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

Pooya Rostami Mazrae
Software Engineering Lab, University of Mons, Belgium
pooya.rostamimazrae@umons.ac.be

***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 both the human (social) and the technical aspects of collaborative software development activity. These dimensions will be integrated into computational machine learning models to allow us to predict change trends in software health, as well as recommendation models to improve future health, based on historical analysis.**

I will focus primarily on evolving software packaging ecosystems to study their health, as they are known to have large technical dependency networks, as well as strong social collaboration networks. In this extended abstract, I present the research questions I am to explore on the health of such packaging ecosystems.

***Index Terms*—Mining Software Repository, Software Health, Open Source Software, Projects Abandonment, Software Ecosystem, Predictive Modeling, Machine Learning**

I. Introduction

The importance of the open source development has increased significantly throughout the last years, covering almost every kind of application domain [1]. Today, over 80% of the software in technological products or services is open source software (OSS), and this trend is still growing[1]. In addition to OSS, 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. Accordingly, software ecosystem analysis has increased in importance in recent years.

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 of reusable software libraries. These so-called software packaging ecosystems contain of large number of package releases that are updated regularly and that have many technical interdependencies, forming huge package dependency networks [3].

The challenges related to the evolution of software ecosystems can be divided in two separate 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") [4]–[6], the risk of having *heroes* who are the only core developers who understand and know certain critical parts of a system [7]. Knowing the reasons why developers are leaving [8] or taking a break [5] might help in mitigating this risk, and finding good replacements for abandoners might further reduce the risk [6].

Health issues in the technical dimension can be related to 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 [3]. 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. Many studies (e.g., [1], [4], [6]–[8], [11]–[13]) gather historical project data from GitHub considering mainly popularity metrics (e.g., number of stars or number of downloads). While this can be one of the factors to select projects, other factors need to be considered as well, especially if not all desired 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].

* 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 et de la Recherche scientifique.

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

Many empirical studies also concentrate on only one dimension of software health. To be able to study both the social and technical dimension, we propose to 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, we will be able to study the evolution of health problems at a new level and hopefully create new ways to improve the health of software packaging ecosystems. Works on this matter has already started. As an example, the Goggins et al. [15] has checked the tool named *CHAOSS* which is the work of a Linux Foundation working group to create metrics to help define community health.

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.

To start my research, I will explore the following research questions:

- **RQ1**: *How to improve the overall performance of truck factor algorithms? How to predict or recommend replacement of truck factor developers by analysing the socio-technical package dependency network?*
- **RQ2**: *How do project contributors migrate within a software ecosystem? How do ecosystems attract, retain or loose contributors? What is the impact of such phenomena on the ecosystem health?*
- **RQ3**: *How does the socio-technical behaviour of project contributors affect project and ecosystem health?*
- **RQ4**: *Can we rely on social media activity (e.g., Reddit, StackOverflow, Twitter) to increase the likelihood of finding new contributors for a given project in the ecosystem?*

II. Background

OSS research has studied a wide range of aspects, often focusing on only the technical or the social dimension of software development. Below, we cover some of these works and how they contribute to the domain of OSS health research.

The well-known Conway's law states that organizations which design systems are constrained to produce 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 ecosystem. Golzadeh [18] provided an initial exploration of the socio-technical congruence in Cargo. Any socio-technical study of software ecosystems is likely to be affected by the socio-technical congruence phenomenon.

Ricca et al. [7] 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 methods presented by Zazworka et al. [19]. Since finding a truck factor plays an important role in software health, Ferreira et al. [4] 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 prediction the truck factor and predicting developers responsible for that truck factor. In [12] they extended their study and found that a reason for poor predictions of truck factor is by not considering social interaction like code review, documentation, tests and supporting tools. This motivates the importance of including the social dimension in health prediction studies.

Another series of health-related studies aim to find the reason for project failures. Coelho et al. [11] studied 5,000 GitHub repositories with the intent of finding the maintenance challenges. They provided 9 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.

Another series of OSS development studies aims to determine why contributors disengage in open source. This is important to know because around 80% of open source projects failure is related to issues with contributor turnover [24]. Miller et al. [8] conducted a study to determine the reasons why people give up working on OSS projects. They considered 3 categories 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. [5] 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. [6] 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 also 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. They also studied the fact that if the new TF developers were aware of risk of surviving system and their reasons behind they contribute. 77% of the new TF were partially aware the risk and their reason for their contributes are: due to using the project; to contribute to an open source project; to avoid the project discontinuation; having interest in project area; getting paid to contribute; to improve their own skill; having the skills required by this project; being a successful project. This study shows us that in the first 2 years of the project development the chance of losing a truck factor developer is 59%. This can give us an idea that losing the developers in different ages of software development can vary and we probably should consider the age variable in different aspects of our study in the software ecosystem more than before.

Another aspect of software health is predicting the maintenance activity of a project, in order to determine whether the project is going to be deprecated or not. Coelho et al. [1] gathered a dataset of 6,785 most starred GitHub projects with more than 2 years of source code data available (and hence corresponding to actual software development projects). They created a machine learning classifier based on the Random Forest algorithm and used combination of project data, contributors and owners for input. After training the model, they achieved 86% precision in predicting if a project is going to be abandoned or not. They confirmed the result with 129 developers and reached a precision of 80%. The top 5 features of their 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 last two years of the project. Being able to predict the abandonment of a project can help us predict the overall health of an evolving software packaging ecosystem. At the level of the ecosystem, this will help maintainers to predict the probable problem with the project their own project depends on or even show them that their own project is at the edge of failure 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.

III. Data Gathering

This section presents the data gathering process that will be used to answer our *Research Questions*.

To be able to work on truck factor, migration of the contributors and studying the socio-technical behaviour of project contributors affect on the project and ecosystem health, we need to gather all the related data to the social and technical of projects. Also, since this studies are related to the ecosystems we need to focus on package managers.

The first phase will consist of gathering all relevant data about the socio-technical package dependency networks of packaging ecosystems. This can be achieved by retrieving package metadata through the API provided by the package manager of the corresponding package distribution. 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.

The second phase of the data extraction consists of retrieving more specific socio-technical information from the development repositories linked to each package in the packaging ecosystem, that are typically hosted on some social coding platform (e.g., GitHub). To do so, we will use the API of the

corresponding hosting platform to retriveve relevant information such as all issues, commits, pull requests, comments, tags and releases, number of stars, forks, downloads and so on.

The third phase will involve data 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-independent way.

To address **RQ4**, in addition to the socio-technical data gathered about the ecosystem and the projects contained in it, we aim to gather the data related to the social media in which the project or ecosystem contributors are involved, through there dedicated APIs.

IV. Research plan

Considering the fact that my PhD research project has just started, I am still actively exploring the research domain socio-technical health of evolving software packaging ecosystems. During my PhD studies, I aim to gain a deeper understanding about the different aspect of health problems in software packaging ecosystems and leverage this understanding by developing models to predict and recommend the health of these ecosystems. These models and associated tools 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.

As a short term goal, I will concentrate on **RQ1** by using data of the socio-technical package dependency network to come up with better algorithms for truck factor prediction and determining truck factor developers. Next, I will try to build a recommendation model for replacing truck factor developers. This study has close connections with [4], [12] mentioned in section II. Having better truck factor algorithm for determining number of truck factor and truck factor persons leads to increase of knowledge about the project health situation. In ecosystem level, this will gives us an overview of important people in the ecosystem and gives us opportunities to study their social and technical behaviour in that ecosystem.

Next, I will focus on **RQ2** to determine the important socio-technical reasons for migration of developers across projects within the ecosystem, as well as the reasons why developers are joining or leaving an ecosystem. This study will complement existing research on project survival by taking the socio-technical dimension of the ecosystem into account.

Third, I will refer the results obtained for **RQ2** by taking into account the socio-technical behaviour of project contributions (**RQ3**) to which extent they play a role in project attraction, abandonment, retention and migration of contributors.

Finally, for **RQ4** I will study to which extent the contributors' activity in social media plays a role in any of the aforementioned problems, which will hopefully lead to a further improvement of the proposed prediction and recommendation models.

References

[1] J. Coelho, M. T. Valente, L. Milen, and L. L. Silva, "Is this GitHub project maintained? Measuring the level of maintenance activity of open-source projects," *Information and Software Technology*, vol. 122, 2020.

[2] K. Blincoe, F. Harrison, and 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.

[3] A. Decan, T. Mens, and P. Grosjean, "An empirical comparison of dependency network evolution in seven software packaging ecosystems," *Empirical Software Engineering*, vol. 24, no. 1, pp. 381–416, 2019.

[4] M. Ferreira, M. T. Valente, and K. Ferreira, "A comparison of three algorithms for computing truck factors," in *International Conference on Program Comprehension (ICPC)*. IEEE, 2017, pp. 207–217.

[5] G. Iaffaldano, I. Steinmacher, F. Calefato, M. A. Gerosa, and 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.

[6] G. Avelino, E. Constantinou, M. T. Valente, and 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.

[7] F. Ricca and A. Marchetto, "Are heroes common in floss projects?" in *ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, 2010, pp. 1–4.

[8] C. Miller, D. G. Widder, C. Kästner, and 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.

[9] G. Fan, C. Wang, R. Wu, X. Xiao, Q. Shi, and 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.

[10] A. Decan and T. Mens, "What do package dependencies tell us about semantic versioning?" *IEEE Transactions on Software Engineering*, 2019.

[11] J. Coelho and M. T. Valente, "Why modern open source projects fail," in *Joint meeting on foundations of software engineering*, 2017, pp. 186–196.

[12] M. Ferreira, T. Mombach, M. T. Valente, and K. Ferreira, "Algorithms for estimating truck factors: a comparative study," *Software Quality Journal*, vol. 27, no. 4, pp. 1583–1617, 2019.

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

[14] G. Avelino, L. Passos, A. Hora, and M. T. Valente, "Measuring and analyzing code authorship in 1+ 118 open source projects," *Science of Computer Programming*, vol. 176, pp. 14–32, 2019.

[15] S. Goggins, K. Lumbard, and 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, and 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, and 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, and K. Schneider, "Are developers complying with the process: an xp study," in *International Symposium on Empirical Software Engineering and Measurement*, 2010, pp. 1–10.

[20] G. Avelino, L. Passos, A. Hora, and M. T. Valente, "A novel approach for estimating truck factors," in *International Conference on Program Comprehension*. IEEE, 2016, pp. 1–10.

[21] P. C. Rigby, Y. C. Zhu, S. M. Donadelli, and 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.

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

[23] H. S. Qiu, Y. L. Li, S. Padala, A. Sarma, and B. Vasilescu, "The signals that potential contributors look for when choosing open-source projects," *Proceedings of the ACM on Human-Computer Interaction*, vol. 3, no. CSCW, pp. 1–29, 2019.

[24] A. Schilling, S. Laumer, and 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.

[25] A. Decan, E. Constantinou, T. Mens, and H. Rocha, "Gap: Forecasting commit activity in git projects," *Journal of Systems and Software*, vol. 165, p. 110573, 2020.