabnis@intel.com

ABSTRACT

Open-source repositories provide wealth of information and are increasingly being used to build artificial intelligence (AI) based systems to solve problems in software engineering. Open-source repositories could be of varying quality levels, and bad-quality repositories could degrade performance of these systems. Evaluating quality of open-source repositories, which is not available directly on code hosting sites such as GitHub, is thus important. In this hackathon, we utilize known code quality measures and GrimoireLab toolkit to implement a framework, named GitRank, to rank open-source repositories on three different criteria. We discuss our findings and preliminary evaluation in this hackathon report.

ACM Reference Format:
Niranjan Hasabnis. 2022. GitRank: A Framework to Rank GitHub Repositories. In 19th International Conference on Mining Software Repositories (MSR '22), May 23–24, 2022, Pittsburgh, PA, USA. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3524842.3528519

1 INTRODUCTION

Open-source code repositories, such as those hosted on popular hosting sites like GitHub, provide wealth of information about software projects such as team dynamics, project schedule and deliverables, among others [1, 2, 4, 22]. Such information is routinely used by researchers working on problems in the field of software engineering. More recently, researchers working at the intersection of machine learning (or AI) and software engineering are also using such information to build systems that automatically learn to solve problems from data [6, 11–13, 18, 27, 33, 34].

Unfortunately, deriving insights using open-source code repositories could be wrong or misleading because popular hosting sites such as GitHub typically do not indicate quality of code repositories. This could partly be because quality of a repository could be subjective. Nonetheless, measuring code quality a well-researched problem in the field of software engineering, and several popular source code metrics such as Halstead volume [10], cyclomatic complexity [21], maintainability index [25], etc., exists [17, 19, 20].

Although, a number of frameworks and source code metrics to measure repository/code quality exist [5, 16, 17, 28–30], we could not find publicly-available, open-source implementation of a framework that ranks code repositories on quality. That is why we decided to build such a framework, named GitRank1, in this hackathon. Additionally, GrimoireLab toolkit [3, 9] that is available as a part of this hackathon incentivized such an implementation as it makes retrieving some of the source code metrics easy. We, nonetheless, faced several challenges, which we discuss in this report.

2 GITRANK: DESIGN AND IMPLEMENTATION

We now describe our framework to rank open-source repositories based on quality, maintainability, and popularity. In addition to source code quality metrics, GitRank also considers popularity and maintainability metrics (similar to NPM registry [24]). GitRank currently supports ranking of repositories that use C/C++ as their primary language. Nonetheless, the design of our framework is easy to extend to other languages as well. GitRank is implemented in 700 lines of Python code and can be broken down into two phases.

Phase 1: Obtaining values of metrics for every repository

Given a set of open-source repositories to be ranked, in the first phase, we obtain values of quality, maintainability, and popularity metrics of every repository individually. Table 1 contains the set of measures that we consider. We use a subset of the typical code measures used in software engineering that are applicable to open-source repositories and are common across a set of languages.

| Category       | Description of the measure                                                                                                                                 |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| Quality        | Average Cyclomatic complexity of a repository                                                                                                             |
|                | Number of style errors per LoC                                                                                                                             |
|                | Number of security errors of low severity per SLoC                                                                                                         |
|                | Number of security errors of medium severity per SLoC                                                                                                      |
|                | Number of security errors of high severity per SLoC                                                                                                        |
| Maintainability| Average maintainability index of a repository                                                                                                               |
|                | Number of closed issues and pull requests over last 2 years                                                                                                 |
|                | Number of closed issues and pull requests over last 1 year                                                                                                 |
|                | Number of closed issues and pull requests over last 6 months                                                                                               |
|                | Number of closed issues and pull requests over last 1 month                                                                                                |
|                | Number of commits per day                                                                                                                                  |
| Popularity     | Number of stargazers per day                                                                                                                              |
|                | Number of forks per day                                                                                                                                   |

Table 1: Measures used in GitRank

Cyclomatic complexity. Cyclomatic complexity of a source code is a well-known metric of the structural code complexity [21]. We use Python APIs of 1izard project [31] to obtain function-level cyclomatic complexity and then average it over a repository to obtain repository-level cyclomatic complexity2.

Style errors. We consider code formatting as a code quality metric and use cpplint project [15] to obtain the number of style errors (of the highest severity level)3. We divide the total number of errors

---

1 Available at https://github.com/nirhasabnis/gitrank
2 We end up using 1izard API directly as CoCom backend of Graal component in GrimoireLab [3] offered commit-specific code complexity instead of function-level or repository-level complexity.
3 Graal’s CoQua backend does not support obtaining this information for C/C++.
by the lines of source code (SLoC) [23] to obtain the error density. Static analysis tools such as cpplint are known to suffer from false positives [26], and although, GitRank does not account for them right now, we believe that they could be accounted by adjusting the weight assigned to style errors in the overall quality score.

**Security errors.** We also consider security errors to determine code quality. Security issues, however, could be of varying severity levels. In our framework, we consider three severity levels: low, medium, and high. We used FlawFinder tool [7] to obtain the security errors in C/C++ programs\(^4\) and divide the total number of errors reported for every level by the lines of source code (SLoC) to obtain the level-specific error density.

**Maintainability index.** Maintainability index is a well-known metric that incorporates a number of traditional source code metrics into a single number that indicates relative ease of maintaining source code [25]. We use the modified formula of MI [32] to obtain MI for individual modules from a repository\(^5\). MI for a repository is then obtained by averaging over MI for individual modules.

**Closed issues.** We use a combination of GitHub’s REST APIs and GrimoireLab’s Perceval [8] to obtain the number of closed issues over a period of time (last 2 years, last 1 year, last 6 months, and last 1 month, with increasing importance in that order) from the date of evaluation to determine maintenance activity of a repository. This is because we want to value current maintenance activity more.

**Stars, watches, and forks.** Existing approaches consider GitHub stars, subscriptions, and forks as popularity metrics [4]. We follow the same terminology. We use a combination of GitHub’s REST APIs [14] and GrimoireLab’s Perceval to obtain this information.

**Phase 2: Obtaining quality, popularity, and maintainability score of repositories.** In the second phase, we compare metrics across repositories to obtain overall score for every repository. Before we calculate scores for all the repositories, we first normalize values of all the measures to the range of 0% (lowest) to 100% (highest). This is performed by obtaining the lowest and the highest value of every measure and computing the position of a value within the range of the lowest and the highest value.

4 We used a standalone tool to obtain this information as we found that Graal’s CoVuln backend did not support our specific case.

5 We extend lizard project with a plugin to calculate Halstead volume in MI.

Normalized values of the measures are then used to compute the quality score and the popularity score as a weighted average, with equal weights for all the measures. For the maintainability score, we use different weights for the measures based on their significance\(^6\). We then compute overall score of a repository as a mean of all three scores. All of these scores are also normalized to the range of 0% (lowest) to 100% (highest).

3 EVALUATION

GitRank accepts a list of GitHub repositories (their URLs) as input and generates their ranked list as output (in CSV and HTML format). In our preliminary evaluation of GitRank, we ranked randomly-selected 500 GitHub repositories that use C++ as primary language. We dropped 12 repositories out of 500 as they had some peculiarities (such as unusual extensions of C++ files) because of which the tools used by GitRank could not obtain their metrics.

**Performance.** To support a large number of repositories, we have implemented phase 1 of GitRank to be highly-parallel (e.g., multiple machines and/or CPU cores). In our experiment, we divided the list of 500 repositories into 5 separate lists of 100 repositories each and processed them separately on 5 different machines. Phase 1 of GitRank took 1 hr and 25 KB of memory, while phase 2, implemented in a serial manner, took 0.1 sec and 1.5 KB of memory.

**Output.** Figure 1 shows the HTML-formatted ranked list of repositories in our experiment. The report also contains next level of details such as quality, maintainability, and popularity scores. Further levels of details (containing values of individual metrics) can be obtained with additional verbosity levels.

4 CONCLUSION

In this report, we described our efforts to build GitRank, a framework to rank open-source repositories using known source code metrics. So far, GitRank supports repositories using C/C++ as their primary languages and uses a combination of GrimoireLab toolkit and publicly-available tools for the implementation. As per our evaluation, GitRank is highly-parallel and performs reasonably well on commodity CPUs. We, however, did not explore applications of GitRank in this hackathon and consider this as a separate project that we plan to explore in the future.

6 We, however, envision that the weights could be adjusted by the GitRank user.
REFERENCES

[1] Rajiv D. Banker, Gordon B. Davis, and Sandra A. Slaughter. Software development practices, software complexity, and software maintenance performance. In The study. Manage. Sci., apr 1998.

[2] Christian Bird, Nachipanjan Nagappan, Brenden Murphy, Harald Gall, and Premkumar Devanbu. Don’t Touch My Code! Examining the Effects of Ownership on Software Quality. In Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, ESEC/FSE '11, 2011.

[3] Bitterga Inc. GitmoreLab: Free, Libre, Open Source Tools for Software Development Analytics. Available at https://chaos.github.io/gitmorelab/.

[4] Hudson Borges and Mario Tulio Valente. What’s in a GitHub Star? Understanding Repository Starring Practices in a Social Coding Platform. CoRR, abs/1811.07643, 2018.

[5] Kyriakos Chatzidimitriou, Michail Papamichail, Themistoklis Diamantopoulos, Michael Tsaparas, and Andreas Syrmosidis. rpm-Miner: An Infrastructure for Measuring the Quality of the rpm Registry. In 2018 IEEE/ACM 13th International Conference on Mining Software Repositories (MSR), 2018.

[6] Mark Chen, Jery Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Castro, Ener Ladd, Gabriel Lueu, Marko Kaprelian, Hann Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrow, Heidy Khllaaf, Gurish Saxtrey, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Parlov-Arelitha Power, Lukasz Kaiser, Mohammad R. Bavarian, Clement Winter, Philipe Tillet, Felipe Petroski Schu, Dave Cummings, Matthias Plappert, Fotos Chantris, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Pamo, Nikolos Teaz, Jie Tang, Igor Babushkin, Suchit Balaji, Shantamu Jain, William Saunders, Christopher Heese, Andrew N. Cape, Jun Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alex Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodio, Sam McCanludh, Ilya Sutskever, and Wojciech Zaremba. Evaluating Large Language Models Trained on Code. 2021.

[7] David Wheeler. FlawFinder. Available at https://dwheeler.com/flawfinder.

[8] Santiago Duenas, Valerio Cosentino, Gregorio Robles, and Jesus M Gonzalez-Barahona. A Novel Approach Leveraging Compilers. In Proceedings of the 4th International Conference on Software Engineering: Companion Proceedings, pages 1–4. ACM, 2018.

[9] Santiago Duenas, Valerio Cosentino, Jesus M. Gonzalez-Barahona, Alvarez del Castillo San Felix, Daniel Izquierdo-Cortazar, Luis Caniz-Diaz, and Alberto Perez Garcia-Plaza. GrimoireLab: A toolset for software development analytics. 7(e601).

[10] Claireanne H. Halstead. Elements of Software Science (Operating and Programming Systems Series). Elsevier Science Inc., USA, 1977.

[11] Niranjana Hasabnis and Justin Gottschlich. Controlflow: A Self-Supervised Iodysyn- cratic Pattern Detection System for Software Control Structures. In Proceedings of the 5th ACM SIGPLAN International Symposium on Machine Programming, MAPS 2021, 2021.

[12] Niranjana Hasabnis and R. Sekar. Extracting Instruction Semantics via Symbolic Ex- ecution of Code Generators. In Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE, 2016.

[13] Niranjana Hasabnis and R. Sekar. Lifting Assembly to Intermediate Representation: A Novel Approach Leveraging Compilers. In Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS, 2016.

[14] GitHub Inc. GitHub REST API. Available at https://docs.github.com/en/rest.

[15] Google Inc. cplint - static code checker for C++. Available at https://github.com/cplint/cplint.

[16] ISO. ISO/IEC 29000:5000. Available at https://www.iso.org/standard/35683.html.

[17] NiranjanHasabnisandRSekar. ExtractingInstructionSemanticsviaSymbolicEx- ecution of Code Generators. In Proceedings of the 24th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, ESEC/FSE ’11, 2011.

[18] T.J. McCabe. A Complexity Measure. IEEE Transactions on Software Engineering, SE-2(4):308–320, 1976.

[19] Jeremy Ludvig and Devin Cline. CBR Insight: Measure and Visualize Source Code Quality. In 2019 IEEE/ACM International Conference on Computer Software and Application (COMPSAC). IEEE, 2019.

[20] Jeremy Ludvig, Steven Xu, and Frederick Webber. Compiling Static Software Metrics for Reliability and Maintainability from GitHub Repositories. In Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering, ASE ’18, pages 566–578, 2018.

[21] T.J. McCabe. A Complexity Measure. IEEE Transactions on Software Engineering, SE-2(4):308–320, 1976.

[22] Nuthan Munialah, Steven Kroh, Craig Cahery, and Meiyanpan Nagappan. Curating GitHub for Engineered Software Projects. Empirical Software Engineering, 2017.

[23] Vu Nguyen, Sophia Driep-Dubin, Thomas Tan, and Barry Boehm. A SLOC Counting Standard. In COCOMO II Forum, 2007.

[24] NPM, Inc. NPM. Available at https://www.npmjs.com/.

[25] P. Oman and J. Hagemeister. Metrics for Assessing a Software System’s Main- tainability. In Proceedings Conference on Software Maintenance 1992, 1992.

[26] Sebastiano Panichella, Venera Arnaoudova, Massimiliano Di Penta, and Giuliano Antoniol. Would static analysis tools help developers with code reviews? In 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), pages 161–170, 2015.

[27] Baptiste Roziere, Marie-Anne Lachaux, lowik Chassault, and Guillaume Lam- ple. Unsupervised Translation of Programming Languages. Advances in Neural Information Processing Systems, 2020.

[28] Ioannis Samoladas, Georgios Gousios, Diomidis Spinellis, and Ioannis Stamelos. The SQO-OSS Quality Model: Measurement Based Open Source Software Eval- uation. In Barbara Russo, Ernesto Damiani, Scott Hassam, Bjorn Lundell, and Giancarloucci, editors, Open Source Development, Communities and Quality, Boston, MA, 2008. Springer US.

[29] Diomidis Spinellis, Georgios Gousios, Vasilios Karakoidas, Panagiotis Louridas, Paul Adams, Ioannis Samoladas, and Ioannis Stamelos. Evaluating the Quality of Open Source Software. Electr. Notes Theor. Comput. Sci., 03 2009.

[30] Ioannis Stamelos, Leheris Angela, Apostolos Oikonomou, and Georgios N. Bleris. Code Quality Analysis in Open Source Software Development. Information Systems Journal, 2002.

[31] Terry Yin and other contributors. A Simple Code Complexity Analyser. Available at https://github.com/terryin/lizard.

[32] Kurt D. Welker, Paul W. Oman, and Gerald G. Atkinson. Development and Application of an Automated Source Code Maintainability Index. Journal of Software Maintenance, may 1997.

[33] Michihiro Yasunaga and Percy Liang. Graph-based, Self-supervised Program Repair from Diagnostic Feedback. In International Conference on Machine Learning (ICML), 2020.

[34] Fangke Ye, Shengtian Zhou, Anand Venkat, Ryan Marucos, Nesime Tatbul, Jesmin Jahan Tithi, Niranjana Hasabnis, Paul Petersen, Timothy Mattson, Tim Krasa, Pradeep Dubey, Vivek Saraz, and Justin Gottschlich. MISEM: An End-to- End Neural Code Similarity System. arXiv preprint arXiv:2006.05285, 2020.