Exploring Software Library Metrics with Repository Badges

by

Monica Bui
University of Alberta
bui1@ualberta.ca

Sarah Nadi
University of Alberta
nadi@ualberta.ca

Scripts & Documentation:

https://github.com/ualberta-smr/bui-course-f18

Contents

1 Introduction **3**

2 Related Work **3**
- 2.1 Library Selection Tools . . . 3
- 2.2 Library Metrics . . . 4

3 Background **5**
- 3.1 Badges . . . 5
- 3.2 Online Badge Services . . . 6

4 Badge Implementation **6**
- 4.1 Overarching Structure of Scripts . . . 6
- 4.2 Security . . . 7
  - 4.2.1 Definition . . . 7
  - 4.2.2 Implementation . . . 8
- 4.3 Last Discussed on Stack Overflow . . . 8
  - 4.3.1 Definition . . . 8
  - 4.3.2 Implementation . . . 8
- 4.4 Issue Response Time . . . 9
  - 4.4.1 Definition . . . 9
  - 4.4.2 Implementation . . . 9
- 4.5 Contributor Pull Request Merge Rate . . . 10
  - 4.5.1 Definition . . . 10
  - 4.5.2 Implementation . . . 10
- 4.6 Release Frequency . . . 11
  - 4.6.1 Definition . . . 11
  - 4.6.2 Implementation . . . 11

5 Evaluation **11**
- 5.1 Open Source Evaluation . . . 11
- 5.2 Future Work Evaluation . . . 12

6 Threats to Validity and Limitations **13**
- 6.1 Validity . . . 13
  - 6.1.1 Classification of Outside Contributors . . . 13
  - 6.1.2 Security Bug Classification . . . 13
- 6.2 Limitations . . . 14

6.2.1 Request Timeouts 14
6.2.2 Local Scripts 14
6.2.3 Badge Specific Limitations 15

7 Conclusion and Future Work **15**

List of Figures

1 Example of Repository Badges [1] 5
2 Flow diagram representing how we created our badges 7
3 Example of Security Badge 8
4 Example of Last Discussed on Stack Overflow Badge 8
5 Example of Issue Response Time Badge 9
6 Example of Outside Contributor PR Merge Rate Badge . . . 10
7 Example of Release Frequency Badge 11
8 Example of Stack Exchange Monthly Questions badge 12
9 Maintainer Example per posted Shields.io issue [2] 13
10 Contributor Example per posted Shields.io issue [2] 13

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 [3] or a website that hosts open source repositories such as Github [4]. On these websites, opinions from the community [5] may sway users to select one library versus another as other users highlight a library's strengths and weaknesses. Users can view badges belonging to a repository that display inner aspects of a library that are more accessible to view [6]. Visible cues or metrics that are transparent allows users to see community support, how "alive" a library is, and to help users assess a project's real value [7, 8]. Users may also dig up other library characteristics like functionality, usability, quality, compatability, security to judge libraries with[6, 7, 9]. These aspects often contribute to the popularity of a library [9, 10] that users find the most useful feature across differentdomains [11]. Users depend on these factors i.e metrics that the related works above mention to help select a library to use.

Our research proposes to build transparent repository badges so that users can easily identify inner metrics of a library that will assist developers in library selection. Based on Trockman's et al. [6] badge recommendations and metrics from Mora et al. [7], we look to create dynamic badges that take more work to create i.e assessment badges, where the metric itself is quantifiable, and implement badges that do not exist yet so we do not develop already created badges. These suggestions are taken into account to produce effective, reliable badges.

2 Related Work

2.1 Library Selection Tools

There are other related works that have focused on users decisions on libraries. Mora et al. [7, 11] created a library comparison website that features metric based comparison between libaries. While our work is based on metrics and the idea of comparing libraries on metrics, we use badges that are integrated into a repository so that users can view a libary's aspects in one place versus a seperate website to view all the extracted library data in. Chen et al. [12] worked on an analogical library recommender that would explicitly report other similar libraries in a different programming languages based on the input library. e.g input is JUnit testing for Java and output is Jest testing library for JavaScript. This tool would help users to help migrate their knowledge across different programming languages. This project gives an actual reccomendation for what library to use but our research allows users to make their own judgement for library selection using badges to simplify this process. Uddin et. al [5] uses sentiment analysis to examine developer opinions on various libraries and summarizes positive and negative opinions under various metrics. We both use metrics as a basis of our research and provide resources for developers to help select libraries with but Uddin focuses more on opinions themselves [3]. Trockman et. al [6] studies the use of repository badges in the NPM (node package modules) ecosystem, what the most popular badges are, and if badges intend to signal their underlying qualities well. Trockman's review does not directly look into how to help users select libraries like our work does but evaluates on different criteria that makes a badge reliable that influence developers decisions to use which library. Mileva et. al [9] examines various library usage trends within different library versions and researchs into different factors that affect a user's decision to use a specific version of a library. In contrast with library versions, we look into providing badges as resources to select between different libraries instead.

2.2 Library Metrics

We base the metrics used in our badges off of other related works and identify other library aspects that were considered in our research process. Hora et al. [10] analyzes API popularity and migration as useful aspects that developers care about with the former showing community support and the latter suggesting ease of use/usability. While a popularity badge already exists, migration is a challenging metric to quantify as a badge. Mora et al. [7, 11] as discussed in the previous subsection researches into many more metrics that help developers make their own judgements on libraries. We use some of their metrics that match our criteria e.g release frequency to build off of. Dabbish et al. [8] looks into transparent library features that developers can easily access and how it affects project workflow and library selection evalutation. Many metrics such as number of forks, and number of commits although easily accessible would not make a effective badge [6] as it simple to find this information. Kononenko et al. [13] examines successful pull requests (PRs) and criteria that determines what PRs get merged in or not. Many of the current PR badges in Shields.io [14] are simple lookup metrics and from the Kononenko's work, we want to create metrics from their useful research insights on successful PRs.

3 Background

3.1 Badges

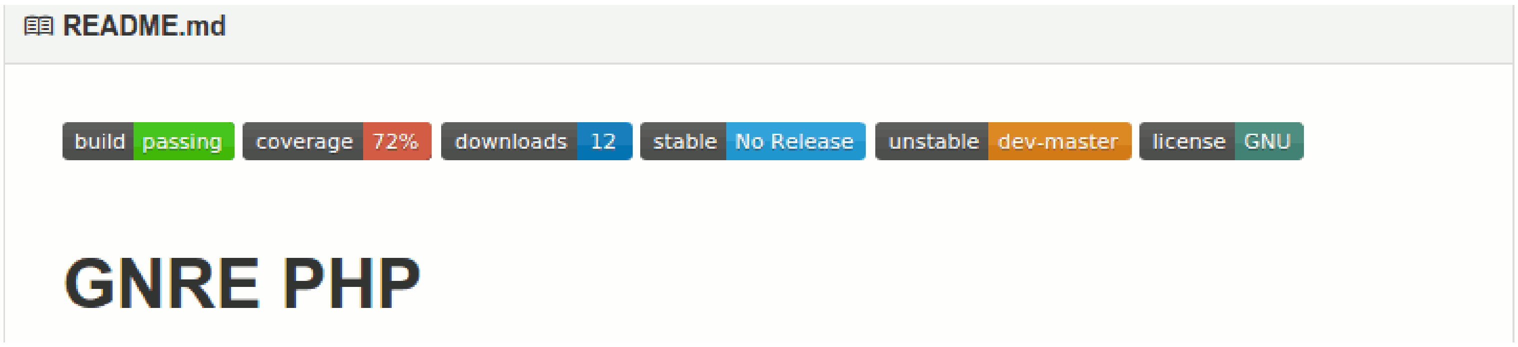

Figure 1: Example of Repository Badges [1]

Badges are small visual images that are usually displayed in a repository's readme file e.g figure 1. Badges display a quantitative metric that indicates a library's inner qualities or aspects that can be difficult to access at times. This assists developers with choosing a library as they can compare and contrast between libraries using badges. These metrics found in the badges can be dynamically changing over time e.g test code coverage percentage or stay static e.g license information.

3.2 Online Badge Services

There are online badge services that exist to help developers use badges for their repositories without having to create badges from scratch. To get a specific purpose badge, we would use web services e.g Snyk for security dependency vulnerabilities [15] and register our application with the service to get a full analysis of our project. Using the Snyk example, once registered we can add the security GitHub badge to our readme as suggested in markdown, [![KnownVulnerabilities](https://snyk.io/test/github/username/repo/badge.svg)](https://snyk.io/test/github/username/repo) [15]. We take our custom project service URL and input its data with the snyk badge svg image itself.

Other services like InchCI for documentation [16] or Circleci for build status [17] for example follow a similar method to create a badge. However to utilize more badges outside of individual services, Shields.io [14] is often used as it is the largest collection of online hosted repository badges that users can contribute new badges to and use already existing badges. Just like with individual services, in Shields.io [14] we would hit custom http endpoints e.g https://img.shields.io/stackexchange/:stackexchangesite/qm/:query.svg [14] by filling in the required query parameters to get a badge to insert into a readme file. Users can also use Shields.io [14] badge templates to customize the image of a badge and input their own information into the badge.

4 Badge Implementation

4.1 Overarching Structure of Scripts

Instead of using Shields.io [14] endpoints, we create our own custom scripts with endpoints so that we can prototype our badges locally in a more flexible development environment. If we were to integrate all the scripts into Shields.io [18] instead of developing them locally, we may hit more extreme cases of run time issues discussed in the limitations section. The diagram in figure 2 represents the flow of how we implement our badges.

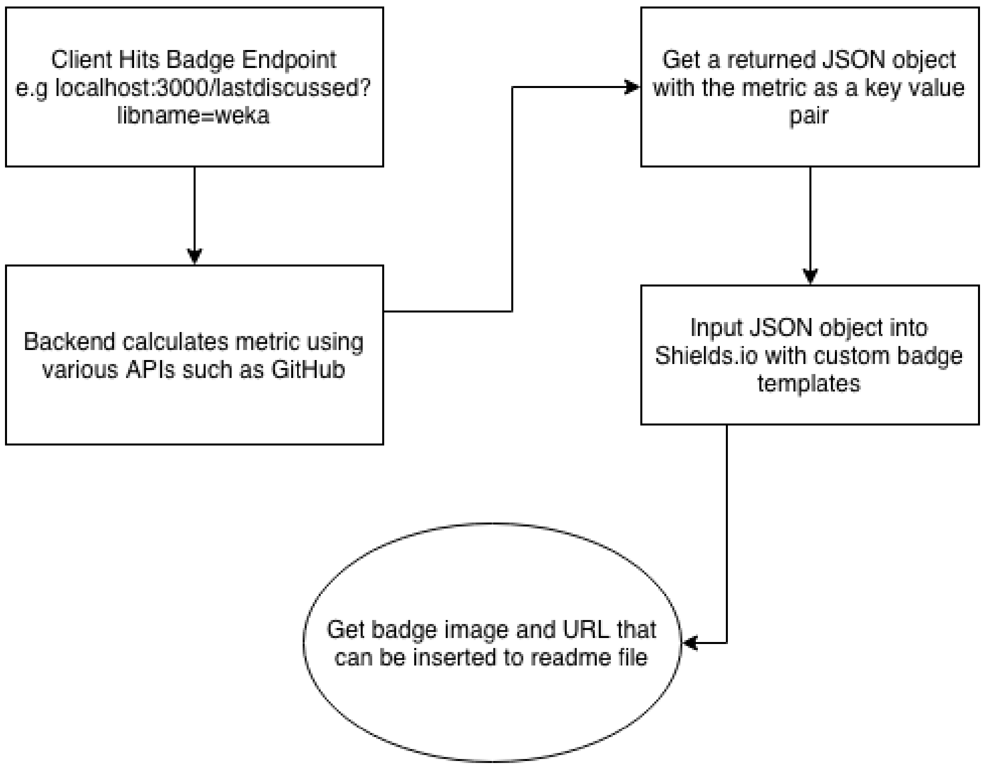
Client Hits Badge Endpoint e.g localhost:3000/lastdiscussed? libname=weka
Backend calculates metric using various APIs such as GitHub
Get a returned JSON object with the metric as a key value pair
Input JSON object into Shields.io with custom badge templates
Get badge image and URL that can be inserted to readme file

Figure 2: Flow diagram representing how we created our badges

4.2 Security

4.2.1 Definition

The security badge is inspired by Mora's et al. [7] 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 [19]. From the SpotBugs [19] website description itself, the program *uses static analysis to look for bugs in Java code.* In conjunction with the FindSecBugs [20] 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 [19] with the FindSecBugs [20] 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

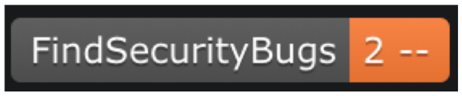

Figure 3: Example of Security Badge

First, we have a text file that holds open source Java library links hosted on GitHub [4]. A shell script clones and compiles them respectively under Gradle [21] or Maven [22]. 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 [14] to output the security badge with an example shown in figure 3.

4.3 Last Discussed on Stack Overflow

4.3.1 Definition

We take Mora's et al. [7] 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 [3]. 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

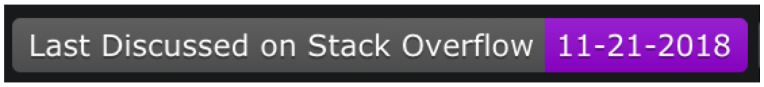

Figure 4: Example of Last Discussed on Stack Overflow Badge

Borrowing the implementation technique from Mora et al. [7], we use the StackExchange API [23] 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 4.

4.4 Issue Response Time

4.4.1 Definition

Using Mora's et al. [7] 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 [7]. It also indicates if there are any changes to the average over time as shown by the rightmost symbol in 5. 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

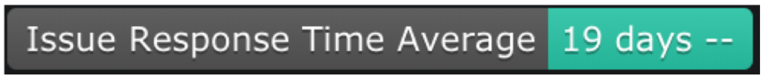

Figure 5: Example of Issue Response Time Badge

We use the algorithm outlined by Mora et al. [7] 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 [4] 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 5.

4.5 Contributor Pull Request Merge Rate

4.5.1 Definition

Kononenko et al. [13] 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 [6] related to pull requests (PRs) as of December 2018. The existing PR badges on Shields.io [14] display readily available PR information that can be easily found on a software repository. Taking both resources [14, 13] 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. [6] 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

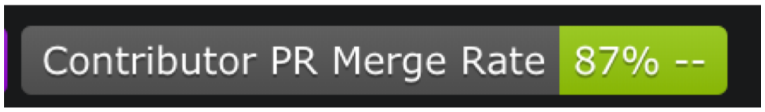

Figure 6: Example of Outside Contributor PR Merge Rate Badge

We have a seperate script endpoint that uses the GitHub [4] 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. [6] 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 [4] 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 6.

4.6 Release Frequency

4.6.1 Definition

As defined by Mora et al. [7], 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

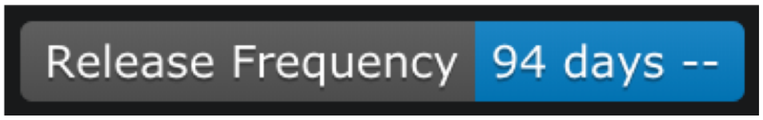

Figure 7: Example of Release Frequency Badge

We use the GitHub [4] API to get all the releases of the library and follow the algorithm steps outlined by Mora et al. [7] 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 7.

5 Evaluation

5.1 Open Source Evaluation

Due to the short timeline of the project, there is no formal evalutation of the work itself. However, we can qualitatively assess our work by contributing our badges to Shields.io [14] and evaluate maintainers thoughts regarding our badges. On the Shields.io Github repository [18], we post an issue [2] proposing the addition of the Last Discussed on Stack Overflow badge. We choose this badge becasuse it has the least amount of data to process and provides us a good indication if our badge can be successfully integrated into Shields.io [14] and used by the general public. A core maintainer responds back highlighting that there are not too many Stack Exchange badges hosted online and our proposal would be useful per design changes to be Number of Questions in Past Month on Stack Exchange instead. This change would assist in eliminating bias in the metric and skewed data if there is a recent question posted but for a rarely discussed library confusing a developer using the badge.

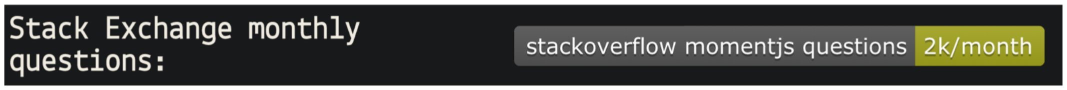
Stack Exchange monthly questions:
stackoverflow momentjs questions
2k/month

Figure 8: Example of Stack Exchange Monthly Questions badge

In order to make a PR for our proposed badge, we make two other pull requests to respectively create unit tests for the StackExchange service class and migrate the existing legacy code to Shields.io's newest API for the service class. This would change the existing StackExchange total questions and StackExchange user reputation badges. Once completed, we make a PR for the Number of Questions in Past Month on Stack Exchange badge using the newest Shields.io API. This is calculated by using the StackExchange API and filtering for the number of questions between the first and last day of the previous month from today. Having this metric also allows us to view questions from other StackExchange sub websites besides Stack Overflow. We were able to successfully get this PR merged and code deployed to the Shields.io [14] production website to be used by the public as displayed in figure 8. Although we cannot generalize that this can be said true for the rest of our created badges, this is a good starting evaluation for our badges that they are indeed useful.

5.2 Future Work Evaluation

If given a longer timeline for this project, posting more GitHub issues for the rest of our created badges would be a good start. This would allow us to evalutate for more badge design changes and if they can be successfully pushed to production. Maintainers can comment on our work and provide an outside opinion on our badges to analyze if our work is suitable. In addition, we can determine the amount of users using our StackExchange and other badges as a metric of popularity to see if users are using our creations.

6 Threats to Validity and Limitations

6.1 Validity

6.1.1 Classification of Outside Contributors

We use Trockman's et al. [6] technique to help classify outside contributors for our PR badge. However, doing some manual testing with our badge with various repositories such as the Shields.io [18] one, it would misclassify some maintainers as outside contributors. This would happen if a maintainer has less than 10% of commits made to a repository which we use as a guideline for contributors. To counter this issue in the future, we could iterate through all the commits of a library and define a maintainer as a user who is an explicit member of the organization that hosts the library e.g figure 9 otherwise, the user would be an outside contributor e.g figure 10.

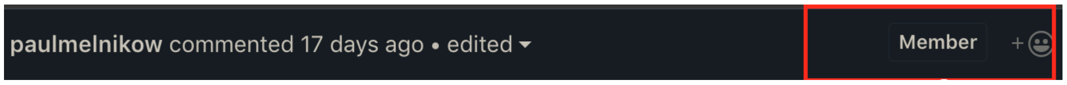
paulmelnikow commented 17 days ago • edited

Figure 9: Maintainer Example per posted Shields.io issue [2]

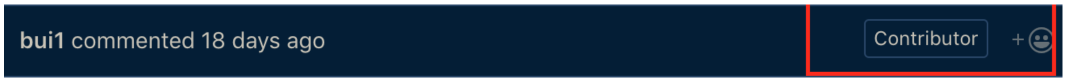
bui1 commented 18 days ago

Figure 10: Contributor Example per posted Shields.io issue [2]

6.1.2 Security Bug Classification

SpotBugs [19] could end up misclassifying and/or missing security bugs for absent bug patterns. To make sure our security badge has the most precise value possible, we use SpotBugs [19] and FindSecBugs [20] on the highest effort and confidence setting to ensure that our range of classifcation is wide enough to capture the most number of security vulnerabilities.

6.2 Limitations

6.2.1 Request Timeouts

Most of the metrics are susceptible to a GitHub [4] image restriction of 3 seconds or less where the badge has to be displayed before it hits the request timeout limit and not display itself. To counteract this issue, we made several design decisions per badge to make sure each metric's run time will be efficient enough to stay within the specified limit.

For the security badge because it takes some time to get the number of bugs from SpotBugs [19], we run the shell scripts that runs SpotBugs seperately from the client server interaction to get the metric. This way when the user hits the security badge endpoint, we only need to return the data previously saved in the database from the shell script execution.

For the issue response badge, we return right away to the user the last saved metric saved from the database. Asynchronously in the background, we calculate the new metric and replace it in the database. This helps to not block Github from displaying the badge right away and the next time the user hits this endpoint, we will see the new badge's value.

For the release frequency badge, we determine if the number of releases changed from the last time a user has hit this endpoint. If there is no change, we just return the previously saved value which saves a lot of time from re-calculating the metric from scratch. If the number of releases changes, we calculate the metric using normal means.

For the PR badge, we calculate the metric starting from the last date a user has hit this specific endpoint. Caching this date allows us to only need to account for new info starting from the previous script execution instead of figuring out the metric from scratch for every run.

6.2.2 Local Scripts

Currently the scripts to calculate the metrics required for the badges only run locally on one's own machine. We utilize ngrok [24] a service that *exposes local servers behind NATS and firewalls to the public internet over secure tunnels* to help us test our badges online without dealing with the overhead of maintaining servers and makes it simpler to prototype the badge's functionalities. However, this would require restarting ngrok every time after your terminal or machine shuts down and makes it difficult to keep the ngrok connection alive for others to use because of this limitation. In the future, deploying our scripts online using services such as Heroku [25] or Now [26] is one possibility. Another solution is to contribute the rest of our badges to Shields.io [18] and have the badges hosted there.

6.2.3 Badge Specific Limitations

Most limitations of each badge is outlined in the Request Timeouts section. Specifically for the security badge, there are programming language boundaries. SpotBugs [19] itself can only be used for Java libraries and we are unable to evaluate the tool's functionalities with other open source libraries written in different programming languages.

All our badges also depend on the library being open source so that it can openly access required information without deadling with authentication issues.

7 Conclusion and Future Work

References

[1] Marabesi. (2015) Github badges!(php repository). [Online; accessed Sept 19, 2018]. [Online]. Available: https://marabesi.com/wp-content/uploads/2015/06/badges2.png

[2] Shields.io. Shields.io stackoverflow issue. [Online]. Available: https://github.com/badges/shields/issues/2378#issuecomment-440735647

[3] Stack-Overflow. Stack overflow. [Online]. Available: http://www.stackoverflow.com

[4] GitHub. Github. [Online]. Available: http://github.com/

[5] 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.

[6] 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.

[7] 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.

[8] "Leveraging transparency." *IEEE Software, Software, IEEE, IEEE Softw*, no. 1, p. 37, 2013. [Online]. Available: http://login.ezproxy.library.ualberta.ca/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.6357175&site=eds-live&scope=site

[9] Y. M. Mileva, V. Dallmeier, M. Burger, and A. Zeller, "Mining trends of library usage," in *Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution (IWPSE) and Software Evolution (Evol) Workshops*, ser. IWPSE-Evol '09. New York, NY, USA: ACM, 2009, pp. 57–62. [Online]. Available: http://doi.acm.org/10.1145/1595808.1595821

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

[11] F. L. de la Mora and S. Nadi, "An empirical study of metric-based comparisons of software libraries," in *Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering*, ser. PROMISE'18. New York, NY, USA: ACM, 2018, pp. 22–31. [Online]. Available: http://doi.acm.org/10.1145/3273934.3273937

[12] 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.

[13] 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

[14] Shields.io. Shields.io. [Online]. Available: https://shields.io/#/

[15] Snyk. Snyk. [Online]. Available: https://snyk.io/

[16] InchCI. Inchci. [Online]. Available: https://inch-ci.org/

[17] Circleci. Circleci. [Online]. Available: https://circleci.com/

[18] Shields.io. Shields.io github repository. [Online]. Available: https://github.com/badges/shields

[19] SpotBugs. Spotbugs. [Online]. Available: https://spotbugs.github.io/

[20] FindSecBugs. Findsecbugs. [Online]. Available: https://find-sec-bugs.github.io/

[21] Gradle. Gradle. [Online]. Available: https://gradle.org/

[22] Maven. Maven. [Online]. Available: https://maven.apache.org/

[23] StackExchangeAPI. Stackexchange api. [Online]. Available: https://api.stackexchange.com/

[24] Ngrok. Ngrok. [Online]. Available: https://ngrok.com/

[25] Heroku. Heroku. [Online]. Available: https://www.heroku.com/

[26] Now. Now. [Online]. Available: https://zeit.co/now