Predictive Modelling of Socio-Technical Health in Evolving Software Packaging Ecosystems

Pooya Rostami Mazrae[1]

[1]University of Mons, Mons, Belgium

Abstract

Software health plays an important role in collaborative software development. My PhD research aims to analyse how the evolution of socio-technical characteristics in large open source software ecosystems affects the health of these ecosystems and their building blocks. In order to capture as many different dimensions of software health, I aim to combine the human (social) and technical aspects of collaborative software development activity. These dimensions will be integrated into computational machine learning models and recommendation models to enable the prediction of change trends in software health, and to improve future health based on historical analysis. I will focus primarily on the health of evolving software packaging ecosystems, as they are known to have large technical dependency networks, as well as strongly connected social collaboration networks. This extended abstract presents the research questions I will explore to reach the aforementioned research goals.

Keywords

Mining Software Repository, Software Health, Open Source Software, Projects Abandonment, Software Ecosystem, Predictive Modeling, Machine Learning

1. Introduction

The importance of open source software (OSS) development has increased significantly throughout the last years, covering almost every application domain [1]. Today, over 80% of the software in technological products or services is OSS, and this trend is still growing[1]. In addition to this, software ecosystems play an ever increasing role in collaborative software development practices. A software ecosystem can be defined as a collection of software projects which are developed and which co-evolve together in the same environment [2]. As software projects are not usually developed in isolation, it is important to take into account the ecosystem of which they are part to understand the bigger picture.

Software package distributions can be considered as a specific kind of software ecosystem. Nearly every popular programming language is accompanied by one or more package managers. These package managers can be used by software developers to easily install reusable software libraries. These so-called software packaging ecosystems contain of large number of package

BENEVOL'21: The 20th Belgium-Netherlands Software Evolution Workshop, December 07–08, 2021, 's-Hertogenbosch (virtual), NL

pooya.rostamimazrae@umons.ac.be (P. R. Mazrae)
https://pooya-rostami.github.io/ (P. R. Mazrae)
0000-0002-4859-1546 (P. R. Mazrae)

[1]https://www.linuxfoundation.org/blog/chaoss-project-creates-tools-to-analyze-software-development-and-measure-open-source-comm

releases, accompanied by their distributed version repositories. These repositories are updated regularly and have many technical and social interdependencies, forming huge socio-technical package dependency networks.

The challenges related to the evolution of software packaging ecosystems can be seen from two dimensions: the *social* dimension that focuses on problems related to persons who are contributing to, and interacting with (parts of) the ecosystem, and the *technical* dimension that addresses problems related to the technical artefacts (such as the source code, tests, documentation, or any other artefacts) being produced or maintained. Given that both dimensions cannot be seen in isolation, *socio-technical* challenges will involve a combination of technical and social issues.

Health issues in the social dimension are manifold. For example, the so-called *truck factor* (TF) aims to measure the risk of a project to stop being maintained because too many of its core developers are abandoning ("run over by a truck") [3, 4, 5], as well as the risk of having *heroes* who are the only core developers who understand and know certain critical parts of a system [6]. Knowing the reasons why developers are leaving [7] or taking a break [4] might help in mitigating these risks, and finding good replacements for abandoners might further reduce these risks [5].

Health issues in the technical dimension have to do with how packages within an ecosystem are interrelated through transitive dependencies that may not be updated when new package releases become available, thereby affecting ecosystem health [8]. As an example, to alleviate the problem of *dependency hell*, approaches like semantic versioning or using dependency graphs [9] have been proposed. Nevertheless, the degree of adherence to semantic versioning can differ significantly depending on the considered package distribution [10].

An important requirement for conducting empirical studies in this domain is having access to a data source containing recent, reliable and sufficiently complete information about large software ecosystems that contain evidence of socio-technical interaction and collaboration patterns between their components (including contributors, projects and packages).

Many studies gather historical project data from GitHub considering mainly popularity metrics (e.g., number of stars or number of downloads) [6, 11, 3, 5, 7, 12, 1, 13]. While this can be one of the factors to select projects for analysis, other factors need to be considered as well, especially if not all relevant socio-technical information is recorded on GitHub. For example, Avelino et al. found examples of contributions to the Linux kernel in which the entire release development was pushed to Git in a single squashed commit, thereby masking many individual contributions on behalf of a unique developer [14].

Many empirical studies also concentrate on only one dimension of software health. To study both the social and technical dimension, we will concentrate on software packaging ecosystems. They contain metadata about the technical interdependencies between all packages. Also, the full development history of these packages is available, from which the social collaboration and interaction between contributors of those packages can be retrieved. By accompanying these social and technical aspects in a single overarching socio-technical dependency network, I will study the evolution of health problems at a new level and propose new ways to improve the health of software packaging ecosystems. Works on this matter has already started. For example, the Linux Foundation working group *CHAOSS* is creating metrics and associated tools to help define and measure OSS project and community health [15].

My PhD research therefore aims to empirically study and reduce socio-technical health issues in evolving OSS packaging ecosystems, by determining the important features of, and connections between, different packages that play an important role in health issues. Based on this, I aim to provide recommendation models and prediction models to reduce these health issues. Overall, my proposed thesis statement is: *Machine learning techniques for predictive modeling of socio-technical aspects can be used to gain information about, and improve the health of, evolving software ecosystems.* To validate this statement, I will explore the following research questions:

- **RQ1**: *How to attract newcomers to projects within a software packaging ecosystem?*
- **RQ2**: *How does the socio-technical activity and abandonment of project contributors affect project and ecosystem health?*
- **RQ3**: *How to predict or recommend replacement of abandoning developers by analysing the socio-technical dependency network?*
- **RQ4**: *How to rely on social media activity (e.g., Reddit, StackOverflow, Twitter) to improve prediction models?*

2. Background

OSS research has studied a wide range of aspects, often focusing on only the technical or the social dimension of software development. Research that combines the social and technical aspects often leads to more enlightening results. Below, we discuss some of these works and how they contribute to the domain of OSS health research.

Conway's law states that organizations tend to design systems that mimic the communication structures of these organizations. Cataldo et al. [16] introduced the notion of socio-technical congruence to reflect the close connection between the technical structure of a software project and the social structure of the project members. Syeed et al. [17] studied such socio-technical congruence in the context of the Ruby packaging ecosystem. Golzadeh [18] provided an initial exploration of the socio-technical congruence in the Cargo packaging ecosystem. We hypothesise that any socio-technical study of software ecosystem health is likely to be affected by the socio-technical congruence phenomenon.

Ricca et al. [6] tried to find the heroes in FLOSS projects, by implementing a tool to compute the truck factors and identify the heroes. The proposed tool was based on the work of Zazworka et al. [19]. Since finding a truck factor plays an important role in software health, Ferreira et al. [3] compared 3 different algorithms for computing it: AVL [20], RIG [21] and CST [22]. Based on an evaluation on 35 open source projects they concluded that AVL is the most accurate in predicting the truck factor and predicting developers responsible for that truck factor. In an extension of their study they found that a reason for poor predictions is by not taking into account social interactions such as code review, documentation, tests and supporting tools [12].

Researchers have also studied the reason for project failure. Coelho et al. [11] studied 5,000 GitHub repositories with the intent of finding the maintenance challenges. They identified nine reasons why open source projects fail. In descending order of happening they are *usurped by competitors*, *obsolescence*, *lack of time*, *lack of interest*, *outdated technologies*, *low maintainability*, *conflict among developers*, *legal problems* and *acquisition.* They also proposed a list of important

open source maintenance practice, including: the presence of a README file; the presence of a separate project license file; the availability of a dedicated website to promote the project, including examples and documentation; the use of a CI service; the presence of a specific file with guidelines for repository contributors; the presence of an issue template and a pull request template.

In a study based on an analysis of 9,977 open-source npm libraries, Qiu et al [23] showed that, for contributors that want to engage in a new OSS project, features like *GitHub stars*, *recent commits*, *comprehensive README files* and *having issue or pull request templates* play an important role. While projects with a higher number of stars will attract more first-time GitHub contributors, the presence of contributing guidelines has a significantly negative effect mostly because it makes newcomers uncomfortable to join the work.

Multiple OSS development studies aimed to determine why contributors disengage from open source. This is important to know because around 80% of OSS project failures is related to issues with contributor turnover [24]. Miller et al. [7] conducted a study to determine the reasons why people give up working on OSS projects. They considered three types of reasons: *occupational*, *social* and *technical*. The *occupational reasons*, in descending order of importance, are: having a new job that doesn't support OSS; change of role/project; left job where they contributed to OSS; no time because of new job; no time because of existing job; used OSS in school but new job doesn't support OSS; too much code at work. The *social reasons*, in descending order of importance, are: loss of interest; no time due to personal reasons; lack of peer support; no time (unspecified reason). The *technical reasons*, in descending order of importance, are: issues with GitHub or industry; individually moved to private repertories; changed platform; feature complete project. A possible extension to this work could be to study whether the same reasons apply for collaborating on packages in a software packaging ecosystem.

To determine the state of OSS project developers, Iaffaldano et al. [4] considered three states: *Alive*, *Sleep* and *Dead*. Based on interview with different developers they determined that *sleeping developers* are those who do not contribute code but still show interest in the project in other ways such as answering emails and participating in discussions. In contrast, *dead developers* are those who not only have stopped providing code contributions for some time, but also do not participate in any other community activity. Calefato et al. [13] further studied this matter and observed that breaks are rather common, in that core members take frequent breaks or varying length and type. They observed that all developers took at least one break, 97% of them transitioned to non-coding and 89% to inactive. They also analyzed the probability of the transitions to/from the inactivity state and observed that core developers are more likely to remain in the projects. However, if they transition to gone, they are less likely to come back (54% probability on average). Extending this study to software package ecosystems could be useful to understand the migration of developers between different software packages, as this may constitute a valid reason for contributors to become inactive or change their state in a given software package.

Avelino et al. [5] studied abandonment and survival of OSS projects. They concentrated on the truck factor developer detachments (TFDDs) and the replacements of such abandoning developers. They found that 59% of the TFDDs happened in the first two years of project development; but 71% of the projects with TFDDs have now between 4 and 7 years of development. They reported that recovering from a TFDD is not uncommon in that about 41% of the

projects survive. In 86% of these cases they do so by replacing the TF developer by a single new contributor. This shows that there are many developer migrations between different projects of the same ecosystems which can be a good source of study for socio-technical health information like the relation of developers with each other and with the technical aspect of projects. The information extracted from these migrations can be later used for our predictive model for health study.

Another aspect of software health is predicting the project maintenance activity, in order to determine whether the project is going to be deprecated. Coelho et al. [1] gathered a dataset of 6,785 most starred GitHub projects with more than two years of software development data available. They created a machine learning classifier based on the Random Forest algorithm combining social and technical project data. After training the model achieved 86% precision in predicting project abandonment. The results were confirmed with 129 developers. The top features of the model were *commits*, *max days without commits* in months 22 to 24 of the project, *max days without commits* in months 10 to 12, *max contributions by developer* in months 16 to 18, and *closed issues* in months 1 to 3 of the last two years of the project. Being able to predict the project abandonment can help us predict the overall health of the encompassing software packaging ecosystem. At the ecosystem level, this will help maintainers to predict possible health issues in the projects they depend upon, or even signal them that their own project is at the edge of becoming deprecated, and prepare them for the future.

Decan et al. [25] proposed a probabilistic model and associated tool (called GAP) to predict the risk of contributor abandonment, based on the previous commit activities of these contributors. The model was evaluated on GitHub repositories corresponding to development activity for reusable software packages distributed through the Cargo package manager for the Rust programming language. Studying migration patterns of developers within the ecosystem is one of our goals and having a tool that predicts when a developer is going to stop contributing to a project can help to signal and predict future developer migration patterns.

3. Data Gathering

This section presents the data gathering process to extract and prepare the socio-technical data from evolving packaging ecosystems that will be required to respond to the *Research Questions*. The first phase will consist of gathering all data required to construct the socio-technical package dependency networks. This will be achieved by retrieving metadata through the APIs provided by the package managers of the corresponding packaging ecosystems. Each package manager (e.g. Cargo for Rust crates, npm for Node.JS packages, and PyPI for Python packages) has its own dedicated package registry and associated API. Examples of project-specific information that can be retrieved in this way includes the project's homepage, repository link, owners and maintainers, project classifier or category, project dependencies, version numbers and release dates. To address **RQ4** we will also gather the data related to the social media channels in which the project or ecosystem contributors are involved (e.g., Reddit, StackOverflow, Twitter, LinkedIn), through the dedicated APIs of these social media.

The second phase of the extraction process consists of retrieving more specific socio-technical information from the development repositories linked to each package. These repositories

are typically hosted on social coding platforms such as GitHub. We will use the API of the corresponding social coding platform to retrieve relevant information such as issues, commits, pull requests, code reviews, comments, tags and releases, number of stars, forks, downloads, and so on. Another way to gather this information is by using archives and research platforms such as GHTorrent [2], Software Heritage [3], and World of Code [4]. The main limitations of such platforms are that they do not necessarily contain a complete and up to date archive of the data of interest.

The third phase will involve cleaning and transforming the collected socio-technical data into a uniform dependency network structure that will enable to study socio-technical issues in an ecosystem-agnostic way. We will explore different graph structures to study these socio-technical networks of projects and project contributors.

4. Research plan

Considering the fact that my PhD research project started in November 2021, I am actively exploring the research domain of socio-technical health of evolving software packaging ecosystems. During my research, I aim to gain a deeper understanding about the different health issues in software packaging ecosystems and leverage this understanding to develop models to predict the health of these ecosystems and recommend improvements to them. These models will be validated following a mixed-methods research approach, combining quantitative analysis based on software repository mining data with qualitative analysis based on surveys and interviews with ecosystem contributors.

I will start by concentrating on **RQ1** by using project-specific socio-technical information to determine the important factors that attract newcomers to participate in OSS projects. Later, By using machine learning algorithms, I will develop prediction models of the overall attractivity of projects, and provide insights about the most important features that make the specific ecosystem attractive to contributors.

The next phase will focus on **RQ2** and **RQ3**. The main purpose of **RQ2** is to study the social interaction between project contributors during technical activities and its effect on the community that is working on the project. For **RQ3**, we will explore different graph structures of the socio-technical networks of the ecosystem, in order to predict abandoning developers and recommend replacements for them.

The third phase will focus on **RQ4**, by studying social media activities of ecosystem contributors and use such information as a new data source for improving software heatlth prediction and recommendation models.

Acknowledgments

This work is supported by Action de Recherche Concertée ARC-21/25 UMONS3 financée par le Ministère de la Communauté française – Direction générale de l'Enseignement non obligatoire

[2] https://ghtorrent.org/
[3] https://www.softwareheritage.org/
[4] https://worldofcode.org/

et de la Recherche scientifique.

References

[1] J. Coelho, M. T. Valente, L. Milen, L. L. Silva, Is this GitHub project maintained? Measuring the level of maintenance activity of open-source projects, Information and Software Technology 122 (2020). doi:10.1016/j.infsof.2020.106274.

[2] K. Blincoe, F. Harrison, D. Damian, Ecosystems in GitHub and a method for ecosystem identification using reference coupling, in: Working Conference on Mining Software Repositories, IEEE, 2015, pp. 202–211. doi:10.1109/MSR.2015.26.

[3] M. Ferreira, M. T. Valente, K. Ferreira, A comparison of three algorithms for computing truck factors, in: International Conference on Program Comprehension (ICPC), IEEE, 2017, pp. 207–217. doi:10.1109/ICPC.2017.35.

[4] G. Iaffaldano, I. Steinmacher, F. Calefato, M. A. Gerosa, F. Lanubile, Why do developers take breaks from contributing to OSS projects? a preliminary analysis, in: 2nd International Workshop on Software Health, 2019, pp. 9–16. doi:https://dl.acm.org/doi/10.1109/SoHeal.2019.00009.

[5] G. Avelino, E. Constantinou, M. T. Valente, A. Serebrenik, On the abandonment and survival of open source projects: An empirical investigation, in: International Symposium on Empirical Software Engineering and Measurement (ESEM), IEEE, 2019, pp. 1–12. doi:10.1109/ESEM.2019.8870181.

[6] F. Ricca, A. Marchetto, Are heroes common in floss projects?, in: ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, 2010, pp. 1–4. doi:10.1145/1852786.1852856.

[7] C. Miller, D. G. Widder, C. Kästner, B. Vasilescu, Why do people give up flossing? a study of contributor disengagement in open source, in: IFIP International Conference on Open Source Systems, Springer, 2019, pp. 116–129. doi:10.1007/978-3-030-20883-7_11.

[8] A. Decan, T. Mens, P. Grosjean, An empirical comparison of dependency network evolution in seven software packaging ecosystems, Empirical Software Engineering 24 (2019) 381–416. doi:10.1007/s10664-017-9589-y.

[9] G. Fan, C. Wang, R. Wu, X. Xiao, Q. Shi, C. Zhang, Escaping dependency hell: finding build dependency errors with the unified dependency graph, in: International Symposium on Software Testing and Analysis, 2020, pp. 463–474. doi:10.1145/3395363.3397388.

[10] A. Decan, T. Mens, What do package dependencies tell us about semantic versioning?, IEEE Transactions on Software Engineering (2019). doi:10.1109/TSE.2019.2918315.

[11] J. Coelho, M. T. Valente, Why modern open source projects fail, in: Joint meeting on foundations of software engineering, 2017, pp. 186–196. doi:10.1145/3106237.3106246.

[12] M. Ferreira, T. Mombach, M. T. Valente, K. Ferreira, Algorithms for estimating truck factors: a comparative study, Software Quality Journal 27 (2019) 1583–1617. doi:10.1007/s11219-019-09457-2.

[13] F. Calefato, M. A. Gerosa, G. Iaffaldano, F. Lanubile, I. Steinmacher, Will you come back to contribute? Investigating the inactivity of OSS core developers in GitHub, arXiv preprint arXiv:2103.04656 (2021). URL: https://arxiv.org/abs/2103.04656.

[14] G. Avelino, L. Passos, A. Hora, M. T. Valente, Measuring and analyzing code authorship in 1+ 118 open source projects, Science of Computer Programming 176 (2019) 14–32. doi:10.1016/j.scico.2019.03.001.

[15] S. Goggins, K. Lumbard, M. Germonprez, Open source community health: Analytical metrics and their corresponding narratives, in: 2021 IEEE/ACM 4th International Workshop on Software Health in Projects, Ecosystems and Communities (SoHeal), IEEE, 2021, pp. 25–33.

[16] M. Cataldo, J. D. Herbsleb, K. M. Carley, Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity, in: ACM-IEEE international symposium on Empirical software engineering and measurement, 2008, pp. 2–11.

[17] M. M. Syeed, K. M. Hansen, I. Hammouda, K. Manikas, Socio-technical congruence in the ruby ecosystem, in: International Symposium on Open Collaboration, 2014, pp. 1–9.

[18] M. Golzadeh, Analysing socio-technical congruence in the package dependency network of Cargo, in: Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2019, pp. 1226–1228.

[19] N. Zazworka, K. Stapel, E. Knauss, F. Shull, V. R. Basili, K. Schneider, Are developers complying with the process: an xp study, in: International Symposium on Empirical Software Engineering and Measurement, 2010, pp. 1–10. doi:10.1145/1852786.1852805.

[20] G. Avelino, L. Passos, A. Hora, M. T. Valente, A novel approach for estimating truck factors, in: International Conference on Program Comprehension, IEEE, 2016, pp. 1–10. doi:10.1109/ICPC.2016.7503718.

[21] P. C. Rigby, Y. C. Zhu, S. M. Donadelli, A. Mockus, Quantifying and mitigating turnover-induced knowledge loss: case studies of Chrome and a project at Avaya, in: International Conference on Software Engineering, IEEE, 2016, pp. 1006–1016. doi:10.1145/2884781.2884851.

[22] V. Cosentino, J. L. C. Izquierdo, J. Cabot, Assessing the bus factor of git repositories, in: International Conference on Software Analysis, Evolution, and Reengineering, IEEE, 2015, pp. 499–503. doi:10.1109/SANER.2015.7081864.

[23] H. S. Qiu, Y. L. Li, S. Padala, A. Sarma, B. Vasilescu, The signals that potential contributors look for when choosing open-source projects, Proceedings of the ACM on Human-Computer Interaction 3 (2019) 1–29. doi:10.1145/3359224.

[24] A. Schilling, S. Laumer, T. Weitzel, Who will remain? an evaluation of actual person-job and person-team fit to predict developer retention in floss projects, in: Hawaii International Conference on System Sciences, IEEE, 2012, pp. 3446–3455. doi:10.1109/HICSS.2012.644.

[25] A. Decan, E. Constantinou, T. Mens, H. Rocha, Gap: Forecasting commit activity in git projects, Journal of Systems and Software 165 (2020) 110573. doi:10.1016/j.jss.2020.110573.