2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) | 978-1-6654-2705-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/SEAA53835.2021.00057

Reuse Opportunities in JavaScript applications

Anastasia Terzi
Dep. of Electrical & Computer Engineering
University of Western Macedonia
Kozani, Greece
anastassia.terzi@gmail.com

Stamatia Bibi
Dep. of Electrical & Computer Engineering
University of Western Macedonia
Kozani, Greece
sbibi@uowm.gr

Panagiotis Sarigiannidis
Dep. Of Electrical & Computer Engineering
University of Western Macedonia
Kozani, Greece
psarigiannidis@uowm.gr

Abstract— JavaScript nowadays is among the most popular programming languages, used for developing web and IoT applications. Currently, the majority of JavaScript applications is reusing third-party components to acquire various functionalities. In this paper we isolate popular reused components and explore the type of functionality that is mostly being reused. Additionally, we examine whether the client applications adapt to the most recent versions of the reused components, and further study the reuse intensity of pairs of components that coexist in client applications. For this purpose, we performed a case study on 9389 components reused by 430 JavaScript applications hosted in GitHub. The results show that Compiler and Testing Frameworks are the most common types of functionality being reused, while the majority of client applications tend to adopt the recent versions of the reused components.

Keywords—component, JavaScript, reuse, reuse intensity

I. Introduction

Undoubtedly, JavaScript is among the most widely used programming languages for developing web and IoT applications [3]. A recent survey on over 17,000 developers, in 159 countries, between November 2019 and February 2020 appointed that over 12.2 million developers are currently using JavaScript worldwide, rendering the language as the most popular one [2]. The growth and penetration of JavaScript is more or less expected mainly due to the fact that it is a lightweight, highly dynamic language, that can be used for a variety of purposes (i.e. front-end and back-end application development), enjoying the added value of well-funded frameworks like AngularJS, React, and Vue.js. Currently, there is growing evidence that JavaScript code development is becoming more distributed and collaborative [12]. Developers have access to a plethora of available open-source software packages that can be freely used to enjoy the benefits of software reuse.

Software reuse according to McIlroy and M.D [10], is "the process of creating software systems from existing software, rather than building software systems from scratch". The benefits acquired when reusing software involve the minimized development cost, the increased efficiency and maintainability and the improved quality [8]. When it comes to JavaScript application development there are already available a series of package managers (i.e. npm, nexus, yarn) that can facilitate the reuse of JavaScript components. Despite this fact the challenge of selecting the right functionality to be reused and determining the appropriate components that will synthesize the newly developed application remains.

In this paper, we address this challenge by investigating the available JavaScript components that are most commonly being reused along with the functionalities that they offer. Our goal is to shed light on the reuse opportunities offered in the context of JavaScript development. For this purpose, we performed an exploratory case study on 9389 JavaScript components retrieved from the GitHub[1] repository. This case study investigates:

- **JavaScript components functionality**: Considering the fact that software reuse is more efficiently performed within the same application domain [11], we investigate the availability of the components with respect to the functionality that they offer. The functionalities are extracted from the description of the components as found in the hosting websites. The studied functionalities involve compilers, development frameworks, Testing Frameworks, user interface components, and interoperability units.
- **JavaScript components coexistence intensity**: Exploiting the full benefits of reuse, most of the time, practitioners, tend to reuse simultaneously a variety of components that serve different purposes It is important to identify common pairs of components that are frequently reused together in the context of a single application. Such evidence can help developers decide upon the functionalities that can be reused within the same context and complement each other.

Current literature on engineering JavaScript applications focuses on trends related to development frameworks [4], the language features [5], and the dynamics of the language [1]. When it comes to the potentials of reuse the research is limited. Kikas, R., Gousios, G., Dumas, M. and Pfahl, D. [6] studied the dependency network formed by the applications that reuse components and examined the consequences caused by the removal of a popular component. Also, Li et al. [9] proposed a framework to reuse JavaScript code snippets found in question and answer websites. This study differentiates from the aforementioned since a) we emphasize the reuse of components implementing a variety of functionalities and not just frameworks and b) we identify common pairs of components that are reused simultaneously in the client application.

The rest of the paper is organized as follows: In Section II we present the study design in the form of a case study protocol. In Section III we provide the results, organized by the research question, and discuss them in Section IV along with the threats to the validity of our study. In Section V, we conclude the paper.

[1] https://github.com/search?l=JavaScript&o=desc&p=5&q=JavaScript&s=forks&type=Repositories

II. CASE STUDY DESIGN

In this section, we present the protocol that has been adopted for designing this case study according to the guidelines of Runeson and Höst [13]

A. Research Questions

The goal of this case study is to identify open-source components with respect to their reusability from the point of view of software engineers in the context of JavaScript application development. In order to achieve this goal, we decompose the goal to three research questions:

[RQ1]: Which JavaScript open-source components are mostly being reused with respect to the functionality offered?

This research question aims at identifying highly reusable JavaScript components and recording the functionality that they offer. The analysis will provide an overall view of the types of functionalities that are highly reused within the scope of JavaScript application development.

[RQ2]: What is the reusability of open-source JavaScript components with respect to their version?

This question examines the reused components with respect to the version that is being reused. Our target is to reach a conclusion on whether it is necessary to reuse the latest version of a JavaScript component. The analysis will provide insight on whether client applications need to immediately absorb changes in the reused components in order to produce a stable operating environment.

[RQ3]: What is the intensity of popular open-source JavaScript reusable pairs of components?

This question aims to identify the pairs of JavaScript components that are frequently reused in the context of a single application. The results of this research question are expected to provide insights on common practices when selecting open-source JavaScript components for reuse.

B. Data Collection and Analysis

The case study of this paper use the 430 most forked JavaScript projects hosted in GitHub by February 2021 We selected applications that have at least a two-year period lifespan, present more than 10 releases, and have at least a new version released in the past year. For each JavaScript project, we have downloaded the file "package.json" and recorded information relevant to the project's dependencies. The project's dependencies indicate the components that are being reused by the particular project. In total, we identified 9389 reused components. The first set of metrics used in the scope of this study are in the **component-level.** Most of these metrics come from the metadata provided by "package.json" files, these include the

- ***name of the client application*** that the component is being reused
- ***name of the reused component***,
- ***version of the reused component***.
- Additionally, for each component, we recorded the type of ***functionality*** that it offers. Based on the descriptions given on the GitHub repository of each component, we concluded on a set of 5 types of functionality such as *frameworks* that provide an integrated environment for developing js applications, *testing frameworks*, *compilers* used for browser compatibility, *user interface* components and *interoperability* components, that are used for connecting different third-party applications.

The second set of metrics calculated are related to the popularity and the intensity of the reused components within the context of the **JavaScript application development ecosystem** as proposed by Kula, R.G., De Roover, C., German, D.M., Ishio, T. and Inoue, K. [7]. In this context we calculated the following metrics:

1. ***UsedBy*** indicates the components $\{v1, v2, \ldots, vn\}$ that reuse the component u. For example UsedBy(eslint)={novnc, shelljs, atom,webpack}

$$UsedBy(u) \equiv \{v \mid v \rightarrow u\} \qquad (1)$$

2. ***Popularity*** for a component (u) indicates the number of ***UsedBy*** relationships .

$$popularity(u) \equiv |UsedBy(u)| \qquad (2)$$

3. ***Popularity of coexistence pairs*** is calculated for pairs of reused components (u,v) and indicates the number of times that components u,v commonly exist in ***UsedBy*** relationships.

$$popularity(u, v) \equiv |UsedBy(u) \cap UsedBy(v)| \quad (3)$$

4. ***Intensity*** is the normalized frequency count of popular pairs. For a given set of reused components *I* for pairs *x*, *y* ϵ *I* we define intensity as following:

$$intensity(x, y, I) = \frac{popularity(x, y)}{\underset{\substack{i,j \in I \\ i \neq j}}{max}(popularity(i, j))} \qquad (4)$$

where $x, y \in I$ and *max* returns the number of times that the most popular pair of components is being reused.

Regarding the data analysis methods employed, for RQ1 we present the related descriptive statistics regarding the reused components identified per functionality type. Additionally, we present the details of the most popular components that explain more than 70% of the variance. For RQ2 we present a stacked bar chart presenting the most popular reused components, where each bar represents the different versions of a component that is being reused. For RQ3 we present a heat-map presenting the Intensity of the most popular coexistence pairs of reused components.

III. RESULTS

This section, presents the results of the case study per RQ.

RQ$_1$: Which JavaScript open-source components are mostly being reused with respect to the functionality offered?

Table I presents the most popular JavaScript components, along with the functionalities that they implement. The components presented explain over 70% of the total variance. It can be observed that *Compilers*, followed by *Testing Frameworks* and I*nteroperability Unit*s components are among the most reused ones. Table II presents the basic descriptive statistics obtained by splitting the reused components based on the functionality that they implement.In terms of the maximum components offered per functionality type, we observe that the maximum value again exists for *Interoperability Units* and *Compilers* whereas the least components per functionality type are found in *User Interface* Category.

***RQ$_2$*: What is the reusability of open-source JavaScript components with respect to their version?**

Fig. 1 presents the distribution of different versions of the components that are commonly used in JavaScript apps. The colours represent the versions of each component. The darker the shade the newer the version and vice-versa. There seems to be no pattern on why creators choose to update to a newer version or use an older one. Overall in JavaScript projects it is important to keep up with the latest version of Node.Js (if the project uses npm) or to at least update to a version that is still under maintenance. Therefore programmers need to update to newer versions of components that support the changes on Node.js. Although this technique is recommended the main disadvantage is the lack of compatibility between multiple components used on the same project, after the migration to the latest version. After research, we concluded that in projects with multiple third party components of different functionalities the creators tend to use stable older versions while in projects that use one or two components or only components with the same functionality, programmers are migrating constantly to the latest version.

***RQ$_3$*: What is the intensity of popular open-source JavaScript reusable pairs of components?** To calculate the intensity of the popular pairs we first run Spearman analysis. Using the results of the analysis, we calculated the *intensity metric* for all the pairs and construct the heat-map of Fig.2 to visualize the results. The colours represent the intensity of the relationship. We used red colour to represent high intensity and green colour for low intensity. The results of the analysis point that the pairs of components with high coefficient level, significant relationship and high-intensity values are those consisted of:

- Compiler and Testing Framework
- Compiler and Interoperability Unit
- Interoperability Unit and Testing Framework.
- Framework and Interoperability Unit

That kind of result was expected since JavaScript is a language used on developing web and IoT applications and runs on multiple platforms by combining frameworks, plugins, and third-party code. To overcome problems of compatibility and solve issues between parties, programmers need to use compilers, middleware, and Testing Frameworks.

I. POPULARITY AND FUNCTIONALITY OF THE MOST REUSED COMPONENTS

Component	*Popularity*	*Percent*	*Functionality*
Babel	1157	12.1	Compiler
Eslint	978	10.2	Testing Framework
Karma	599	6.3	Testing Framework
Grunt	551	5.8	Interoperability Unit
Rollup	419	4.4	Interoperability Unit
Webpack	339	3.6	Interoperability Unit
Gulp	324	3.4	Compiler
Mocha	194	2.0	Testing Framework
Typescript	176	1.8	Interoperability Unit
React	171	1.8	User Interface
Sinon	129	1.4	Testing Framework
Chai	117	1.2	Framework
Prettier	105	1.1	Compiler
Rimraf	101	1.1	Interoperability Unit
Lodash	99	1.0	Compiler
Express	87	0.9	Framework
Semver	84	0.9	Compiler
Glob	81	0.8	Interoperability Unit
Nyc	77	0.8	Interoperability Unit
Chalk	73	0.8	Interoperability Unit
Browserify	72	0.8	Interoperability Unit
Cross-env	70	0.7	Framework
Jest	69	0.7	Testing Framework
Fs-extra	68	0.7	Interoperability Unit
Vue	66	0.7	Framework
Jquery	63	0.7	User Interface
Uglify-js	62	0.6	Interoperability Unit
Lint-staged	61	0.6	Testing Framework
Postcss	60	0.6	User Interface
Coveralls	59	0.6	Testing Framework
Core-js	57	0.6	Framework
Css-loader	57	0.6	User Interface
Jasmine	57	0.6	Testing Framework
Sass	57	0.6	User Interface

II. BASIC DESCRIPTIVE STATISTICS OF COMPONENTS BASED ON USEDBY METRIC AND PER FUNCTIONALITY TYPE

Functionality Type	*N*	*Minimum UsedBy value*	*Maximum UsedBy value*	*Mean UsedBy value*
Interoperability Unit	48	17	101	30.04
Compiler	31	19	1157	130.77
Testing Framework	24	17	978	111.04
Framework	15	16	87	44.73
User Interface	15	16	171	46.47

II. DISCUSSION

A. Implications to researchers and practitioners

The results of this study provide useful information and guidance for **practitioners** on planning the reuse of components in the context of JavaScript application development. In particular, some take away messages are:

- When examining opportunities for reuse, practitioners can consider reusing functionality related to *Interoperability Units*, *Compilers,* and *Testing Frameworks* that are mostly implemented by most of the available components.
- Regardless of the availability of components the most reused functionality on average we observe that *Compilers*, *Testing Frameworks,* and *User Interface* components are the ones more frequently reused. This fact shows that there is an interest in reusing libraries that can overcome the problem of compatibility with different browsers, instead of dealing with these problems at first hand.

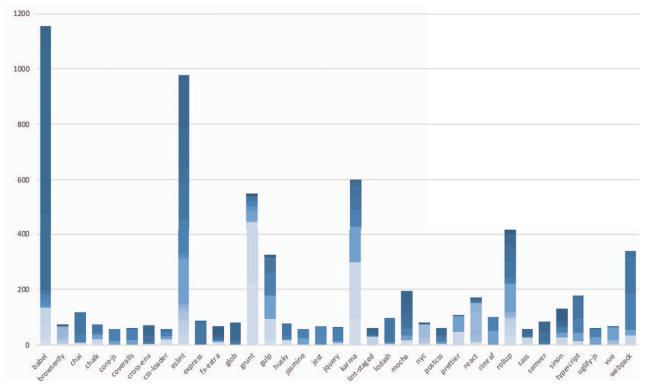


Fig 1. Distribution of different versions Used by components

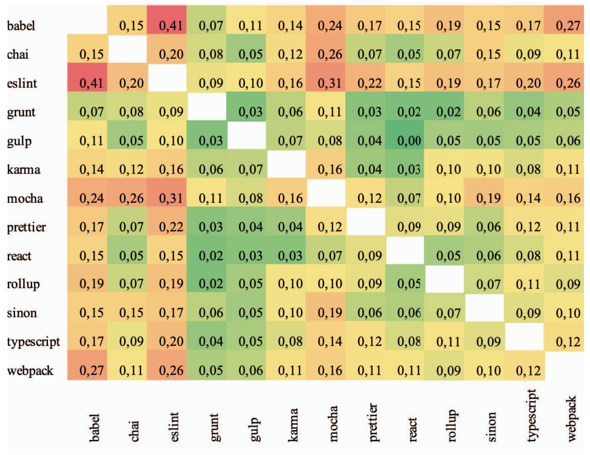
- Regarding the need to update the reused component, we diversify between projects that present significant reuse more than 2 components and projects that present limited reuse and advise practitioners to adopt a) in the first case, the most recent stable version (not the latest one) to be able to handle dependencies among the reused components more efficiently and b) in the second case, migrate to the new versions and keep up with the changes in the related dependency manager (i.e npm), since in that case the risk is limited due to the small number of reused components.
- Regarding the potential to reuse pairs of components, we advise practitioners to combine Compiler with Testing Frameworks or Interoperability and Frameworks, since all work under the same scope, targeting to solve issues with third-party components.

Based on the results of this case study, we encourage researchers to:

- Perform empirical studies on the reused JavaScript components. Currently, there is a large repository of reusable components that can offer great opportunities for reuse. Therefore, it is important to guide the software industry on how to design JavaScript applications to maximize the benefits of reuse.
- Examine the quality of the reusable components to be able to check prior to adoption whether the components will introduce vulnerabilities and jeopardize the maintenance process of the client application.

B. Threats to validity

In this section, we discuss the threats to validity which we have identified for this study, based on the categorizations presented in [13]. Regarding Construct Validity, we should mention that we adopted a set of reuse metrics that are targeted to measure reuse within a software ecosystem (i.e. Github) [7]. Our rationale behind selecting these metrics was based on content and scope similarities with [7]. Though we plan to examine the evaluation of non-selected alternative metrics as future work. Regarding Internal Validity, in this study we do not attempt to identify causality relationships, therefore the threat is not applied. Concerning reliability, we believe that the replication of our research is safe since the process that has been followed in this study has been thoroughly documented in Section II. The only part where subjective opinion is inserted is in the classification of the reusable component into a type of functionality. Most of the time this was performed by



	babel	chai	eslint	grunt	gulp	karma	mocha	prettier	react	rollup	sinon	typescript	webpack
babel		0,15	0,41	0,07	0,11	0,14	0,24	0,17	0,15	0,19	0,15	0,17	0,27
chai	0,15		0,20	0,08	0,05	0,12	0,26	0,07	0,05	0,07	0,15	0,09	0,11
eslint	0,41	0,20		0,09	0,10	0,16	0,31	0,22	0,15	0,19	0,17	0,20	0,26
grunt	0,07	0,08	0,09		0,03	0,06	0,11	0,03	0,02	0,02	0,06	0,04	0,05
gulp	0,11	0,05	0,10	0,03		0,07	0,08	0,04	0,00	0,05	0,05	0,05	0,06
karma	0,14	0,12	0,16	0,06	0,07		0,16	0,04	0,03	0,10	0,10	0,08	0,11
mocha	0,24	0,26	0,31	0,11	0,08	0,16		0,12	0,07	0,10	0,19	0,14	0,16
prettier	0,17	0,07	0,22	0,03	0,04	0,04	0,12		0,09	0,09	0,06	0,12	0,11
react	0,15	0,05	0,15	0,02	0,03	0,03	0,07	0,09		0,05	0,06	0,08	0,11
rollup	0,19	0,07	0,19	0,02	0,05	0,10	0,10	0,09	0,05		0,07	0,11	0,09
sinon	0,15	0,15	0,17	0,06	0,05	0,10	0,19	0,06	0,06	0,07		0,09	0,10
typescript	0,17	0,09	0,20	0,04	0,05	0,08	0,14	0,12	0,08	0,11	0,09		0,12
webpack	0,27	0,11	0,26	0,05	0,06	0,11	0,16	0,11	0,11	0,09	0,10	0,12	

Fig 2. Heat-map of Coexistence component pairs

isolating keywords from the description of the component in the hosting repository. This process was performed by the first author and the results were verified by the second author. Concerning the external validity and in particular the generalizability supposition, changes in the findings might occur if we altered samples of the projects studied. Future replication of this study in other sets of JavaScript projects would be valuable to verify these findings.

V. Conclusions

In this paper, we have performed a case study on 9389 components reused by 430 JavaScript applications hosted in the GitHub repository. Our goal was to identify popular reused components and explore the type of functionality that is mostly being reused. We examined the reused components with respect to the version that is being reused. As a final step, we studied the reuse intensity of coexistence pairs. The results show that Compiler and Testing Frameworks are the most common types of functionality coexisting, while the majority of client applications tend to adopt the recent versions of the reused components.

Acknowledgments

This paper has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No. 957406 (TERMINET)

References

1. Axel Rauschmayer and Amazon.com (Firm (2012). The past,present, and future of JavaScript : where we've been, where we are, and what lies ahead. Sebastopol, Ca: O'reilly Media.
2. Carraz, M., Korakitis, K., Crocker, P., Muir, R. and Voskoglou, C.: Developers Economics: State of the Developer Nation. 18th ed. SlashData Ltd. (2020)
3. Chatzimparmpas, A., Bibi, S., Zozas, I. and Kerren, A. (2019). Analyzing the Evolution of JavaScript Applications. Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering.
4. Delcev, S. and Draskovic, D. (2018). Modern JavaScriptframeworks: A Survey Study. Zooming Innovation in Consumer Technologies Conference (ZINC). pp.106-109.
5. Gude, s., Hafiz, M., Wirfs-Brock, A. 2014. JavaScript: The Used Parts, 2014 IEEE 38th Annual Computer Software and Applications Conference, Vasteras, Sweden, 2014, pp. 466-475.
6. Kikas, R., Gousios, G., Dumas, M. and Pfahl, D. (n.d.). Structure and Evolution of Package Dependency Networks.
7. Kula, R.G., De Roover, C., German, D.M., Ishio, T. and Inoue, K.(2018). A generalized model for visualizing library popularity, adoption,and

diffusion within a software ecosystem. 2018 IEEE25thInternational Conference on Software Analysis, Evolution and Reengineering (SANER).

8. Leach, R.J. (n.d.). Methods of Measuring Software Reuse for the Prediction of Maintenance Effort. Software Maintainance – Research and Practice, Volume 8(Issue 5), pp.309-320.
9. Li, X., Wang, Z., Wang, Q., Yan, S., Xie, T. and Mei, H. (2016). Relationship-aware code search for JavaScript frameworks. Proceeding of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering.
10. McIlroy and M.D(1968). Mass Produced Software Components. Software Engineering, NATO Science Committee.
11. Paschali, M.-E., Ampatzoglou, A., Bibi, S., Chatzigeorgiou, A. and Stamelos, I. (2016). A Case Study on the Availability of Open-Source Components for Game Development. Lecture Notes in Computer Science, pp.149–164.
12. Reid, B., Barbosa, K., d'Amorim, M., Wagner, M. and Treude, C(2021). NCQ: code reuse support for Node.js developers. CoRR. 4 Jan
13. Runeson, P. and Höst, M. (2008). Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering, 14(2), p.131.