CMPUT 499: Mining Sofware Repositories

Literature Review

Monica Bui

1 Introduction

Third-party software libraries and Application Programming Interfaces (APIs) offer a way for developers to use existing features and integrate them within their projects without having to re-invent the wheel. There are many such libraries to use for different programming languages and for different purposes e.g. encryption, text communications etc. Although there is a lot of flexibility when it comes to choosing the appropriate library, it is also conflicting to figure out which one is best for your needs. There are a variety of factors involved such as security and developer support that will affect your project's overall functionality. In this review, we look at several diferent articles to help analyze various properties of libraries and to determine a reliable method of comparison between them.

2 Article Review

2.1 Library Recommenders

One problem is researching for new analogical libraries to use that has similar functionalities to the ones you already know of. Chen's et. al [1] paper discusses their library recommender that consumes as input a list of libraries from community resources such as blogs and Q&A sites like StackOverflow [2] and outputs a list of recommended libraries in a different language of choice that offers similar features to the input language argument.

This is implemented by first mining tags from questions posted online on StackOverflow [2]. These tags are split into two knowledge bases: relational and categorical. Respectively, relational knowledge is how pairs of tags are correlated to each other e.g. Java and JUnit, while categorical knowledge consists of how tags are grouped into categories such as language, operating system, concept, or library. Both these bases are analyzed by NLP to ensure efficient extraction of data and are sub categorized correctly. Having these seperated tag categories and relatonships between tags often mentioned together provides a simpler way to recommend a new library. With the database in place, users can then search for recomendations through the built web application called *SimilarTech* [3].

While trying out the application myself, I found that search results would yield for libraries only mentioned in specific contexts. Axios [4] a popular

Javascript HTTP API should output expected recomendations like the Requests [5] module for Python but the actual output printed was empty. Axios [4] itself is mentioned around many situations like REST, APIs, and HTTP requests making it difficult to target in what circumstance it should be recommended in.

While the number of languages it can suggest libraries for is limited to 5, the precision metric is impressive with 1 language at 81% and with 5 being at 67% demonstrating its potential to grow in the future.

Going back to the key problem of deciding on the right library to use, Uddin's et. al [6] article highlights their approach to this by looking at personal developer opinions on different resources and how it affects the reader's decision. The sentiment behind this can be used to indicate if its a positive, questionable, or a negative API to use. The tool created, Opiner [7] is a summarization engine that examines these opinions and evaluates its sentiment to see if the API should be recommended through displaying ratings on API traits in addition to organizing top opinions for both positive and negative sides.

The backend of the application web crawls through a Q&A site, StackOverflow [2] to extract answer information surrounding an API topic. This data is used to help discover when an API in referenced and what kinds of opinions are associated with an API by "finding its nearest API mention" [6] through a API name, link, or code said in the same sentence. This combination of data is then used to summarize common aspects developers care about such as usability and performance by finding their interests through a survey.

Using Opiner [7] through a small research study evaluated on 2 seperate groups of particpant choices to see if just StackOverflow [2] alone would work for selection of an API or using both StackOverflow and Opiner would be better. We view that developers are more confident in their selection with both tools versus just StackOverflow alone.

For correctness, Opiner gives an overall 90% precision metric on library recommendations that demonstrates that this data is indeed reliable. A weakness is that although the evaluation of the study proves useful, having 100% as a research study result for using both StackOverflow and Opiner is difficult to generalize for a larger population.

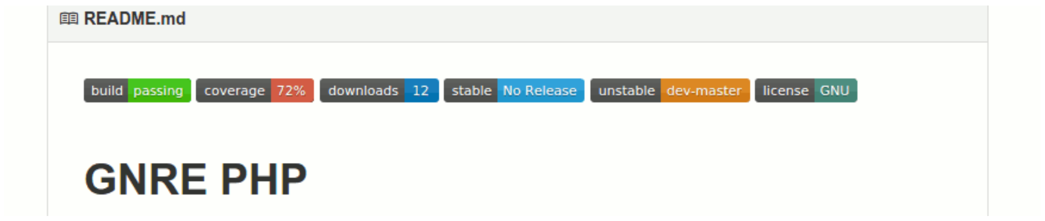

Figure 1: Example badges in a open source PHP repository [10]

2.2 Github Badges

With many open source software to use, it's difficult to pin point which is worth your time to contribute to and/or integrate with your project. Trockman et. al [8] looks at the in depth quality of a package by analyzing repository badges eg. Figure 1 that maintainers display on their README file. Trockman not only suggests that badges are signals that makes internal software aspects more transparent but also "may impact users' and contributers' decision making" [8]. Game like elements known as *Gamification* are not explicitly embedded into these badges but in reality motivate users to contribute higher quality code to increase the signal power displayed by these visuals. The key questions here are, *What are the most common badges and what does displayong them intend to signal?* and *To what degree do badges correlate with qualities that developers expect?* [8]. To examine the impact of badges, Node Package Modules (npm) [9] is used as the research repository as it's the largest online collection of Javascript packages.

Data is collected by mining all npm [9] packages and then keeping those with metadata and a public Github [11] repository. They extract the badges through the git history associated with the README file by matching the regular expression for a badge insertion then further classifying them into specific categories such as quality assurance, popularity, dependencies etc. which each have different signaling intentions. With their survey insights, they develop sub questions to validate if the badges exert the correct qualities as intended. For each type of signal (dependencies, popularity, test suite, and quality pull requests), they look at impact before and after badge adoption through longitudinal analysis, statistical regressions to see how it is correlated with their key questions, and any underlying indicators that the badge may not overly express at first glance.

The results suggest that displaying badges highly correlate with better code practices specifically higher test coverage, updated dependencies, and a increase in quality code. However, overwhelming your repository with badges loses its intended signaling effect thus turns away users leading to a decrease in downloads. Assessment dynamic signals that are more costly to produce are more reliable than static conventional badges that display trivial information easily found on the page.

With their survey design research method that collected important libraries metrics from contributers and maintainers, having a low response rate of 15.3% is challenging to generalize that these results will stay reliable for future studies.

2.3 Metrics

With software libaries evolving at a rapid speed, upgrading to the latest version can be cumbersome for developers as challenges of backwards incompatability and deciding on new APIs for your project prove to be a problem. Hora et. al [12] dicusses these 2 obstacles and implements a web application *apiwave* [13] to mitigate this and highlight API popularity and migration in depth with the former measured by the number of users using the service.

The research method takes the Git history code of a repository and outputs information about the API's popularity metric and method of migration with code snippets as examples on this process. This code is further extracted by its Git diff. Based on insertions and deletions, we can detect which lines have been modified for migration and update the popularity statistic by 1 suggesting that the old library lost a follower and the new library gained one. Mining import statement diffs assist with which APIs have been added or removed. The client side of *apiwave* [13] can present a library at many levels including specific interface lookup eg. java.util vs java.util.Map. *Apiwave* also displays addition and deletions statistics, an overall popularity graph, and code snippets that highlights recommended libraries to transfer over to based on diffs.

These results indicate different popularity trends and can answer real life StackOverflow [2] questions regarding API migration in testing with actual human answers. This helps to recommend the best software to use for developers based on compatability through migration and the number of others who support this – popularity.

Even though there is a high number of vistors and page views, it is only limited to the Java language even though there are many lanugages out there such as Javascript and Python that host readily available open source libaries.

Selecting the best suited library for your work is often difficult as you need to make sure that they are not only reliable but also contain your desired features. Mora et al. [14] creates a method of comparison between libraries on a set criteria of metrics to help developers compare and contrast between different software. Mentioned in the article, Uddin et al. [1] proves that aspects such as performance, documentation, and usability also highlighted in Trockman's et al. [8] argument are popular points to look for in an API. Providing a quantative measure that looks at library qualities could lead to the best decision on library selection. Key questions to look at are how to compile this dataset, and finding metrics that express the underlying quality of the library that developers care about.

Data is collected from Stack Overflow [2], a Q&A resource, with source code and issues found on GitHub [11] all to provide diversified content to see the perspective of API qualities at all angles. Many metrics eg. Release frequency, Performance presented by Mora et. al [14] are also dicussed previously by other authors [12, 1, 8, 6]. For each metric a detailed description is given on what the metric portrays, how this metric is calculated by git commit extraction or previous related research from other authors, and how it supports the method of comparison.

The web application displays detailed information all at one place to showcase the different quantitative metrics for each API. Although there is no audience feedback within the article, Mora et al. [14] gives specific future prospects on how to improve the application looking into survey feedback and dynamic updates to the information displayed.

The study is limited to further generalization to show that quanitative measures work without specifying tests in the article.

3 Conclusion

Deciding between different APIs in various libraries and frameworks proves to be challenging. Chen et al. [1] and Udin et al. [6] look at explicitly recommending libraries through their web application with the latter targeting on opinion sentiment. Trockman et al. [8] examines the impact of GitHub [11] badges and whether they influence developer choices. Finally, Hora et. al [12] and Mora et. al [14] analyze quantitative statistics on API metrics with the former focusing on popularity and migration.

For future software metric research based on popular metrics found by Mora et. al [14] and Trockman et al. [8], we hope to create high quality, assessment Github [11] badges based on the most popular metrics to help recommend the best libraries for users through first glance at the README file. Spending more effort on these assessment badges in addition to taking the most popular metrics will ensure that the impact on code quality increases but does not overload the user with too much information on the page.

References

[1] 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.

[2] Stack-Oveflow, "Stack overflow," http://www.stackoverflow.com.

[3] Similartech, "Similartech," https://graphofknowledge.appspot.com/similartech.

[4] Axios, "Axios," https://github.com/axios/axios.

[5] Requests, "Requests," https://github.com/requests/requests.

[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] Opiner, "Opiner," http://sentimin.soccerlab.polymtl.ca:38080/opinereval/.

[8] 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.

[9] NPM, "Npm js," "https://www.npmjs.com/".

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

[11] Github, "Github," "http://github.com/".

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

[13] apiwave, "apiwave," "http://apiwave.com/".

[14] 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.