Exploring Software Library Metrics with Repository Badges

by

Monica Bui
University of Alberta
bui1@ualberta.ca

Contents

List of Figures

1 Introduction

Libraries, Frameworks, and Application Programming Interfaces (APIs) provide developers a way to reuse existing functionalities built by someone else without having to re-implement already built features. Given a large collection of libraries out there, it is often difficult and not clear how to select the best one to use for your own project.

Developers may resort to first doing a general search of their desired library features with search results indicating various resources such as a Q&A website like StackOverflow [1] or a website that hosts open source libraries such as Github [2].

Previous research has examined different, inner aspects of libraries that may have mentioned in the above online resources [3, 4, 5, 6, 7]. These aspects or defined

2 Related Work

2.1 Library Selection Methods

2.2 Metrics

2.2.1 Quality Assurance

2.2.2 Community Support

2.2.3 Repository General Information

2.3 Summary

3 Background

3.1 Badges

3.2 Online Badge Services

4 Badge Implementation

4.1 Overarching Structure of Scripts

4.2 Security

4.2.1 Definition

The security badge is inspired by Mora's et. al [5] security metric implementation. However, there are limitations related to classifying some security vulnerabilities due to inaccurate issue descriptions. These inaccuracies would suggest that there was a security problem but in reality was another issue altogether. To avoid this conflict brought by issue descriptions, we propose to use another existing tool called SpotBugs [8]. From the SpotBugs [8] website description itself, the program *uses static analysis to look for bugs in Java code*. In conjunction with the FindSecBugs [9] plugin to provide a larger data set of security bug patterns to look for, both these tools will allow for greater accuracy of targeting security bugs.

The security badge represents the number of security bugs reported by SpotBugs [8] with the FindSecBugs [9] plugin. We filter for only security

bug patterns and configure SpotBugs to only classify bugs on the highest confidence setting with maximum effort toggled on to increase precision. To the user, this badge helps to detect for any known security vulnerabilities before using the library.

4.2.2 Implementation

Figure 1: Example of Security Badge

First, we have a text file that holds open source Java library links hosted on GitHub [2]. A shell script clones and compiles them respectively under Gradle [10] or Maven [11]. After compilation, a script runs SpotBugs and FindSecBugs per library under our defined configured settings and stores the respective result into the database. The client can hit the security script endpoint to retrieve the saved results which then can be placed into Shields.io [12] to output the security badge with an example shown in figure 1.

4.3 Last Discussed on Stack Overflow

4.3.1 Definition

We take Mora's et. al [5] Last Discussed on Stack Overflow metric and transform it into a badge. Based on the metric, the badge represents the latest date of a question posted for a specific library on the Stack Overflow website [1]. To the user, this badge will display if there is any recent activity using the library and the community involvement behind it.

4.3.2 Implementation

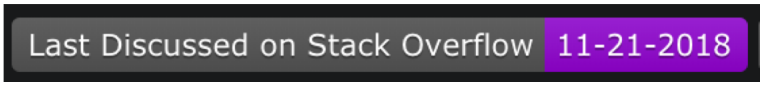

Figure 2: Example of Last Discussed on Stack Overflow Badge

Borrowing the implementation technique from Mora et. al [5], we use the StackExchange API [13] to search up the library's name under tag search and extract the most popular tag. With the tag, we make a GET request to the API to search for the most recent question containing the tag and extract the date. An example of the badge is shown in figure 2.

4.4 Issue Response Time

4.4.1 Definition

Using Mora's et. al [5] Issue Response Time definition, this badge illustrates the average amount of days needed to get the first reply to an issue. The badge is calculated via the response time formula described in [5]. It also indicates if there are any changes to the average over time as shown by the rightmost symbol in 3. This badge will allow users to check if there is active maintainence of the library and if there is community support through the issue discussions.

4.4.2 Implementation

Figure 3: Example of Issue Response Time Badge

We use the algorithm outlined by Mora et. al [5] to calculate the issue response time average which is the difference of all the dates between issue creation date and first comment date over the total number of accounted issues. The script itself is tweaked to use GitHub's [2] GraphQL API instead of REST to decrease the runtime of iterating through all the issues and respective comments. The earlier API type allows us to get both the issue creation date and first comment date in one request call versus the latter API with two calls. We also discard deleted accounts and do not take them into account in our average. An example of the badge is shown in figure 3.

4.5 Contributor Pull Request Merge Rate

4.5.1 Definition

Kononenko et. al [14] highlights that, *PRs submitted by the owners are approved more quicklier compared to those made by external devs.* This can hinder open source development for developers who wish to partake in the project but are not core maintainers or closely associated with the project. In addition, there is the lack of assessment defined badges [3] related to pull requests (PRs) as of December 2018. The existing PR badges on Shields.io [12] display readily available PR information that can be easily found on a software repository. Taking both resources [12, 14] into account, this has inspired the development of the Outside Contributor Pull Request Merge Rate assessment badge.

The badge represents the percentage of PRs that have been merged into the repository authored by outside contributors. We define outside contributors using the survey design outlined by Trockman et. al [3] if they have less than 10% of commits made to a repository. This badge helps users to select libraries that are friendly and open to open source contributions, and easily evaluate the merge rate not accessible beforehand. With a higher merge rate, it will provide opportunities for developers to extend features of a library with higher confidence that their PR will be integrated into the repository.

4.5.2 Implementation

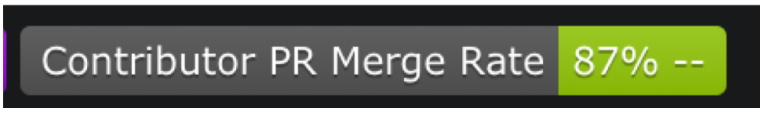

Figure 4: Example of Outside Contributor PR Merge Rate Badge

We have a seperate script endpoint that uses the GitHub [2] API to get all the users that have made commits to the library. We classify users as outside contributors or not using the technique described by Trockman et. al [3] and store them as a JSON object in the database. Completing this task will allow us to use another endpoint to calculate the PR metric. With the GitHub [2] API, we calculate the average as follows: total number of merged PRs divided by total number of all PRs where PRs of all statuses

are associated with outside contributors. An example response is shown in figure 4.

4.6 Release Frequency

4.6.1 Definition

As defined by Mora et. al [5], release frequency is the average number of days between releases in repositories. This badge similar to security, pull request, and issue response metrics has another visual indicator beside the metric value itself that displays whether or not the average changed over time. To the user, the badge suggests if there are fresh releases of software so old code is not being integrated with your project that can be more vulnerable to bugs and security problems.

4.6.2 Implementation

Figure 5: Example of Release Frequency Badge

We use the GitHub [2] API to get all the releases of the library and follow the algorithm steps outlined by Mora et. al [5] to calculate the release frequency metric. The average is the difference of all the dates between the releases over the total number of releases. Although not explicitly stated by Mora, we only account for repositories that have two or more releases to compute the metric. The final result would look similar to the example pictured in figure 5.

5 Evaluation

5.1 Open Source Evaluation

5.2 Future Work Evaluation

6 Threats to Validity and Limitations

6.1 Validity

6.1.1 Classification of Outside Contributors

6.1.2 Security Bug Classification

6.2 Limitations

6.2.1 Request Timeouts

All metrics are susupetible to this due to GitHit image restrictions 3 seconds or less.

6.2.2 Local Scripts

7 Conclusion and Future Work

References

[1] Stack-Overflow, "Stack overflow," http://www.stackoverflow.com.

[2] GitHub, "Github," "http://github.com/".

[3] A. Trockman, S. Zhou, C. Kästner, and B. Vasilescu, "Adding sparkle to social coding: An empirical study of repository badges in the npm ecosystem," in *International Conference on Software Engineering*, ser. ICSE. ACM, 2018, pp. 511–522.

[4] A. C. Hora and M. T. Valente, "Apiwave: Keeping track of api popularity and migration." in *ICSME*. IEEE Computer Society, 2015, pp. 321–323.

[5] F. L. de la Mora and S. Nadi, "Which library should I use?: a metric-based comparison of software libraries," in *ICSE (NIER)*. ACM, 2018, pp. 37–40.

[6] G. Uddin and F. Khomh, "Opiner: An opinion search and summarization engine for apis," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Oct 2017, pp. 978–983.

[7] C. Chen, S. Gao, and Z. Xing, "Mining analogical libraries in q amp;a discussions – incorporating relational and categorical knowledge into word embedding," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, March 2016, pp. 338–348.

[8] SpotBugs, "Spotbugs," "https://spotbugs.github.io/".

[9] FindSecBugs, "Findsecbugs," "https://find-sec-bugs.github.io/".

[10] Gradle, "Gradle," "https://gradle.org/".

[11] Maven, "Maven," "https://maven.apache.org/".

[12] Shields.io, "Shields.io," "https://shields.io/#/".

[13] StackExchangeAPI, "Stackexchange api," "https://api.stackexchange.com/".

[14] O. Kononenko, T. Rose, O. Baysal, M. Godfrey, D. Theisen, and B. de Water, "Studying pull request merges: A case study of shopify's active merchant," in *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, ser. ICSE-SEIP '18. New York, NY, USA: ACM, 2018, pp. 124–133. [Online]. Available: http://doi.acm.org/10.1145/3183519.3183542