An Empirical Study of Technical Debt Management as a Motivation for Forking

1st Mercy Njima
Department of Computer Science
University of Antwerp
Antwerp, Belgium
mercy.njima@uantwerpen.be

2nd John Businge
Department of Computer Science
University of Antwerp
Antwerp, Belgium
john.businge@uantwerpen.be

3rd Serge Demeyer
Department of Computer Science
University of Antwerp and Flanders Make
Antwerp, Belgium
serge.demeyer@uantwerpen.be

***Abstract*—Forking is an often used idiom in software ecosystems where it allows for immediate reuse of existing software packages. Further, reuse and forking address software engineering goals such as long term software quality. However, there is a lack of sufficient knowledge exploring the validity and applicability of forking as an approach to solve software quality issues. In this position paper we present a plan to investigate whether forking is a useful avenue for managing technical debt.**

***Index Terms*—Forking, Software reuse, npm, SonarQube, Technical debt**

I. Introduction

Software product line approaches advocate strategic, planned reuse that yield predictable results. In practice though product variants often emerge ad-hoc, when companies have to release a new product that is similar, yet not identical, to existing ones [1]. To implement new product functionality, developers often fork an existing product and modify it to fit new requirements using the "clone-and own" approach [2]. Forking provides a rapid way to address new requirements by adapting an existing solution [3].

Open source software provides an existing code base that acts as a starting point for software developers to reuse and create a software variant by forking an existing project. Prior work has shown that code reuse can be beneficial in reducing the time-to-market, improving software quality and boosting overall productivity [4], [5]. Thus, platforms such as Node.js have emerged to encourage reuse and facilitate code sharing through packages or modules that are available on package management platforms, such as the Node Package Manager (npm) [6], [7].

First, developers may fork mainlines with a high technical debt so that they address that technical debt and send back the contributions to the mainlines (these kind of forks are called social forks [8]). However, previous research shows that some projects do not easily accept contributions into their repositories [9]. When the contribution is rejected, the fork developer may end up maintaining the fork and in the end it evolves into a variant of the mainline with variant specific code. In fact Zhou et al. [10] reports that many variant forks actually start as social forks.

In addition to the aforementioned motivations and benefits, we observe that reuse and forking may address other software engineering goals such as long term software quality. High quality code is code that incurs low maintenance costs and allows for the fast integration of new team members. One of the impediments to software quality is technical debt. "Technical debt refers to a collection of design or implementation constructs that are expedient in the short term, but set up a technical context that can make future changes more costly or impossible. It presents an actual or contingent liability whose impact is limited to internal system qualities, primarily maintainability and evolvability" [11].

II. Background

Researchers have collected a large body of knowledge on forking and the motivations for it. Robles et al. performed an in depth study on several hundred forks and reported on the date when the forking occurred, the reason of the fork, and the outcome of the fork for the original and forking project [12]. Nyman et al. report on the possible benefit of forking serving as an invisible hand in the long term sustainability of software projects and safeguarding against unfavorable decisions from a single developer or organization [13], [14]. Viseur reported on a detailed study of twenty six open source projects highlighting the motivations and impact of forking [15]. They found that the main motivations of forking are technical divergences, governance mismatches, end of the original project, license change, conflict about trademark and strong cultural differences. Businge et al. performed an exploratory study on clone-based reuse practices for open-source Android apps [16]. They found that the motivations for the fork variants were re-branding and simple customisation, feature extensions, supporting of the mainline and development of different, but related features. Jing et al. explore why and how developers fork what from whom in GitHub [17]. From their study they found that the reasons developers fork projects are to submit pull requests, add new features, fix bugs and keep copies of the original repository.

To the best of our knowledge, this is the first study to examine technical debt management as a motivation for forking. However, Ernst et al., addressed the question of whether requirements were a basis for a fork and they hypothesized that forking was required to address the soft goals of maintainability and usability [3]. They confirmed that indeed the fork had a better code base and also satisfied the soft goals of usability and maintainability.

We hypothesise that mainlines with higher values of technical debt are likely to be associated with higher numbers of variant forks. Thus, this paper proposes a study to investigate whether forking is a potentially useful avenue for managing the technical debt issues of maintainability and reliability.

III. Empirical Study Design

We follow a mixed method approach to examine the relationship between forking and technical debt. First, we mine and analyse mainline forks from npm and later perform a confirmatory analysis using a survey of the maintainers of the forked projects.

A. Goal and Research Questions

The goal of the study is to investigate whether forking and the creation of forked product variants are intended approaches in managing technical debt. To this end, we plan to answer the following research questions.

- RQ1: Is there a relationship between the amount of forking and the amount of technical debt?
- RQ2: How do open-source contributors perceive forking as a way to manage technical debt?

B. Study Setup and Data Collection

To perform our study, we obtained a dataset of Node.js packages from the npm registry on which we perform technical debt analysis using the community edition of SonarQube. Fig 1 shows our intended workflow of the study.

Since its inception, npm has grown to become one of the largest software ecosystems [18], [19]. We chose npm for the following reasons: it provides API access to all package releases and metadata, most npm packages point to a GitHub repository and the npm registry and GitHub both show the package's README file, providing a common place where more information is displayed. Moreover, the npm community is innovation friendly and broadly experiments with and adopts developer services including cloud-based continuous integration [20], [21].

SonarQube claims to be one of the leading tools for continuously inspecting code quality and security and guiding development teams in code reviews [22]. SonarQube calculates several metrics such as: lines of code, complexity, coverage, false positive issues, code smells and vulnerabilities. The analysis is violation-based and examines the health of the code according to a set of rules. If the code violates these coding rules, SonarQube reports this as an 'issue'. There are three main issue domains in SonarQube [23]:

- Maintainability: Commonly referred to as technical debt, maintainability issues are reported as 'code smells' which may need to be addressed in the future.
- Reliability: Referred to as bugs in the code, reliability issues are critical programming errors that may be thrown at run-time.

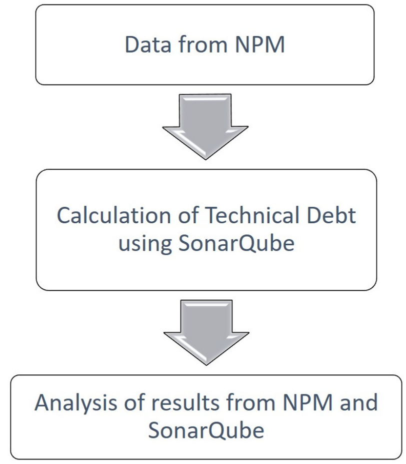

Fig. 1: Empirical study workflow.

- Security: referred to as vulnerabilities, are flaws in programs that can lead to misuse and exploitation of the application.

For the purpose of our work, we are interested in studying the following two concepts as they relate to technical debt and the evolvability of the packages we will analyse from npm.

- Code smells which are a maintainability issue that makes the code difficult to maintain in the long run and increase the overall technical debt.
- Bugs: issues that throw an error during run-time should be fixed as soon as possible.

RQ1. To address the first research question, we collected a data set of npm packages by mining all npm packages then kept only those that had at least two forks, contained metadata on dependencies, dependents, and maintainers and had a link to a GitHub repository. We are currently implementing a SonarQube scanner pipeline to analyse our dataset and provide us with the technical debt measurements we require to test our hypothesis. The results from the SonarQube analysis will be collected in an Excel file. We will apply open coding to these results to explore the relationship between forked packages and the amount of technical debt.

RQ2. To answer research question two, and gauge developer perceptions we will perform an online survey targeting npm maintainers and contributors of the most active packages. We have a cut off period of 100 days of updates given that developers may clearly recall their reasoning behind intent of code, data modified in code, owners of files, files that rarely/often changed, recent changes etc within that time period following work done by Kruger et al. [24].

IV. Expected Results

As we report on our findings about the intersection between forking, variants and technical debt, this work will be useful in generating knowledge about the problem where the literature does not provide much insight. We will report on the amounts and types of technical debts contained in the npm packages and

Github repositories we study. In addition, we will motivate the need for more studies on the nature of requirements in forking and whether requirements are a justification for forking.

ACKNOWLEDGMENT

This work is supported by Flanders Make vzw, the strategic research centre for the manufacturing industry.

REFERENCES

[1] Y. Dubinsky, J. Rubin, T. Berger, S. Duszynski, M. Becker, and K. Czarnecki, "An exploratory study of cloning in industrial software product lines," in *2013 17th European Conference on Software Maintenance and Reengineering*, 2013, pp. 25–34.

[2] J. Rubin, A. Kirshin, G. Botterweck, and M. Chechik, "Managing forked product variants," in *SPLC '12*, 2012.

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

[4] W. C. Lim, "Effects of reuse on quality, productivity, and economics," *IEEE Software*, vol. 11, no. 5, pp. 23–30, 1994.

[5] P. Mohagheghi, R. Conradi, O. M. Killi, and H. Schwarz, "An empirical study of software reuse vs. defect-density and stability," in *Proceedings of the 26th International Conference on Software Engineering*, ser. ICSE '04. USA: IEEE Computer Society, 2004, p. 282–292.

[6] R. Abdalkareem, O. Nourry, S. Wehaibi, S. Mujahid, and E. Shihab, "Why do developers use trivial packages? an empirical case study on npm," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017. New York, NY, USA: Association for Computing Machinery, 2017, p. 385–395. [Online]. Available: https://doi.org/10.1145/3106237.3106267

[7] "Npm docs," https://docs.npmjs.com/about-npm, accessed October 2020.

[8] K. H. Fung, A. Aurum, and D. Tang, "Social forking in open source software: An empirical study," in *CAiSE Forum*, 2012.

[9] S. Zhou, S. Stanciulescu, O. Leßenich, Y. Xiong, A. Wasowski, and C. Kästner, "Identifying features in forks," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, 2018, pp. 105–116.

[10] 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

[11] P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman, "Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162)," *Dagstuhl Reports*, vol. 6, no. 4, pp. 110–138, 2016. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2016/6693

[12] G. Robles and J. M. Gonzalez-Barahona, "A comprehensive study of software forks: Dates, reasons and outcomes," in *OSS*, 2012.

[13] L. Nyman, T. Mikkonen, J. Lindman, and M. Fougere, "Perspectives on code forking and sustainability in open source software," in *8th IFIP WG 2.13 International Conference, OSS 2012, Hammamet, Tunisia, September 10-13, 2012. IFIP Advances in Information and Communication*, ser. IFIP Advances in Information and Communication. Springer, 2012, pp. 274–279, ei UT-numeroa 27.8.2013
Contribution: organisation=ohj,FACT1=1
Publisher name: Springer.

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

[15] R. Viseur, "Forks impacts and motivations in free and open source projects," *International Journal of Advanced Computer Science and Applications*, vol. 3, 2012.

[16] J. Businge, M. Openja, S. Nadi, E. Bainomugisha, and T. Berger, "Clone-based variability management in the android ecosystem," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. Los Alamitos, CA, USA: IEEE Computer Society, sep 2018, pp. 625–634. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICSME.2018.00072

[17] 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

[18] E. Wittern, P. Suter, and S. Rajagopalan, "A look at the dynamics of the javascript package ecosystem," in *Proceedings of the 13th International Conference on Mining Software Repositories*, ser. MSR '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 351–361. [Online]. Available: https://doi.org/10.1145/2901739.2901743

[19] C. Bogart, C. Kästner, J. Herbsleb, and F. Thung, "How to break an api: Cost negotiation and community values in three software ecosystems," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 109–120. [Online]. Available: https://doi.org/10.1145/2950290.2950325

[20] A. Trockman, S. Zhou, C. Kästner, and B. Vasilescu, "Adding sparkle to social coding: An empirical study of repository badges in the npm ecosystem," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, 2018, pp. 511–522.

[21] S. Mirhosseini and C. Parnin, "Can automated pull requests encourage software developers to upgrade out-of-date dependencies?" in *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2017. IEEE Press, 2017, p. 84–94.

[22] "About sonarqube," https://www.sonarqube.org/about, accessed October 2020.

[23] "Code quality - sonarqube," https://www.sonarsource.com/why-us/code-quality/, accessed October 2020.

[24] J. Krüger, "What developers (care to) recall: An interview survey on smaller systems," 2020.