# The Fault in Our Stars: An Analysis of GitHub Stars as an Importance Metric for Web Source Code

Simon Koch, David Klein, Martin Johns
Technische Universität Braunschweig
{simon.koch,david.klein,m.johns}@tu-braunschweig.de

*Abstract*—**Are GitHub stars a good surrogate metric to assess the importance of open-source code? While security research frequently uses them as a proxy for importance, the reliability of this relationship has not been studied yet. Furthermore, its relationship to download numbers provided by code registries – another commonly used metric – has yet to be ascertained. We address this research gap by analyzing the correlation between both GitHub stars and download numbers as well as their correlation with detected deployments across websites. Our data set consists of $925\,978$ data points across three web programming languages: PHP, Ruby, and JavaScript. We assess deployment across websites using $58$ hand-crafted fingerprints for JavaScript libraries. Our results reveal a weak relationship between GitHub Stars and download numbers ranging from a correlation of $0.47$ for PHP down to $0.14$ for JavaScript, as well as a high amount of low star and high download projects for PHP and Ruby and an opposite pattern for JavaScript with a noticeably higher count of high star and apparently low download libraries. Concerning the relationship for detected deployments, we discovered a correlation of $0.61$ and $0.63$ with stars and downloads, respectively. Our results indicate that both downloads and stars pose a moderately strong indicator of the importance of client-side deployed JavaScript libraries.**

## I. Introduction

When doing web security research, one persistent question is how to select applications to test and how to assess their importance, commonly implying a large developer base depending on the code. While for large-scale web studies, the community has converged on using the Tranco list [1], no such rankings exist for code analysis studies. Past research has used a wide range of different importance metrics to select suitable targets, e.g., GitHub stars and forks [2–8], download counts [5, 6, 9–12] – sometimes interchangeably – or even availability on Bitnami [13, 14] a provider of packaged, ready-to-deploy applications. Nevertheless, the security community has converged on using GitHub stars as the preferred metric; thus, researchers pick applications or libraries that maximize this metric. But so far, there has been no systematic evaluation of how representative GitHub stars are regarding real-world usage.

Different language communities might use GitHub stars differently, developers starring projects because they seem interesting without ever using them, single projects being deployed various times, and the possibility to game the system are sources for bias to creep in. Furthermore, dependency chains complicate the picture. While a developer directly using a dependency might recognize the project by starring it on GitHub, it is unlikely that they are even aware of what projects they are transitively depending on, i.e., what dependencies their dependencies have. The second popular choice is to refer to download numbers provided by code registries to assess how often a project has been deployed. But this is not more reliable, as not every download necessarily represents a new usage instance. Continuous integration services frequently build software, i.e., fetch their dependencies, thus increasing the download count without ever exposing the software to human users.

The only dead sure way to assess the significance of a given project would be to analyze each and every existing machine and count the deployed instances, an obviously infeasible task. We do the next best option by assessing the deployment count of client-side code across publicly available web instances. As this is a cumbersome and lengthy process that is unfeasible to perform independently by every security researcher, we then correlate to the star count and the download numbers to assess which meta-metric is better suited. Finally, we also assess the correlation between download counts and GitHub stars to assess the relationship between the two meta-metrics. We conduct our metadata study for three popular web languages: PHP, Ruby, and JavaScript. Then, we assess the deployment count for client-side JavaScript libraries. Our results show a weak relationship between GitHub stars and download counts across the $925\,978$ projects included in our data set with our deployment assessment for $58$ JavaScript counter-intuitively indicating that either is a moderately strong indicator for deployments. However, we also observe diverging patterns between PHP, Ruby, and JavaScript, indicating that JavaScript presents a special case and a split between server-side code (i.e., node modules) and client-side code has to be made.

To summarize, our contributions are the following:

- We develop a software framework to correlate the importance metrics of GitHub Stars and registry download count and how they measure up against real-world deployments.
- We collect a data set of $925\,978$ projects across the three popular web languages PHP, Ruby, and JavaScript, connecting download numbers with GitHub stars
- We assess real-world deployments of $58$ client-side

Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb) 2024
1 March 2024, San Diego, CA, USA
ISBN 979-8-9894372-2-1
https://dx.doi.org/10.14722/madweb.2024.23004
www.ndss-symposium.org

JavaScript libraries across the top 100 000 Tranco websites We will open source the code for the fingerprinter, the analysis framework, and artifacts upon publication.

Following, we first provide a short background on project importance metrics (II) followed by our methodology (III) and results (IV). We subsequently discuss the results (V), followed by an overview of the related work (VI). Finally, we summarize our key contributions and results in the conclusion (VII).

## II. Background

Sub-fields of the security community deal with security and privacy assessment of program code. Examples are fuzzing, static and dynamic analysis, as well as enforcement techniques. To evaluate such work, the authors showcase their ability to detect security or privacy issues in real software. Here, the question arises of how to select suitable targets to demonstrate the efficacy of the proposed approach. While the web security community has standardized on using the Tranco list [1] as a data set to analyze popular websites, such a consensus is (still) lacking in other areas of security research.

One proxy for a project's importance is called *stars* on GitHub. Despite the fundamental issue that this only captures projects hosted on GitHub, it also abuses an unrelated metric. The official documentation describes stars as a way "to keep track of projects you find interesting and discover related content in your news feed" [15]. Whether considering a project as *interesting* leads to usage, i.e., makes a project an important research target, is an open question. Similarly, ample opportunities exist to game the star-based system [16].

Another proxy for the importance of a project is the download count provided by the corresponding registry. A code registry is a public hoster of code such as *rubygems* [17] for Ruby or *npm* [18] for JavaScript usually combined with a dependency manager that manages dependencies of the current project of a simple list or a more complex configuration file. This central management of dependencies allows the hoster to count the number of downloads a given project enjoys over its lifetime or any other timespan, providing a possibly valuable cache of important metadata. However, this metric might be skewed as the registry cannot ascertain whether a download relates to an update, test deployment, or a new deployment, and thus, download numbers are likely inflated when assessing the number of unique instances being deployed.

## III. Testing GitHub Stars as a Metric

We presented the common hypothesis that GitHub stars or download numbers are correlated to the usage, i.e., importance, of projects. In this section, we present our methodology and corresponding implementation to study this hypothesis.

### A. Hypotheses and Study Pre-Registration

While download counts are available for almost every package repository (e.g., Ruby or npm), real-world usage is more difficult to infer. Relying on self-reporting might work for large projects, e.g., big web frameworks such as Angular, but such user success stories are prone to become outdated and paint an incomplete picture at best. Consequently, we need to derive usage counts from the web at large. For this, fingerprinting is the most promising approach. As we also need comparable languages, we choose three languages enjoying large deployment in the web context: JavaScript, Ruby, and PHP. Given the complexities of fingerprinting server-side code, we only assess the deployment for JavaScript projects.

Based on our language and meta-metric selection, we pose five distinct hypotheses we want to test: **1)** The total download count correlates with the GitHub stars for PHP projects; **2)** The total download count correlates with the GitHub stars for Ruby projects; **3)** The total download count correlates with the GitHub stars for JavaScript projects; **4)** The detected deployment count of JavaScript projects correlates with its GitHub stars; and **5)** The detected deployment count of JavaScript projects correlates with its total download count. We registered our study and hypothesis at the Open Science Framework hosted by the Center for Open Science [19] prior to completing our deployment assessment. The registration provides a timestamp and is publicly available [20]. To account for the issue of testing multiple hypotheses, we apply the Bonferroni correction [21] and adjust our threshold for significance down to 0.01.

### B. Meta Metrics: GitHub Stars and Total Downloads

We selected two meta metrics to compare against each other and for JavaScript against our assessment of deployments: *GitHub Stars* and *Total Downloads*. To gain access to the total download numbers as well as the corresponding GitHub repositories, we leveraged their package manager APIs available at rubygems.org, packagist.org, and npmjs.com. Both Ruby and PHP allow for a complete iteration across all registered projects providing us with the download numbers and corresponding GitHub repository. As directly accessing the list of all npm registered projects is not possible, we base our metadata generation for JavaScript on the latest data set gathered by Pinckney et al. [22] downloaded 2023-12-15. This data set provides us with a list of available npm projects we can then use to obtain download data and GitHub repository from npmjs. We complemented our data with the stars of the corresponding repositories by using the GitHub API. We set our cutoff point for projects at 1000 total downloads to reduce the noise in our data set.

### C. Deployment Metric: Fingerprinting the Web

Deployed server-side code is opaque as the browser only receives its computed results but client-side code is executed in the visitor's browser. Consequently, with a browser we have full access to all application code that makes up the dynamic behavior of the frontend of each website, i.e., the client-side JavaScript. This should, at least in theory, allow us to derive all used client-side libraries via detecting representative patterns, e.g., error messages or specific file names. The real-world situation, however, is more involved, as websites employ bundlers and minifiers to save on bandwidth and obfuscate their application's code. Such minifiers also employ techniques

```
// Does any JavaScript file match?
content.match(/Alpine Expression Error:/g)

// Does any retrieved URL match?
url.match(/alpinejs.*(\.min)?\.js/g)

// Does this expression return, i.e., not throw?
Alpine.version.trim()
```

Fig. 1: Fingerprinting strategies exemplified for Alpine.js

like tree shaking [23] to purge code not actually used inside the web application. Consequently, attempting to recognize used JavaScript libraries in large websites is an error-prone process. Nevertheless, retrieving usage statistics via library fingerprinting gives a useful approximation of deployments.

To this end, we started from *retire.js* [24], a fingerprinting library aimed at detecting outdated libraries with known security vulnerabilities. Generally, the fingerprinter visits a website and records all incoming responses containing HTML or JavaScript. It has three different ways to detect the presence of a library on a website: 1) matching retrieved URLs against known patterns, 2) matching the response's content against known code patterns, and 3) dynamic evaluation of library-specific code. We provide an example for each of the three strategies in Figure 1.

We extended the initial 26 libraries with handwritten fingerprinting code for a total of 58 JavaScript libraries. Out of the initial fingerprints, we were only able to take over 13 as the remaining were either not contained in our GitHub star or downloads data set or posed a high risk of biasing our results as 12 out of the 26 fingerprints were jQuery related; we kept only 4 of them. Furthermore we modified the existing fingerprints to be version agnostic. We base our selection of additional fingerprints on download numbers mirroring the distribution of our JavaScript data set via sampling. Necessarily, we restricted our selection to recognizable and client-side JavaScript libraries. That is, a selected library must both be deployable on a website and also contain unique features, such as descriptive error messages. We selected code aspects for fingerprinting likely to survive the discussed fingerprinting impediments and aimed for several detection mechanisms per library, taking care that the selected ones do not cause false positives.

### D. Calculating the Correlation and Significance

As we cannot expect to encounter a linear correlation for the different hypotheses we want to test we chose a correlation metric capable of dealing with non-linear correlations [25]: Distance Correlation. Distance Correlation is a metric introduced by Székely et al. [26] and can calculate non-linear correlations. Its result domain is in the $0 \dots 1$ range. To calculate the significance of our calculated correlation, we use the same approach as proposed by Székely et al. [26] by randomly reallocating each observation from our first set (e.g., GitHub Stars) to a different observation from our second set (e.g., weekly downloads), without replacement, and recalculating the distance correlation. The ratio of resulting correlations higher than our initially observed one is the p-value of our calculated correlation, i.e., a high value indicates that we cannot rely on the correlation value.

TABLE I: Calculated metadata averages and $\sigma$ for PHP, Ruby, and JavaScript libraries

| Language | #Libs | Avg. ⬇ | $\sigma$ | avg ⭐ | $\sigma$ |
|---|---|---|---|---|---|
| PHP | 90 882 | 1 066 904.14 | 16 465 936.64 | 87.64 | 1235.17 |
| Ruby | 108 409 | 1 308 098.38 | 18 151 830.41 | 148.85 | 1678.77 |
| JavaScript | 726 687 | 10 176 316.43 | 224 280 131.34 | 1398.81 | 8054.25 |

## IV. Results

We explained our acquisition of the meta metrics GitHub Stars and Total Downloads for JavaScript, Ruby, and PHP, as well as our approach to detect deployments of JavaScript libraries on the web. In this Section, we present the results of applying our methodology.

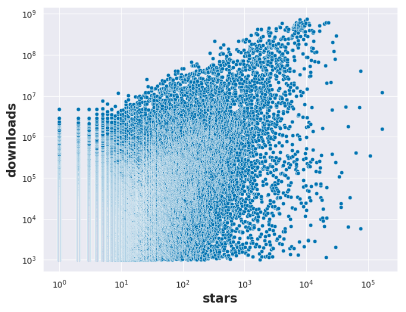
### A. GitHub Stars and Total Downloads

Our initial data set consists of 90 882, 174 197, and 1 178 942 libraries for PHP, Ruby, and JavaScript with at least 1000 total downloads respectively, according to their registries. As not all of those libraries also have a publicly available GitHub repository, we have to reduce the final numbers of libraries down to 90 882, 108 409, and 726 687. We list the calculated averages and corresponding standard deviations in Table I.

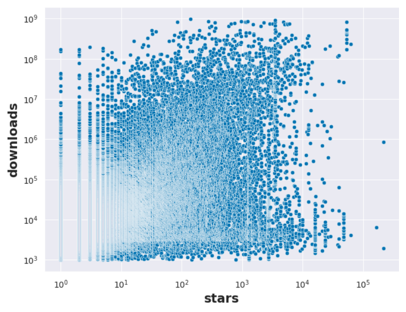
The standard deviations reveal a large spread of total download numbers and stars as either deviation is larger than the corresponding average. This observation gains credence by looking at the median number of downloads, e.g., for JavaScript. According to our data set, the median number of total downloads is 4318, indicating that a large portion of libraries have a download count close to our cut-off of 1000 total downloads. This is in stark contrast to the maximum download count of 31 546 267 200 explaining the large standard deviation. The stars count exhibits a similar pattern with a median of 3 and a max of 216 506. Those observations also hold for PHP with the median values of downloads and stars of 7242 and 2 compared to the maximum of 750 381 350 and 166 157. In contrast, Ruby's median download number of 10 216 is noticeably higher than for the other two languages with a comparable maximum of 216 565. But this difference only holds for downloads as the median of stars is as low as PHP and JavaScript with 2 and the max number of 216 565 is within a rounding error to JavaScript.

### B. Detected Deployments

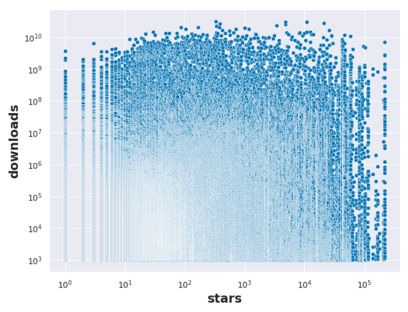
To assess whether downloads or GitHub stars correlate with real-world deployments, we ran our fingerprinter on the landing pages of the 100 000 most popular websites, according to the Tranco ranking, to assess real-world usages [27]. In total, our crawler successfully visited and collected fingerprints from 70.8 % of these websites. We summarize the number of hits reported by our fingerprinter in Table IV.



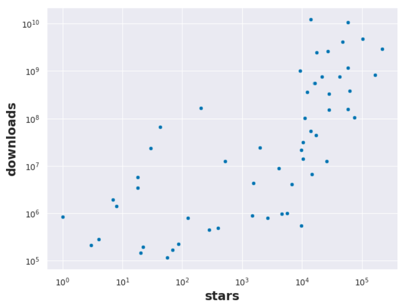
(a) PHP ($corr = 0.47$, $\rho = 0.00 < 0.01$)



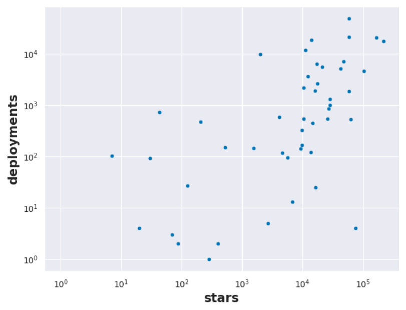
(b) Ruby ($corr = 0.33$, $\rho = 0.00 < 0.01$)



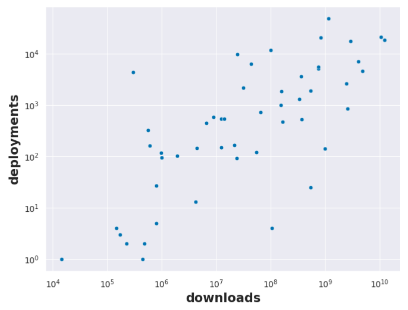
(c) JS ($corr = 0.14$, $\rho = 0.00 < 0.01$)



(d) 🫆 ($corr = 0.52$, $\rho = 0.00 < 0.01$)



(e) 🫆 ($corr = 0.61$, $\rho = 0.00 < 0.01$)



(f) 🫆 ($corr = 0.63$, $\rho = 0.00 < 0.01$)

Fig. 2: Scatter plots of the relation between stars and downloads (a-c) as well as their relation with detected deployments separately plotted for our selection of fingerprinted libraries (d-f). Mind the logarithmic scales.

### C. Correlations

To calculate the distance correlation of our data sets, we used the Python library *statsmodels* version 0.14.1. Calculating the distance correlation is resource-intensive, so we only used a subset of all available data points. We sampled 10 000 data points from our data sets, meaning that for our metadata, we only use a subset of all available data points, while for the fingerprints, we use the whole set of 58. After calculating the correlation, we performed 10 000 random reallocations of our chosen data points to calculate the significance value. We list the results in Table II.

Our collected metadata for JavaScript, Ruby, and PHP only exhibit a weak correlation with a high significance, contradicting a proper relationship between download numbers and GitHub stars, with the weakest correlation for JavaScript and the strongest for PHP. The correlation for PHP is about three times as high as for JavaScript. Concerning the correlation between detected deployments and the metadata, we found a moderately strong relationship around 0.63 for either with a high significance.

TABLE II: Calculated correlations and significance values based on sample of size 10 000 and 10 000 random reallocations.

| Lang. | corr(⭐, 📥) | corr(⭐, 🫆) | corr(📥, 🫆) |
|---|---|---|---|
| JS | 0.14 ($\rho$=0.00) | 0.61 ($\rho$=0.00) | 0.63 ($\rho$=0.00) |
| Ruby | 0.33 ($\rho$=0.00) | N/A | N/A |
| PHP | 0.47 ($\rho$=0.00) | N/A | N/A |

## V. Discussion

The presented results show an overall weak correlation between stars and downloads across languages and a moderate correlation for deployments. To contextualize these results, we first discuss our observations concerning the reliability of the connection between a registry project and its presumed GitHub code repository (V-A), followed by a deep dive into the apparent disconnect between download numbers and GitHub stars (V-B), and the implications of our observed correlations (V-C). Finally, we summarize our key findings and provide an outlook on how to proceed based on our observations (V-E).

### A. Untrustworthy Connection Between Project and Repository

We encountered 1125 (1.24%), 3473 (3.20%), and 33 586 (4.62%) projects that list the same GitHub repository with an average of 2.09, 3.21, and 5.75 projects per repository across PHP, Ruby, and JavaScript, respectively. Depending on where a researcher sources the code, this has implications on the validity of stars or downloads as a reliable metric. If the code is sourced from GitHub, then the star count fits the code, but the download count is shared between the two projects referencing the repository, distributing the count across either project. However, if the code is sourced from the registry, then both the download numbers and the star count are unreliable. The download count is now split between these libraries, while the star count is inflated as multiple projects guide developers to the same GitHub repository. Consequently, to use either number as an importance metric, researchers need to check and discuss whether multiple projects are referencing the same

TABLE III: Amount projects residing in the four different quadrants of our scatter plot (Figure 2). We kept the logarithmic scale while calculating the quadrants.

| Lang. | I | II | III | IV |
|---|---|---|---|---|
| JS | 25 745 (3.5%) | 50 333 (6.9%) | 604 227 (83.2%) | 46 382 (6.4%) |
| Ruby | 2613 (2.4%) | 22 702 (20.9%) | 81 184 (74.9%) | 1910 (1.8%) |
| PHP | 2131 (2.3%) | 23 044 (25.4%) | 65 349 (71.9%) | 358 (0.4%) |

repository, a daunting task. The issue of the reliability of the referenced GitHub repository and its star count also manifests beyond shared references. We encountered instances where the referenced repository did not match the expected repository. Take, for example, *highcharts* a charting package for JavaScript with 165 358 617 downloads but apparently only 203 stars. A closer inspection shows that the npm package references $highcharts - dist$ [28] a self-described “shim repo” by the same owner of the actual repository for highcharts [29]. The proper repository has a 50 times higher star count of 11 683 as of 2024-01-05. Another striking example would be the repository of *react* [30] with 216 885 stars as of 2024-01-05 referenced by the ruby repository *react-source-fb-cloned* [31] with only 1953 downloads. As the original react repository does not contain any Ruby code, we are at a loss as to why it is provided as the reference repository for this Ruby gem. While either example is only anecdotal evidence and follow-up research has to conduct a more in-depth analysis between the connection of the code provided by the project registry and the referenced GitHub repository, they demonstrate that star counts for a given registry project cannot be taken at face value. Therefore, researchers must investigate and sanity-check values before referencing.

## B. Unreasonable Disconnect Between Downloads and Stars

When counting the projects residing in the four different quadrants of our scatter plots (Figure 2), we observe a large aggregation of projects in the third, i.e., lower left, quadrant and a smaller amount of projects in the first, i.e., upper right, quadrant consistently across all languages. This fits a distribution where few have a lot, and most have nothing. However, we can observe a pattern opposing such a distribution when assessing the remaining quadrants. Both Ruby and PHP show a large aggregation of projects in the second quadrant with 20.9% and 25.4% of projects, respectively. This means that a non-trivial number of projects are downloaded frequently but do not receive much attention in terms of stars. We assume this is due to smaller utility projects, i.e., projects use and depend on multiple small projects without the developer acknowledging the usage via stars. Finally, only a minuscule amount of projects reside in the fourth quadrant, with only 1.8% and 0.4% for PHP and Ruby, respectively. This again matches our presumed distribution. JavaScript, however, breaks with this pattern and has a similar amount of projects in both the second and the fourth quadrant with 6.9% and 6.4%, respectively. The second quadrant is noticeably lower than for Ruby or PHP, indicating that either projects that are being used a lot are also recognized more in terms of stars or that there are fewer overall utility projects. As the second explanation directly contradicts past research [32], we believe the first to be true, indicating a different starring culture in JavaScript. However, not only the second but also the fourth quadrant poses an anomaly compared to PHP and Ruby, as the proportion of projects residing here is four and sixteen times higher. This indicates that a non-negligible share of JavaScript projects is renowned but rarely downloaded from npm. We attribute this anomaly to the area of application for the languages: While PHP and Ruby are solely server-side languages, JavaScript is not. For a server-side language, any module or code used has to be included in the project, and it is reasonable to assume that this happens via the corresponding package manager. Each installation, i.e., usage, is then reflected in the download numbers of the registry. Not using a package manager would come with an additional overhead as the code has to be manually downloaded and included in the project, with subsequent updates requiring the same steps again. This usage pattern does not necessarily hold for JavaScript, as websites wanting to use a JavaScript library can either include it directly or reference a hosting URL. The hosting URL can then be from a content delivery network (CDN), the own server, or another source code provider. None of these options are reflected in the download numbers of the package manager. An overview of the amount of projects in the different quadrants is given in Table III.

## C. Correlations

The correlation between stars and downloads is low throughout with 0.47, 0.33, and 0.14 for PHP, Ruby, and JavaScript, respectively. Consequently, they are only loosely connected and cannot be used interchangeably. However, whether this has to be a uniform decision across all languages and for all cases is not obvious. When visually inspecting the scatter plot for PHP provided in Figure 2, the connection between downloads and stars tightens when moving to the larger values. The same holds, even though not as apparent, for Ruby. Thus, either language still has a low connection between download and stars across the lower values. Still, the connection gets stronger the larger the values become, providing an argument that researchers can use either meta-metric interchangeably to represent the significance of a project as long as the values are sufficiently large. JavaScript does not show such behavior, leaving the question of which value – downloads or stars – to use as an importance metric.

Our deployment assessment suggests that both metrics are moderately strong predictors for deployments. This is highly non-intuitive given that JavaScript exhibited the lowest correlation between download numbers and stars with only 0.14. However, it is important to remember that correlations are not necessarily transitive in nature, and thus, a high pairwise correlation between two out of the three pairs is not by itself a contradiction. An explanation for the high correlations could be that we introduced a selection bias into our selection of JavaScript projects to fingerprint. We had to select libraries that can be deployed on the client side and, thus, excluded server-side-only code such as node modules. It is entirely possible that the JavaScript community consists of two sub-communities – web and server – that are different in their starring culture. Assessing this hypothesis and searching for other explanations are tasks of future work.

### D. Limitations

We only look at projects with a download number greater 1000 to reduce noise in our data set. This does introduce a bias towards more known projects and, thus, needs to be kept in mind when assessing the reported observations. When assessing the fingerprinting results it is important to understand that we (a) only looked at client side, (b) JavaScript projects (c) that have some form of identifiable code pattern. This impacts the generalizability of the results as neither PHP nor Ruby support client side deployment and by selecting for libraries with multiple and unique code patterns we are necessarily biased towards larger and more complex projects introducing a bias.

### E. Lessons Learned and Outlook

Our *first lesson learned* is that there is only a weak relationship between download numbers and GitHub Stars proscribing an interchangeable use of either metric. Our *second lesson learned* indicates that a large share of projects is downloaded a lot but not appreciated via GitHub stars, indicating that download numbers are a better indicator of significance than stars. However, the decision of which metric to use is not necessarily uniform across languages as our *third lesson learned* is that we can observe distribution differences between Ruby, PHP, and JavaScript, with Ruby and PHP displaying a large share of downloaded and not appreciated projects but JavaScript does not. Instead, JavaScript has a comparably large amount of projects being appreciated but apparently not downloaded. Our final and *fourth lesson learned* is that when it comes to deployments, both downloads or GitHub stars are a moderately strong predictor and can thus, at least for client-side JavaScript, be used to assess importance.

Based on our observations we recommend to report both numbers – downloads and GitHub Stars – to provide a surrogate metric of the popularity an application enjoys. This allows the community to contextualize the impact of research on less appreciated but frequently used libraries as well as on libraries that are widely renown but not downloaded as much via their registries. Furthermore, researchers need to discuss where they obtained the source code as well as presented metrics to account for duplicate use of repositories or a disconnect between the used GitHub repository and repository listed in the registry.

## VI. Related Work

The selection of targets to test is a widespread question in security research. Consequently, different subfields have come up with their own solutions or suggestions. The initial metric to select websites to analyze was the Alexa list, provided by Amazon [33] and discontinued in 2022 and superceeded by the Tranco list suggested by Le Pochat et al. [1]. The Tranco list solves issues such as possible manipulation of the ranking of specific websites Fuzzing research has standardized on a set of synthetic benchmarks, such as LAVA [34] and an otherwise uniform set of targets [35]. Past research on project popularity involving registries and GitHub is varied. Borges et al. [36] conducted a study on 2,279 popular GitHub repositories, analyzing characteristics such as age, number of commits, number of contributors, and number of forks across multiple languages. However, they already assume that stars indicate popularity as they select the repositories based on star count. Similarly, Aggarwal et al. [37] assess the connection between documentation changes and popularity, with popularity being defined as the sum of stars, forks, and pulls. Syed et al. [38] propose a different measure for popularity – the sum of watchers and pull requests – and study the relationship between it and the amount of code changes. Finally, Zerouali et al. [39] conducted a large-scale analysis in 2018 on 175 774 npm repositories assessing different correlations between metadata. Their results show a low correlation between stars and downloads (0.33) and are somewhat higher but in line with ours (0.14).This past research shows that the options for target selection are rich and that a large variety of assumptions concerning the meaning of popularity have been made. However, an assessment of the connection between the meta metrics for popularity and deployments is still a research gap.

## VII. Conclusion

We presented our correlation analysis between the importance surrogate metrics GitHub Stars and download numbers across 925 978 projects for the web languages PHP, Ruby, and JavaScript. Our results show only a weak relationship. Consequently, we need to rethink how we assess the importance of a given library, as the low correlations between downloads and stars demonstrate that many stars do not necessarily imply usage. Consequently, at least for PHP and Ruby, we have to decide between either using GitHub stars or download numbers. Using a set of handcrafted fingerprints for 58 JavaScript libraries, we discovered 202 077 fingerprints across the Tranco Top 100 000 websites, revealing a moderate yet significant correlation between downloads, stars, and deployments. However, considering we selected only client-side JavaScript projects, this conclusion might not hold up for server-side deployed code. This valuation gains credence by the diverging distribution of projects concerning the combination of stars and downloads across our studied languages. The pure server-side languages show a large proportion of projects enjoying large download numbers but low star counts, and JavaScript as a mixed language shows an increased amount of low download but high star projects. Future work needs to extend our assessment of deployments to include server-side modules and languages. Still, for now, we recommend, at least for client-side JavaScript, using downloads as an importance metric as it has a slightly higher correlation than stars. For any other code reporting both metrics – downloads and GitHub Stars – seems most prudent.

## Acknowledgment

We gratefully acknowledge funding by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2092 CASA – 390781972 as well as from the European Union's Horizon 2020 research and innovation programme under project TESTABLE, grant agreement No 101019206.

## References

[1] V. Le Pochat, T. van Goethem, S. Tajalizadehkhoob, M. Korczynski, and W. Joosen, "Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation," in *Network and Distributed System Security Symposium*, 2019.

[2] M. Musch and M. Johns, "U can't debug this: Detecting javascript anti-debugging techniques in the wild." in *USENIX Security Symposium*, 2021.

[3] S. Khodayari and G. Pellegrino, "It's (DOM) Clobbering Time: Attack Techniques, Prevalence, and Defenses," in *IEEE Symposium on Security and Privacy*, 2023.

[4] J. Bosamiya, W. S. Lim, and B. Parno, "Provably-safe multilingual software sandboxing using webassembly." in *USENIX Security Symposium*, 2022.

[5] R. P. Kasturi, J. Fuller, Y. Sun, O. Chabklo, A. Rodriguez, J. Park, and B. Saltaformaggio, "Mistrust plugins you must: A large-scale study of malicious plugins in wordpress marketplaces." in *USENIX Security Symposium*, 2022.

[6] F. Xiao, J. Huang, Y. Xiong, G. Yang, H. Hu, G. Gu, and W. Lee, "Abusing hidden properties to attack the node.js ecosystem." in *USENIX Security Symposium*, 2021.

[7] J. C. Davis, E. R. Williamson, and D. Lee, "A sense of time for javascript and node.js: First-class timeouts as a cure for event handler poisoning." in *USENIX Security Symposium*, 2018.

[8] S. Koch, M. Wessels, D. Klein, and M. Johns, "Poster: The risk of insufficient isolation of database transactions in web applications," in *ACM SIGSAC Conference on Computer and Communications Security*, 2023.

[9] C. Zuo and Z. Lin, "Playing without paying: Detecting vulnerable payment verification in native binaries of unity mobile games." in *USENIX Security Symposium*, 2022.

[10] W. Li, J. Ming, X. Luo, and H. Cai, "Polycruise: A cross-language dynamic information flow analysis." in *USENIX Security Symposium*, 2022.

[11] Y. Li, Y. Sun, Z. Xu, J. Cao, Y. Li, R. Li, H. Chen, S.-C. Cheung, Y. Liu, and Y. Xiao, "Regexscalpel: Regular expression denial of service (redos) defense by localize-and-fix." in *USENIX Security Symposium*, 2022.

[12] D. Klein and M. Johns, "Parse Me, Baby, One More Time: Bypassing HTML Sanitizer via Parsing Differentials," in *IEEE Symposium on Security and Privacy*, 2024.

[13] G. Pellegrino, M. Johns, S. Koch, M. Backes, and C. Rossow, "Deemon: Detecting CSRF with Dynamic Analysis and Property Graphs," in *ACM SIGSAC Conference on Computer and Communications Security*, 2017.

[14] S. Koch, T. Sauer, M. Johns, and G. Pellegrino, "Raccoon: Automated verification of guarded race conditions in web applications," in *ACM Symposium on Applied Computing*, 2020.

[15] GitHub Docs, https://docs.github.com/en/get-started/exploring-projects-on-github/saving-repositories-with-stars, visited: 2023-12-11.

[16] K. Du, H. Yang, Y. Zhang, H. Duan, H. Wang, S. Hao, Z. Li, and M. Yang, "Understanding promotion-as-a-service on github," in *Annual Computer Security Applications Conference*, 2020.

[17] Rubygems, https://rubygems.org.

[18] npmjs, https://www.npmjs.com.

[19] Center-For-Open-Science, https://www.cos.io/initiatives/prereg, visited: 2023-12-11.

[20] S. Koch, D. Klein, and M. Johns, https://osf.io/gjc52/?view_only=ac745cc43a1e4496be827d4c57d7cceb.

[21] M. H. Herzog, G. Francis, and A. Clarke, *The Multiple Testing Problem*, 2019.

[22] D. Pinckney, F. Cassano, A. Guha, and J. Bell, "Npm-Follower: A Complete Dataset Tracking the NPM Ecosystem," in *ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023.

[23] MDN, https://developer.mozilla.org/en-US/docs/Glossary/Tree_shaking, visited: 2023-12-11.

[24] E. Oftedal, https://github.com/RetireJS/retire.js, visited: 2023-12-11.

[25] V. Kotu and B. Deshpande, "Chapter 4 - classification," in *Data Science (Second Edition)*, second edition ed. Morgan Kaufmann, 2019.

[26] G. J. Székely, M. L. Rizzo, and N. K. Bakirov, "Measuring and testing dependence by correlation of distances," *The Annals of Statistics*, 2007.

[27] V. Le Pochat, T. van Goethem, S. Tajalizadehkhoob, M. Korczynski, and W. Joosen, https://tranco-list.eu/list/3V2KL.

[28] GitHub, https://github.com/highcharts/highcharts-dist, accessed 2024-01-05.

[29] ——, https://github.com/highcharts/highcharts, accessed 2024-01-05.

[30] ——, https://github.com/facebook/react, accessed 2024-01-05.

[31] Rubygems, https://rubygems.org/gems/react-source-fb-cloned, accessed 2024-01-05.

[32] M. Zimmermann, C.-A. Staicu, C. Tenny, and M. Pradel, "Small world with high risks: A study of security threats in the npm ecosystem." in *USENIX Security Symposium*, 2019.

[33] W. Archive, https://web.archive.org/web/20201026074526/https://www.alexa.com/about, accessed 2024-01-07.

[34] B. Dolan-Gavitt, P. Hulin, E. Kirda, T. Leek, A. Mambretti, W. K. Robertson, F. Ulrich, and R. Whelan, "LAVA: Large-Scale Automated Vulnerability Addition," in *IEEE Symposium on Security and Privacy*, 2016.

[35] M. Schloegel, N. Bars, N. Schiller, L. Bernhard, T. Scharnowski, A. Crump, A. Ale-Ebrahim, N. Bissantz, M. Muench, and T. Holz, "SoK: Prudent Evaluation Practices for Fuzzing," in *IEEE Symposium on Security and Privacy*, 2024.

[36] H. Borges, A. Hora, and M. T. Valente, "Understanding the factors that impact the popularity of github repositories," in *IEEE International Conference on Software Maintenance and Evolution*, 2016.

[37] K. Aggarwal, A. Hindle, and E. Stroulia, "Co-evolution of project documentation and popularity within github," in *International Conference on Mining Software Repositories*, 2014.

[38] D. Syed, J. Sessa, A. Henschel, and D. Svetinovic, "Data analysis of correlation between project popularity and code change frequency," in *Neural Information Processing*, 2016.

[39] A. Zerouali, T. Mens, G. Robles, and J. M. Gonzalez-Barahona, "On the diversity of software package popularity metrics: An empirical study of npm," in *IEEE International Conference on Software Analysis, Evolution and Reengineering*, 2019.

## Appendix

TABLE IV: The JavaScript projects included in the fingerprint study with their corresponding download count according to npm, GitHub stars, and number of detected deployments.

| # | Library | ⬇ | ★ | Count |
|---|---|---|---|---|
| 1 | optical-properties | 3 444 780 | 18 | 0 |
| 2 | rapidoc | 894 874 | 1469 | 0 |
| 3 | nua | 836 507 | 1 | 0 |
| 4 | trim-right-x | 7 810 877 | 0 | 0 |
| 5 | non-layered-tidy-tree-layout | 5 745 148 | 18 | 0 |
| 6 | security | 1 397 439 | 8 | 0 |
| 7 | pdfmake-unicode | 214 181 | 3 | 0 |
| 8 | utils-deep-get | 287 564 | 4 | 0 |
| 9 | helix-ui | 116 330 | 56 | 0 |
| 10 | xml-beautify | 195 794 | 22 | 0 |
| 11 | dhtmlx-scheduler | 449 047 | 281 | 1 |
| 12 | measure-text | 225 193 | 87 | 2 |
| 13 | huebee | 484 183 | 393 | 2 |
| 14 | a11y-accordion-tabs | 171 068 | 69 | 3 |
| 15 | reading-position-indicator | 147 253 | 20 | 4 |
| 16 | svelte | 105 885 048 | 74 523 | 4 |
| 17 | avro-js | 793 394 | 2660 | 5 |
| 18 | alasql | 4 146 582 | 6812 | 13 |
| 19 | markdown-it | 548 898 716 | 16 490 | 25 |
| 20 | bitmovin-player-ui | 786 753 | 124 | 27 |
| 21 | froala-editor | 23 790 849 | 30 | 93 |
| 22 | plupload | 1 008 044 | 5617 | 96 |
| 23 | raf-polyfill | 1 927 041 | 7 | 103 |
| 24 | jplayer | 982 241 | 4609 | 119 |
| 25 | tinymce | 54 658 518 | 13 828 | 122 |
| 26 | jszip | 1 002 809 881 | 9278 | 143 |
| 27 | dojo | 4 381 012 | 1535 | 148 |
| 28 | ckeditor | 12 545 370 | 516 | 152 |
| 29 | ember | 598 869 | 0 | 161 |
| 30 | mathjax | 21 681 376 | 9717 | 168 |
| 31 | jquery-mobile | 555 892 | 9708 | 328 |
| 32 | scrollmagic | 6 620 715 | 14 632 | 450 |
| 33 | highcharts | 165 358 617 | 203 | 468 |
| 34 | chart.js | 375 815 970 | 62 604 | 521 |
| 35 | knockout | 13 961 224 | 10 386 | 541 |
| 36 | alpinejs | 12 527 026 | 25 677 | 541 |
| 37 | yui | 8 944 075 | 4105 | 586 |
| 38 | datatables.net | 65 543 631 | 43 | 731 |
| 39 | underscore | 2 608 656 055 | 27 150 | 860 |
| 40 | backbone | 153 704 263 | 28 056 | 993 |
| 41 | vuex | 332 129 241 | 28 300 | 1298 |
| 42 | angular | 157 516 160 | 59 010 | 1858 |
| 43 | mustache | 548 355 018 | 16 128 | 1906 |
| 44 | jquery-validation | 31 576 952 | 10 329 | 2171 |
| 45 | handlebars | 2 448 503 818 | 17 512 | 2646 |
| 46 | dompurify | 359 961 126 | 12 125 | 3588 |
| 47 | axios | 4 802 711 325 | 102 861 | 4602 |
| 48 | vue | 745 375 498 | 42 166 | 5133 |
| 49 | js-cookie | 751 975 585 | 21 303 | 5587 |
| 50 | lazysizes | 43 609 147 | 17 152 | 6336 |
| 51 | moment | 4 058 780 891 | 47 632 | 7034 |
| 52 | jquery-migrate | 24 459 980 | 1966 | 9775 |
| 53 | jquery-ui | 100 676 754 | 11 191 | 11 682 |
| 54 | react | 2 907 303 439 | 216 468 | 17 736 |
| 55 | uuid | 12 217 638 867 | 13 945 | 18 360 |
| 56 | bootstrap | 829 269 792 | 166 274 | 20 580 |
| 57 | lodash | 10 601 765 714 | 58 117 | 21 237 |
| 58 | jquery | 1 147 448 038 | 58 170 | 48 755 |