An Empirical Investigation of Forks as Variants in npm

John Businge,* Alexandre Decan,† Ahmed Zerouali,‡ Tom Mens† and Serge Demeyer,*
*University of Antwerp, Antwerp, Belgium
†University of Mons, Mons, Belgium
‡ Vrije Universiteit Brussels, Brussels, Belgium

CEUR-WS.org/Vol-2912/paper1.pdf

***Abstract*—Software developers often need to create variants to accommodate different customer segments. These variants have a common code base but also comprise variant-specific code. A common strategy to create a variant is to clone&own (or fork) an existing repository and then adapt it to the new requirements. This form of reuse has been enhanced with the advent of social-coding platforms such as GitHub, and package distribution platforms like npm. GitHub offers facilities for forking, pull requests, and cross-project traceability. npm offers facilities for managing package release dependencies and dependents on the distribution platform. Little is known about the maintenance practices of the variants. We therefore performed an exploratory investigation on the evolution of variants, focusing on their technical aspects. We collected variants from the JavaScript ecosystem, whose sources are hosted on GitHub, and whose packages are released on npm. We have identified a total 12,813 variant forks from the JavaScript ecosystem. In general, we observed that mainlines have more number of package releases, package dependencies, dependent packages and dependent projects compared to their variant counterparts. However, it is still interesting that some variants have quite a considerable number of package releases and dependent packages/projects; in a some cases even more than their mainline counterparts.**

***Index Terms*—software variants, npm, depandencies, software ecosystems**

I. Introduction

To develop quality software faster, developers rely on code reuse and distributed collaborative development tools. Social coding platforms like GitHub have substantially improved both code reuse and collaborative development, providing a huge bazaar of software projects and components that can be reused through explicit project dependencies or forking of software repositories. This is supported by various automated facilities such as pull requests, dependency management tools, issue tracking systems (e.g. JIRA), source code review tools (e.g., Gerrit), Q&A services (e.g. StackOverflow), continuous integration tools (e.g., Travis CI), and package distribution managers (e.g., npm). Social coding platforms comprise a number of software ecosystems i.e., large collections of interdependent software components that are maintained by large and geographically distributed communities of collaborating contributors [1], [2]. These ecosystems form large socio-technical networks of technical artefacts and social actors that interact with each other on top of common software and hardware platforms. The unprecedented growth of these ecosystems relies on substantial software reuse using different methods and tools [3].

Our research focuses on the phenomenon of *forking* in particular. Forking a software repository (referred to as the *mainline*, i.e., the original repository) produces several *forked* repositories. Two types of forks exist [4]. *Social forks* are created for isolated development, but with the goal of contributing back to the mainline. *Variant forks* are created for splitting off a new development branch, often to steer the development into another direction than the mainline, without the intention to contribute back. Variant forking may split the core development team and always splits the contributing community. Variant forking creates variants of the mainline repository, which share common code, but also contain variant-specific code that needs to be maintained. A mainline repository together with all its variant repositories can be seen as a software product family, inspired by the notion of *software product lines* [5]. The family members have software artefacts in common, but also contain artefacts that are specific to one or multiple variants.

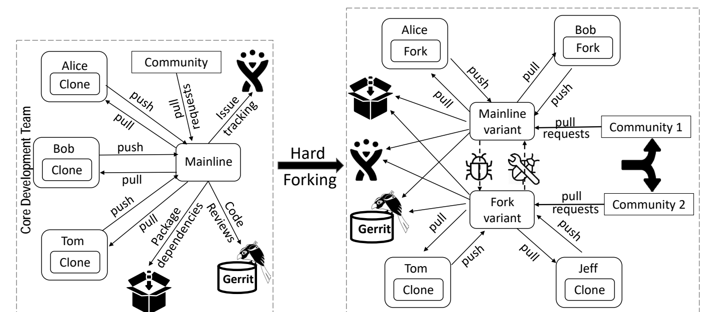

Fig. 1: Maintenance of a software repository before (left) and after (right) variant forking.

Figure 1 illustrates the development activities on a repository before and after variant forking. On the left, three core developers (Alice, Bob and Tom) have write access to the mainline repository. The community interacts with the mainline by sending pull requests, submitting issues and conducting change reviews (social forking). After variant forking, shown on the right, the core developers are split into two. Alice and Bob remain with the original mainline, while Tom is joined by a new developer Jeff to maintain a new fork variant in parallel to the mainline project. This parallel development between mainline and variant has split the contributing community into *Community 1 & 2*. New contributors could decide to contribute

Copyright 2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

to the mainline or the variant, for instance depending on which one is more open to accommodate newcomers. Furthermore, as a result of parallel maintenance, developers in one of the projects may identify and fix bugs in shared artefacts. Developers in each variant of the family are not obligated to contribute back but if they wish, the fixes could be propagated to other members of the family to avoid effort duplication.

There are many studies on variant forks. However, most of these studies were carried out on SourceForge [6]–[11], before the advent of social coding platforms such as GitHub. We have found only two studies that investigated variant forks on GitHub [4], [12]. While the pre-GitHub studies report controversial perceptions around variant forks [11], [13]–[17], Zhou et al. [4] reports that these perceptions have changed with the advent of GitHub. Jiang et al. [18] state that, although forking is controversial in traditional open source software (OSS) communities, it is actually encouraged as a built-in feature in GitHub. Jiang et al. [18] further reports that developers fork repositories to submit pull requests, fix bugs, add new features and keep copies (social forks). Zhou et al. [4] also report that many variant forks actually start as social forks.

While numerous studies have investigated variant forking, we are not aware of any study that has investigated the socio-technical specificities of these variant forks that are part of **software ecosystems**. Our research therefore aims at empirically investigating the socio-technical evolution of **software families** within these software ecosystems, that are composed of variant project repositories. Specifically, we aim to investigate the evolution of socio-technical specificities of software families in npm for JavaScript packages whose repositories are hosted on GitHub. For the social aspects, it is interesting to study the interaction and collaboration between communities involved in the mainline and variants. For example, finding out how different are collaborations with respect to who forked and how the fork has been created? For the technical aspects, ecosystems are often characterised by numerous dependencies between software packages released on package distribution platforms [19]. As a result of parallel development of the mainline and its variants, while offering similar functionality, we aim to investigate if other packages/projects (outside the family) depending on them may migrate from one variant of the required package to another in a family.

Before stating our research goal and research questions, let us first discuss the terminology we will use in our analysis and also provide a concrete example:

- **Package.** A reusable software component that is distributed through the npm package manager.
- **Mainline.** A repository that is hosted on GitHub whose package releases are distributed on npm.
- **Variant.** A fork repository of the mainline that is hosted on GitHub whose package is distributed on npm.
- **Software family.** A set of two or more repositories (the mainline and its variants) that are hosted on GitHub with the package releases of both mainline and its variants distributed on npm.
- **Release.** A specific package version that is publicly distributed on npm.
- **Dependency.** A package that is required for the proper functioning of an other package.
- **Dependent package.** A package A depending on a package B is a "dependent package of B". By definition, both A and B are distributed in a package distribution platform.
- **Dependent project.** A project is a repository in which a package is developed, but not necessarily distributed in a package distribution platform. By extension, a project A depending on a package B is a "dependent project".

To put the terminologies in perspective, let us present a concrete example of a software family. The mainline repository `blackjk3/react-signature-pad` is hosted on GitHub (with a total of 119 forks and six contributors as of December 02, 2020). A signature pad implementation for `react`. The mainline and 2 of the 119 forks (`itgjz/react-signature-pad` and `agilgur5/react-signature-canvas`) have their package releases distributed on npm. Table I presents statistics corresponding to the package releases, package dependencies, dependent packages and dependent projects for three repositories in the software family. From previous studies in the Android ecosystem, we observed that most forks are only active active for a limited period of time [20], [21]. Most of them are mainly social forks that are created to fix a bug or introduce a new feature and then stopped.

TABLE I: Count of the mainline and variants releases, dependencies, dependent packages and dependent projects. M = mainline and F = fork.

	releases	dependencies	dependent packages	dependent projects
blackjk3/react-signature-pad (M)	6	8	9	7
itgjz/react-signature-pad (F)	8	8	0	0
agilgur5/react-signature-canvas (F)	25	23	31	0

II. Goal and Research Questions

This research is the first of its kind in studying socio-technical specificities of the variant forks that are part of **software ecosystems**. In this study, our goal is to perform an exploratory investigation on the evolution of variants focusing on their technical aspects. We mined repositories from the JavaScript ecosystem, whose sources are hosted on GitHub, having their package releases on npm. This allows us identify the software families as well as studying the technical aspects of those families. We state the four research questions:.

RQ_0 *How prevalent are software families in the JavaScript ecosystem on GitHub?* We would like to determine whether software families exist in software ecosystems. If software families rarely exist, results about their socio-technical evolution may not be statistically significant.

RQ_1 *How do the distributions of package releases in mainlines and their variants compare to each other?* This RQ will help us determine if mainlines and variants are

continuously maintained. A package that is continuously distributing new releases means that it introduces new features and addresses issues raised by its users.

RQ_2 *How do the distributions of package dependencies in mainlines and their variants compare to each other?* This RQ will help us determine if the mainlines and their variants depend on other packages in the npm ecosystem. If they have dependencies on other packages, then studying the dependency relationships between the packages in the software families and other packages would be interesting.

RQ_3 *Do variant packages exhibit other dependent packages/projects than the Mainline?* This RQ will help us determine variations in dependencies within product families. If a variant has other dependents than the mainline, it may be an indication of which features are demanded in the ecosystem.

III. Methods and Dataset

According to the 2020 Stack Overflow Developer Survey[1] to which over 65,000 developers participated, JavaScript is one the most commonly used programming language (67.7% of all respondents make use of JavaScript).

Our dataset of GitHub repositories from the JavaScript ecosystem and their corresponding npm packages, dependents and dependencies were extracted from the `libraries.io` release 1.6.0 of January 12, 2020. We use the following heuristics to extract the mainline–variant pairs of the software families in the JavaScript ecosystem on GitHub: the repository A is a variant of the repository B if (1) the package releases of A and B are distributed on npm and (2) A is a fork of B on GitHub. Since we focus on the JavaScript ecosystem on GitHub, we extracted software families whose repositories are hosted on GitHub and their packages distributed on npm. For each mainline and its corresponding variants we collected their releases, dependencies and dependents (projects and packages).

IV. Results

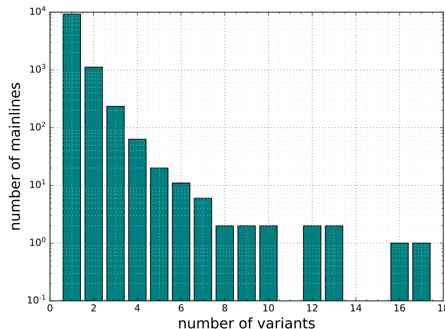

Fig. 2: Family size (number of variants in a family).

RQ_0: How prevalent are software families in the JavaScript ecosystem on GitHub?

With this first RQ, we aim to determine if software families exist in software ecosystems. In the JavaScript ecosystem we discovered a total 10,743 distinct mainlines and 12,813 variants in total. This means that we have a total of 10,743 software families. Figure 2 presents a histogram of the distribution of the number of variants per mainline. The y-axis (in log-scale) shows the number of mainlines and the x-axis shows the number of variants per mainline. For example, the first bar tells us that there are 9,280 mainlines that have only one variant. We also observe two mainlines having 16 and 17 variants. The results of RQ_0 reveal that software families indeed exist in the considered ecosystem.

> *We have identified 10,743 distinct mainlines and 12,813 variants. This shows that software families exist in the JavaScript ecosystem on GitHub.*

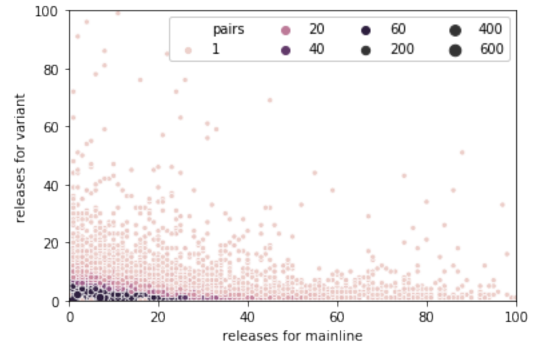

Fig. 3: Distribution of the mainline versus variant package releases.

RQ_1: How do the distributions of package dependencies in mainlines and their variants compare to each other?
Figure 3 presents a scatter plot of releases for the variants versus the releases for the mainlines. The x-axis shows the number of releases for the mainlines and the y-axis shows the number of releases for the variants. The color of the data points represents the number of mainline–variant pairs. For example, darkest circle around the point (1,1) tells us that there are over 600 mainline–variant pairs that have one release each in the data point. The lightest circles tell there is only one mainline–variant pair in the data point. For example the point (100, 1) tells us that there is one mainline–fork pair where the mainline has 100 releases and the variant has only one release. The data points along the y-axis tell us that there are some mainline–variant pairs where the variants have more releases compared to the mainline. This implies that these specific variants are being maintained more than their mainline counterparts. Overall, we observe more mainlines being maintained compared to their variant counterparts. However, we also observe a significant amount of variants being maintained. This is interesting since developers variants did not make a one off package distribution; they are continuously distributing new releases of their package. Table II shows some examples of variant releases that are maintained more than their mainline counterparts.

[1]https://insights.stackoverflow.com/survey/2020

TABLE II: Example of variants that have more package releases compared to their mainline counterparts.

mainlines	variants	mainline dependent packages	variant dependent packages	diff
weex-pack	weexpack	1	129	128
restyped-giphy-api	restyped-staffjoy-api	1	116	115
cogs-javascript-sdk	cogs-sdk	5	104	99
gulp-galen	gulp-galenframework	11	99	88

While many mainlines are being maintained more that their variant counterparts (which is not surprising), we also observe a good number of variants being maintained in parallel. Interestingly we also observed a good number of variants that are more actively maintained than their mainline counterparts.

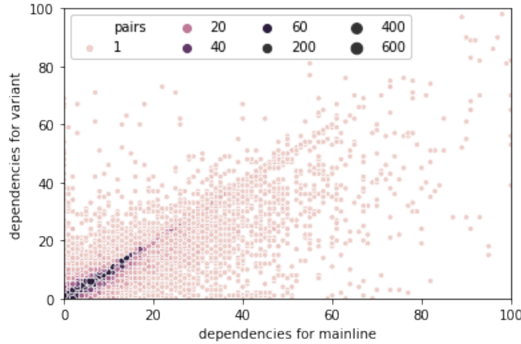

Fig. 4: The distribution of the mainline versus variant package dependencies.

RQ_2: **How do the distributions of package dependencies in mainlines and their variants compare to each other?**

Figure 4 presents a scatter plot showing the the package dependencies of the variant versus the dependencies of the mainline. The x-axis shows the number of dependencies of the mainline and the y-axis shows the number of dependencies of the variant. The explanation of the data points is the same as that of Figure 3. We observe some kind of correlation between the dependencies of the mainlines and variants. This means the more dependencies mainlines are associated with more dependencies of the variants. This could imply that the fork variant inherits the original dependencies of the mainline.

We observe a correlation between the number of mainline and variant dependencies. This could imply that the variant continues using the packages inherited from the mainline.

RQ_3: **Do the variant projects have dependent packages / projects?**

Figure 5 and 6 present a scatter plots for mainlines versus variants for the dependent packages and dependent projects, respectively. The x-axes represent the number of dependent packages / projects for the mainlines and the y-axes represent

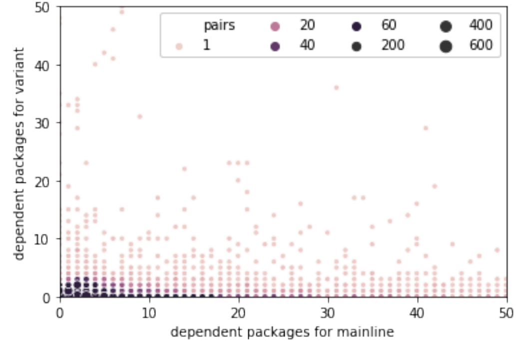

Fig. 5: The distribution of the dependent packages for mainline versus variant .

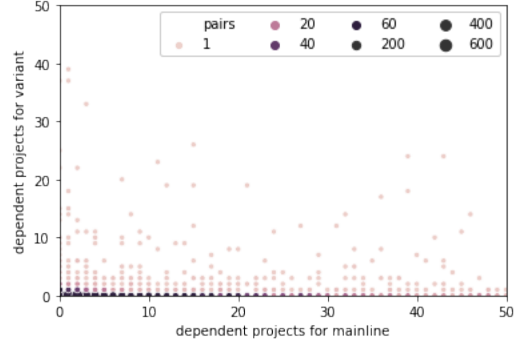

Fig. 6: The distribution of the dependent projects for the mainline versus variant.

the number of dependent packages / projects for the variants. The explanation of the data points is the same as that of Figure 3. Looking at both Figure 5 and 6, we observe that most of the data points are concentrated on the x-axis. This implies that most mainline have many dependent packages/projects compared to their variants counterparts. However, we observe a few scattered data points a long the y-axes indicating a few variants which have many dependent packages/projects compared to their mainline counterparts. In Table III we present variants that have more dependent packages compared to their mainline counterparts.

TABLE III: Example of variants that have more dependent packages compared to their mainline counterparts.

mainlines	variants	mainline dependent packages	variant dependent packages	diff
selenium	selenium-server	97	2046	1949
replace2	replace	0	1043	1043
grunt-mocha-screenshot	grunt-mocha	2	651	649
mocha-istanbul	grunt-mocha-istanbul	606	987	381

Compared to the mainline counterparts, the variants have fewer dependent packages/projects. Since it is plausible to assume that the mainline and variants offer similar functionality because of the common code base, it is still interesting to observe that other projects use the variant

package releases.

V. Conclusion

We performed an exploratory investigation on the evolution of variants focusing on their technical aspects. We mined repositories from the JavaScript ecosystem, whose sources are hosted on GitHub, having their package releases on npm. We identified a significant number of variants from the JavaScript ecosystem. We observed that in general mainlines have more package releases, package dependencies, dependent packages and dependent projects compared to their variant counterparts. However, we observe a a considerable number of variants having many package releases, package dependencies and dependent packages/projects.

As future work, we plan to carry out a detailed investigation on the socio-technical specificities of the software families in the JavaScript ecosystem whose repositories are hosted on GitHub.

References

[1] M. Lungu, "Towards reverse engineering software ecosystems," in *2008 IEEE International Conference on Software Maintenance*, 2008, pp. 428–431.

[2] A. Decan, T. Mens, and M. Claes, "An empirical comparison of dependency issues in oss packaging ecosystems," in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2017, pp. 2–12.

[3] I. J. Mojica, B. Adams, M. Nagappan, S. Dienst, T. Berger, and A. E. Hassan, "A large-scale empirical study on software reuse in mobile apps," *IEEE Software*, vol. 31, no. 2, pp. 78–86, 2014.

[4] S. Zhou, B. Vasilescu, and C. Kästner, "How has forking changed in the last 20 years? a study of hard forks on GitHub," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings*, ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 268–269. [Online]. Available: https://doi.org/10.1145/3377812.3390911

[5] T. Berger, J.-P. Steghöfer, T. Ziadi, J. Robin, and J. Martinez, "The state of adoption and the challenges of systematic variability management in industry," *Empirical Software Engineering*, 01 2019.

[6] L. Nyman, T. Mikkonen, J. Lindman, and M. Fougère, "Perspectives on code forking and sustainability in open source software," in *Open Source Systems: Long-Term Sustainability*, 2012, pp. 274–279.

[7] G. Robles and J. M. González-Barahona, "A comprehensive study of software forks: Dates, reasons and outcomes," in *Open Source Systems: Long-Term Sustainability*, 2012, pp. 1–14.

[8] R. Viseur, "Forks impacts and motivations in free and open source projects," *International Journal of Advanced Computer Science and Applications - IJACSA*, vol. 3, no. 2, 02 2012.

[9] L. Nyman and J. Lindman, "Code forking, governance, and sustainability in open source software," *Technology Innovation Management Review*, vol. 3, pp. 7–12, 01/2013 2013.

[10] A. S. Laurent, *Understanding Open Source and Free Software Licensing* . O'Reilly Media, 2008.

[11] L. Nyman and T. Mikkonen, "To fork or not to fork: Fork motivations in sourceforge projects," in *Open Source Systems: Grounding Research*, 2011, pp. 259–268.

[12] J. Businge, O. Moses, S. Nadi, E. Bainomugisha, and T. Berger, "Clone-based variability management in the Android ecosystem," in *34th IEEE International Conference on Software Maintenance and Evolution (ICSME), Industry Track*, 2018.

[13] B. B. Chua, "A survey paper on open source forking motivation reasons and challenges," in *21st Pacific Asia Conference on Information Systems, PACIS 2017, Langkawi, Malaysia, July 16-20, 2017*, R. A. Alias, P. S. Ling, S. Bahri, P. Finnegan, and C. L. Sia, Eds., 2017, p. 75.

[14] J. Dixion, ""different kinds of open source forks – salad, dinner, and fish"," https://jamesdixon.wordpress.com/2009/05/13/different-kinds-of-open-source-forks-salad-dinner-and-fish/, 2009.

[15] N. A. Ernst, S. M. Easterbrook, and J. Mylopoulos, "Code forking in open-source software: a requirements perspective," *ArXiv*, vol. abs/1004.2889, 2010.

[16] L. Nyman, "Hackers on forking," in *Proceedings of The International Symposium on Open Collaboration*, 2014, p. 1–10.

[17] E. S. Raymond, *The Cathedral & the Bazaar: Musings on linux and open source by an accidental revolutionary*. O'Reilly Media, Inc., 2001.

[18] J. Jiang, D. Lo, J. He, X. Xia, P. S. Kochhar, and L. Zhang, "Why and how developers fork what from whom in GitHub," *Empirical Softw. Engg.*, vol. 22, no. 1, p. 547–578, Feb. 2017. [Online]. Available: https://doi.org/10.1007/s10664-016-9436-6

[19] A. Decan, T. Mens, and M. Claes, "On the topology of package dependency networks: A comparison of three programming language ecosystems," in *Proccedings of the 10th European Conference on Software Architecture Workshops*, ser. ECSAW '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2993412.3003382

[20] J. Businge, S. Kawuma, E. Bainomugisha, F. Khomh, and E. Nabaasa, "Code authorship and fault-proneness of open-source Android applications: An empirical study," in *PROMISE*, 2017.

[21] J. Businge, M. Openja, D. Kavaler, E. Bainomugisha, F. Khomh, and V. Filkov, "Studying Android app popularity by cross-linking GitHub and google play store," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2019, pp. 287–297.