An Empirical Study on the Survival Rate of GitHub Projects

Adem Ait
IN3 - UOC
Barcelona, Spain
aait_mimoune@uoc.edu

Javier Luis Cánovas Izquierdo
IN3 - UOC
Barcelona, Spain
jcanovasi@uoc.edu

Jordi Cabot
IN3 - UOC, ICREA
Barcelona, Spain
jordi.cabot@icrea.cat

ABSTRACT

The number of Open Source projects hosted in social coding platforms such as GitHub is constantly growing. However, many of these projects are not regularly maintained and some are even abandoned shortly after they were created. In this paper we analyze early project development dynamics in software projects hosted on GitHub, including their survival rate. To this aim, we collected all 1,127 GitHub repositories from four different ecosystems (i.e., NPM packages, R packages, WordPress plugins and Laravel packages) created in 2016. We stored their activity in a time series database and analyzed their activity evolution along their lifespan, from 2016 to now. Our results reveal that the prototypical development process consists of intensive coding-driven active periods followed by long periods of inactivity. More importantly, we have found that a significant number of projects die in the first year of existence with the survival rate decreasing year after year. In fact, the probability of surviving longer than five years is less than 50% though some types of projects have better chances of survival.

CCS CONCEPTS

• **Software and its engineering → Open source model**; • **Information systems → Information retrieval**.

KEYWORDS

Open Source Analysis, Survival Analysis, Mining Software Repositories, Empirical Study

ACM Reference Format:
Adem Ait, Javier Luis Cánovas Izquierdo, and Jordi Cabot. 2022. An Empirical Study on the Survival Rate of GitHub Projects. In *19th International Conference on Mining Software Repositories (MSR '22), May 23–24, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3524842.3527941

1 INTRODUCTION

Open Source Software (OSS) is a key player in the modern software industry, underlying the software infrastructure of a massive number of private and public-sector enterprises. OSS proposes a collaborative development model where the code is accessible and anyone can contribute. However, this very own collaborative nature becomes one of its main sources of problems, as developers abandoning the project causes serious sustainability problems [7, 18].

As such, when entering to any OSS ecosystem, either to search for a library to integrate into our software product or to collaborate in a project, the long-term sustainability or, at the very least, the level of activity of a project and its temporal evolution, should be a factor to consider when choosing the best OSS match for our goals.

Unfortunately, the task of exploring and selecting OSS projects is not an easy task. The number of OSS projects to explore is very large (e.g., GitHub stores more than 200 million repositories) and the selection process requires analysis data and processing mechanisms that are not trivial to use. Indeed, current filtering and query criteria supported by platforms such as GitHub are not sufficient to effectively locate good project candidates. Even less when it comes to choose projects that are *alive and kicking*. They mostly provide querying capabilities for basic repository information, such as name, technology, topics or number of stars; but they lack of more advanced features to spot active projects, which requires specific support for time-based metrics.

We propose to study the identification of active OSS projects via survival analysis, which is a set of methods and techniques for analyzing the time until an event occurs. This type of analysis pays special attention to the time-dependent nature of the OSS project information and allows us to understand the probability of survival of OSS projects.

More specifically, we perform a quantitative analysis of the survivability of OSS projects over time, focusing on their survival rate, evolution dynamics and the identification of factors that positive correlate with higher surviving rates. We focus our analysis on four ecosystems in GitHub, specifically: NPM packages, R packages, WordPress plugins and Laravel packages. By focusing in specific ecosystems we ensure similar development practices and structure but also a high variety on their purpose. We study the evolution of the project survival status over time (i.e., alive, zombie or dead) and perform a survival analysis of the projects in each ecosystem.

Our results revealed that the typical development process consists of intensive active periods, where the main activity is focused on coding tasks; followed by long periods of inactivity. With regard to the survival analysis, we have found that more than a half of the projects dies in their first four years. GitHub projects developed by organizations seem to have a higher survival rate, and the same happens when the community of the project is large.

We believe the results of our analysis are also useful to raise awareness on the volatility of Open Source projects, and the risks incurred by many organizations that heavily rely on Open Source projects without paying enough attention to their health nor contributing to their sustainability.

Our results complement previous works more focused on the characterization of individual contribution activity by taking a more

MSR '22, May 23–24, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9303-4/22/05…$15.00
https://doi.org/10.1145/3524842.3527941

global project perspective. And provide useful data that could be used by other projects looking to develop machine learning models to forecast the future evolution of a project.

The paper is structured as follows. Section 2 presents the background of the study. Sections 3 and 4 describes the methodology applied and presents the results, respectively. Section 5 highlights additional insights and discusses some future work. Section 6 presents the replicability package of our study, while Section 7 describes the threats to validity. Section 8 presents the related work, and Section 9 concludes the paper.

2 BACKGROUND

2.1 GitHub

The development of OSS projects is a collaborative effort where any contributor can jump in and help on the evolution of the project. Well-known solutions such as GitHub provide an online platform to enable this collaboration. GitHub is a social code hosting platform which, besides offering repository hosting services, provides features such as issue-tracking or pull request support to help to evolve OSS projects and manage the community behind the project.

A GitHub project is built upon the concept of a code repository. Along this paper, we use the terms *repository* and *project* indistinctly. The main elements of any GitHub project are: (1) commits, which becomes the change unit in any code repository and represent a set of changes in the code; (2) issues, which are used to report bugs, questions, or announcements of any kind; (3) pull requests, which are requests to perform changes in the project's codebase; (4) code reviews, which incorporate changes and suggestions to the code of pull requests; and (5) comments, which can be attached to issues, pull requests and code reviews, and are the main communication mechanism to debate any point in question of the project.

A GitHub project can also include a wiki and a forum (called discussion boards in GitHub), which we do not consider in our study due to their low usage and novelty [11] in the platform, respectively.

In GitHub we can find three types of users: (1) individuals, which represent users using the platform; (2) organizations, which represent a company, project or development initiative; and (3) bots, which provide services for repositories (e.g., automatically create issues or launch external tools each time an event occurs). Projects in GitHub can be owned by an individual (i.e., a user of the platform) or by an organization (i.e., an account representing a company, project or development initiative).

2.2 Time Series and Survival Analysis

Time series analysis refers to the set of methods and techniques to study sequences of data collected over an interval of time for a set of events. To perform this analysis, the recording of the data is done at consistent intervals of times rather than intermittently or randomly. In time series analysis, the main objective is to explore how event data changes over time, in particular, to find patterns in the data which depends on time.

Time series databases are a specific type of databases tailored to the analysis of time series data. This kind of databases offers support for (1) storing and retrieving time series data; (2) scaling, as time series data are extensive; and (3) providing visualization tools or easy integration components with third-party visualization tools, which are common tasks in time series analysis. TimeScaleDB[1] and InfluxDB[2] are two of the most common databases used in time series analysis.

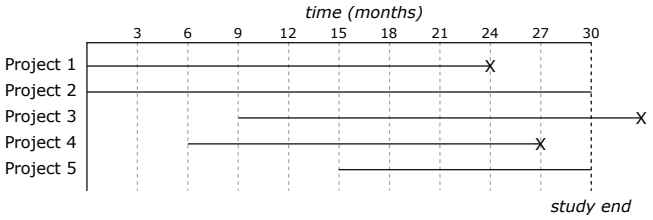


Figure 1: Examples of survival analysis data. Timelines represent project's activity while the symbol X represents the abandonment of the project.

Survival analysis is a statistical modeling technique to study the expected time for an event to occur [16]. Due to the data used in survival analysis, it could be considered as a special case of time series analysis. By applying survival analysis, we study the proportion of population which will survive to a particular event in a given period of time. Furthermore, survival analysis also considers censoring, which occurs when we have information about individual survival time, but we do not know the survival time exactly.

In the context of this paper, we consider that a project survives[3] when its development has not been abandoned at the time we collected the project's data activity (i.e., end of our study). Thus, the death event comes after a project activity stops. Note that censoring may occur whether a project is alive at the time we collected the data, but it is later abandoned. This situation is called right-censoring and usually happens in any kind of survival data as the event can generally occur after conducting an empirical study. Figure 1 shows different examples of projects that may be found in our study. As can be seen, projects 1 and 4 are considered abandoned ones (i.e., dead) while projects 2, 3 and 5 are considered alive ones; at the time our study ends. Note that project 3 is a right-censoring case, as it is considered alive, but it is later abandoned.

3 RESEARCH METHOD

In this section we describe how our study has been set up. We first present our research questions (Section 3.1), then we report on the dataset construction process (Section 3.2), and finally we present the main descriptive statistics of our dataset (Section 3.3).

3.1 Research Questions

In this paper we are interested in better understanding the survivability of OSS projects. To this aim, we study the activity of a set of GitHub repositories created in 2016 and observe their evolution until October 2021. More specifically, we have identified the following research questions:

RQ1 How does the project activity change over time? This research question analyzes the different status (or phases) a project

[1] https://www.timescale.com/

[2] https://www.influxdata.com/

[3] Along the paper we use the terms *dead* and *alive* as they are typically used in survival analysis, but it is important to note that an OSS project can come back to life after declared dead.

may transition during their lifespan (e.g., periods of time with high activity and others when the project may look dead). By studying the evolution of the project activity, we can better understand the dynamism and common evolution patterns of projects' lives.

RQ2 What is the survival rate? In this research question we focus on identifying projects that have died and when they have died (i.e., been abandoned and with no human activity). We then calculate the survival rate (i.e., dead vs. alive projects) which will allow us to understand the survivability of the projects across a number of dimensions.

To answer each research question we propose to collect a set of OSS projects within specific ecosystems and then follow a general-to-specific approach. We first analyze projects in each ecosystem to provide a general view, and then we conduct a deeper analysis grouping the full collection of projects to uncover pieces of evidence in projects of specific groups. Based on our knowledge and participation in different forums related to Open Source, we propose two factors to analyze the projects as groups, namely: project type and community size. On the one hand, there are two project types: those owned by an individual and those owned by an organization. The former lives in a user's GitHub profile, while the latter is developed within an organization account as commented in Section 2.1. Our hypothesis is that there will be differences between individual and organization projects, as the purposes and driving forces living in each account type are different.

On the other hand, we define three tiers for the project community size. We rely on the descriptive statistics of the project' community sizes to set the limits of these tiers. Thus, Tier 1 contains those projects with a number of contributors lower than the minimum value of the interquartile, Tier 2 contains those projects with a number of contributors which falls into the interquartile, and Tier 3 contains those projects with a number of contributors higher than the maximum value of the interquartile. The term contributor refers to any individual who participates in the project via commits, issues, pull requests, code reviews or comments. Our hypothesis is that projects in Tier 1 will behave different from those in Tiers 2 and 3; as the main responsibility to keep the project alive goes to a few contributors (sometimes even one or two contributors).

3.2 Dataset Construction

In this section we report on the construction of the dataset for our empirical study. The creation of the dataset followed three phases, namely: ecosystem selection, project analysis and database import. Next we describe each of these phases.

Ecosystem selection. To build our dataset we decided to select a set of GitHub projects within specific ecosystems. Focusing on specific ecosystems ensures similar development practices and structure but also high variety on their purpose. Our dataset contains GitHub projects of four ecosystems, namely: NPM packages, R packages, WordPress plugins and Laravel packages. To select the projects we applied the following criteria: (1) they have been created in 2016, (2) they have been updated at least once in 2016[4], and (3) they are part of the ecosystem of interest.

Table 1: Number of repositories in GitHub per ecosystem and creation date.

Topic	In 2016	All-time
`npm-package`	280	4,691
`r-package`	199	1,993
`wordpress-plugin`	540	7,223
`laravel-package`	108	1,424
Total	1,127	15,331

The gathering of the projects was done by querying the GitHub Search API. Criteria (1) and (2) are directly supported by specific time filters of the API, while the criteria (3) uses the filter which relies on repository topics offered by the same GitHub API. Thus, we used the `npm-package` topic for NPM packages, `r-package` topic for R packages, `wordpress-plugin` topic for WordPress plugins and `laravel-package` topic Laravel packages. As the GitHub Search API returns a maximum of 100 entries per query, we iteratively queried the API (i.e., using pagination) to collect all projects in the considered period of time[5].

Table 1 shows the number of projects created in 2016 per ecosystem (second column) and the size of these ecosystems in GitHub (last column). In total, we selected 1,127 projects. It is also important to note that none of the projects selected in our dataset is a fork of another project.

Project analysis. In this phase we cloned and analyzed the selected projects. The analysis was performed by SourceCred[6], a measurement tool for collaborative solutions such as GitHub projects, among others. SourceCred builds a collaboration graph from GitHub projects, where nodes represent assets of the repository (e.g., users, commits, comments, issues, pull requests, etc.) and edges represent relationships among them (e.g., a user authors a commit, a comment belongs to an issue, etc.). The graph also includes temporal information (e.g., timestamp of the creation of a commit or an issue), which we used to perform our study.

An important issue to consider when retrieving information from git repositories is the special support required for user disambiguation (i.e., different commit metadata information can refer to the same committer). As SourceCred relies on the GitHub API, which uniquely identify users by their username, it ensures a proper disambiguation. Furthermore, the use of the GitHub API also allows identifying bots, as the API flags this kind of users.

Database import. From the collaboration graphs extracted in the previous phase, we populated a Time Series Database (TSDB). We used a relational TSDB solution named TimescaleDB, which provides specific support to deal with time series data. For instance, we leveraged some functions provided by TimescaleDB, such as `time_bucket`, to manage our time series data efficiently.

[4]This removes projects that were mere mirrors of projects hosted elsewhere and that were only uploaded to GitHub for reference purposes, not really with the goal of starting/continuing their development there.

[5]An example of GitHub Search API query to get the first page of `r-package` projects is `https://api.github.com/search/repositories?q=topic:r-package created:2016-01-01..2016-12-31 pushed:>=2016-01-01&per_page=100&page=1`

[6]https://sourcecred.io/

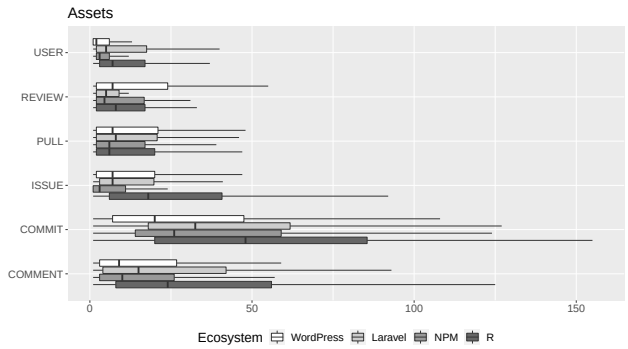


Figure 2: Assets per repository.

Table 2: Total number of assets.

Topic	Commits	PRs	Issues	Comments	Revs.	Users
npm-package	41,767	8,858	3,947	22,961	4,549	4,319
r-package	48,965	2,202	7,032	23,366	1,260	2,689
wordpress-plugin	134,426	16,152	27,348	103,999	16,878	17,163
laravel-package	11,070	1,841	1,561	5,310	228	2,005

The database schema includes a table for each repository asset considered in the study, namely: commits, issues, pull requests, comments, and code reviews. All table entries include a timestamp of the creation of the asset, thus representing events for each asset type. For the sake of efficiency in our queries, information regarding the author, the author type (i.e., whether the author is a bot or not) and repository of each asset was denormalized and included as columns in all tables. Thus tables include this shared columns plus an additional one with the unique identifier of the asset (e.g., the hash of a commit). In total, the database schema defines five tables (i.e., commits, issues, pull requests, comments, and code reviews) with five columns each (timestamp, repository, author, author type and unique identifier).

Once the database has been initialized, we performed the following steps to import the data from the collaboration graphs: (1) read the data from the collaboration graphs, (2) locate and extract the required node attributes, classified by its type (e.g., commits, issues, comments, etc.), and (3) insert the data into the corresponding table. For instance, given a graph node representing a commit, we retrieve the information about the hash, timestamp, repository name, author username, and whether the user it is a bot; and then insert the data into the `commits` table.

3.3 Dataset Descriptive Statistics

The dataset was built distinguishing each ecosystem. Figure 2 shows the distribution of the assets per ecosystem, while Table 2 shows the total number of assets per ecosystem. In total, our dataset contains 236,228 commits, 29,053 pull requests, 39,888 issues, 155,636 comments, 22,915 reviews and 26,056 unique users.

As commented in Section 3.1, in our study we used two factors to classify repositories, namely: community size and project type. Community size factor defines three tiers (or levels) which depend on the distribution of the community size of the projects in the ecosystem. Table 3 shows the range considered in each tier per ecosystem.

Table 3: Range considered in each tier per ecosystem for the community size factor.

Topic	Tier1	Tier2	Tier3
npm-package	[1, 1]	(1, 6]	(6, 13]
r-package	[1, 3]	(3, 19]	(19, 43]
wordpress-plugin	[1, 1]	(1, 6]	(6, 13]
laravel-package	[1, 2]	(2, 18.5]	(18.5, 43]

Table 4: Number of repositories per factor.

Factors		Tier1	Tier2	Tier3	Total
User		324	214	141	679
Organization		96	185	167	448
	Total	420	399	308	1,127

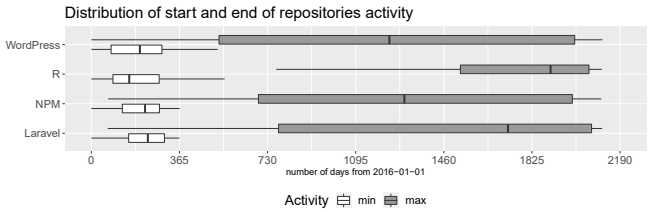


Figure 3: Distribution of starting and ending timestamps for the projects in each ecosystem.

On the other hand, Table 4 shows the number of projects per factor (see last column and row for totals) and the number of projects crossing both factors. We can appreciate a slight difference between user and organization projects. While user projects are usually smaller, with regard to the community size; organization projects tend to be larger.

Finally, Figure 3 shows the distribution of the starting and ending timestamps of the project lifespan. The time is reported by the number of days since the initial date considered in our study (i.e., January 1^{st}, 2016). For the sake of clarity, days are reported grouped by years. Starting and ending timestamps indicate the first and last events detected in the project. Note that even though our selection criteria collected projects created in 2016, there are some projects which show activity after the first year of life (i.e., projects created in 2016 but with a commit, issue, pull request, comment or code review created later). On the other hand, the distribution of ending timestamps is spread along almost the full time-lapse considered in our study.

4 RESULTS

We report in the following how we address each research question considered in our study.

4.1 RQ1

In this first research question, we study the evolution of the repository through different activity status along the time period considered in our study. We define a repository is either *alive* or *dead*, depending on whether there is any activity in the repository or not

at all, respectively. Furthermore, for alive projects, we differentiate between *running* ones, which are those projects that show at least some coding activity (i.e., commits) even if they have also other types of activity; and *zombie* ones, which include projects for which the only activity is non-coding (e.g., opening an issue or adding a comment) and bot activity but not actual coding contributions by human developers. To perform our analysis, we infer the repository status at a certain month by applying the previous criteria on the set of activity events occurring in that month of the project's life. We consider that all repositories start in the *running* status.

For a repository, we define its evolution path as a list of status, one per month, starting from the first month of life (e.g., `Running-Running-Zombie-Dead-...`). For the sake of clarity, when reporting this list we group repeated consecutive status and count the number of months (e.g., previous example would become `Running(2m)-Zombie(1m)-Dead(1m)`, where `Xm` means X months).

We then analyze the evolution paths for all repositories in our dataset, grouped by ecosystem, and represent them as a state machine, where each state corresponds to a status, and transitions indicate a change in the repository status. States are labelled with the name of the status (i.e., *running*, *zombie* or *dead*) and the average time (in months) the project activity stays in that state. Transition labels indicate the probability to change status each month, calculated as the number of times such outbound transition is detected in the evolution paths divided by the total number of outbound transitions with the same source state, and expressed as percentage. Note that we calculate the probability regardless the position of the transition in the path (e.g., `Running-Zombie-Dead-Running-Zombie` includes two transitions `Running-Zombie`). Furthermore, the sum of all outbound transitions must be equal to 100%. Given this state machine definition, self-transitions represent a repository status that repeats on consecutive months.

Figure 4 shows the state machines for the four ecosystems. As can be seen, the behavior for all ecosystems are alike. Projects tend to not move from one state to another, but if they do so, it is likely that they move from *running* state to *dead* one, thus meaning that projects show intermittent activity. An example illustrating this intermittent behavior is the project `asheabbott/wordpress-custom-post-types-taxonomies`, which is part of the WordPress ecosystem and its evolution path is `Running(1m)-Dead(14m)-Running(1m)-Dead(19m)-Running(1m)-Dead(31m)`. Another example is the project `gswalden/pkg-env`, which is part of the NPM ecosystem and its evolution path is `Running(1m)-Dead(2m)-Running(1m)-Dead(17m)-Running(1m)-Dead(50m)`. For the sake of space, we chose two examples of projects which died prematurely (see the long period of time of the last *dead* state) and therefore have short evolution paths; however, most of the projects presents this pattern, where short *running* states alternate with longer *dead* states.

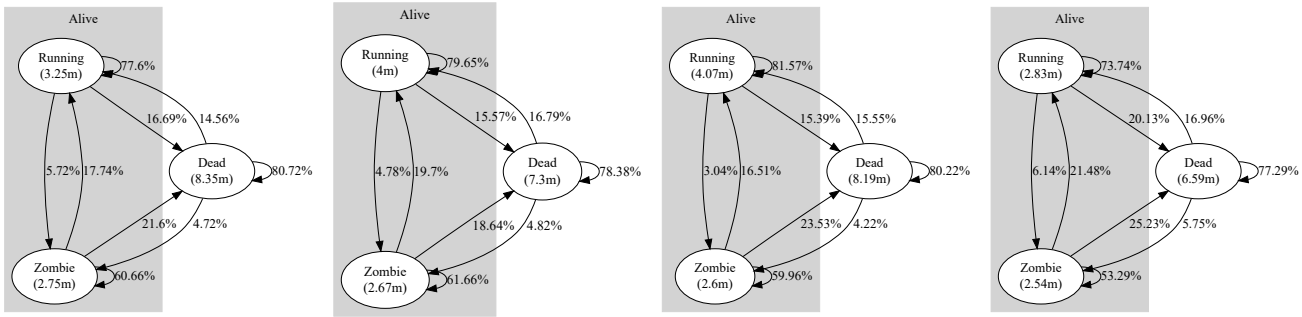
The fact that the percentage to move from *dead* to *zombie* states is lower than to move from *dead* to *running* states reveals the impact of coding contributions (and not on the non-coding actions alone, for instance, via issues or comments) in the project development process. The project `eddelbuettel/rcppmsgpack` from the R ecosystem is a good example to illustrate this. The project goes from *running* to *dead* states with similar duration (i.e., it has a period of X months running, and then it has a period of X months dead). An excerpt of its evolution path is `...-Running(3m)-Dead(3m)-Running(3m)-Dead(4m)-Alive(4m)-Dead(4m)-Running(6m)-Dead(5m)-Running(6m)-...`, with exceptional transitions to *zombie* state (i.e., only once in `...-Zombie(6m)-Dead(6m)-Running(6m)-...`), which may reveal a code-focused repository but also a maintenance process.

Furthermore, once projects move to the *zombie* state, the probability to move to the *dead* state is higher in the NPM, WordPress and Laravel ecosystems. Given this behavior, the *zombie* state could be considered the prelude to the *dead* state. In fact, further analysis revealed that 15.96% of the transitions from *zombie* to *dead* are final ones (i.e., there are no more transitions and therefore the project is dead), and it is detected in 21.30% of the projects of our dataset. It is interesting to note that the average time a project stays in the dead state is higher than the average time in the other states. We also computed the average number of transitions between different states per ecosystem, which can give a hint on the diversity on the projects' evolution paths. The average number of transitions in each ecosystem is: (1) R, 18.18; (2) NPM, 11.74; (3) Laravel, 15.99; and (4) WordPress, 11.16.

In addition to this evolution path study, we performed the same analysis on all data using factors regarding the community size (i.e., using Tiers 1, 2 and 3), and the project type (i.e., User or Organization). Figures 5 and 6 show the state machines for project type and community size factors, respectively. Regarding the results for project type factor, we can only see a slight variation in both average state time and state transitions. However, average state time in *running* seems to be higher in Organization projects, while *dead* average time is lower. Hence, it appears to be greater activity in Organization projects.

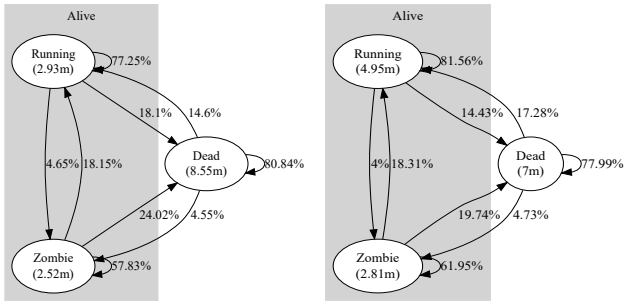
On the other hand, when considering the community size factor, we detect a notable difference in the average time in each state. The average time in *running* and *zombie* states increases along with the tier, while *dead* state decreases. Tier 3 is the only tier with the highest average time in *running* than in anyone, thus indicating that there are longer periods of activity. Concerning state transitions, there are no notable differences.

Answer to RQ1: The analysis of projects' activity evolution during their lifetime reveals an intermittent activity, which moves from *running* (i.e., any activity) to *dead* (i.e., no activity at all) states. Besides, the average time in *dead* state is approximately two times the average time in the *running* state, thus revealing longer periods of inactivity. On the other hand, we detected that when a project moves to the *zombie* state (i.e., non-development activity) it is likely to move to the *dead* state. These results suggest that the typical development process in the four ecosystems evolves in short intensive periods of time with long periods of inactivity. When having into account the project type and community size factors, we found that projects owned by organizations ensure higher levels of activity, and the same happens for those projects with bigger community sizes.



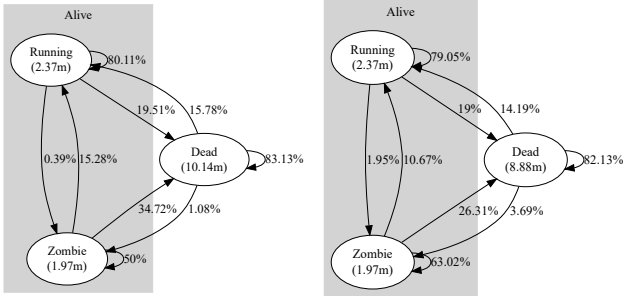
(a) NPM **(b) R** **(c) WordPress** **(d) Laravel**

Figure 4: State machines representing the evolution paths in each ecosystem. State labels indicate the name of the status and the average time (in months). Transition labels indicate the probability to change status each month.

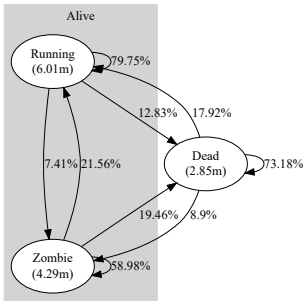


(a) User **(b) Organization**

Figure 5: State machines representing the evolution paths grouped by project type. State labels indicate the name of the status and the average time (in months). Transition labels indicate the probability to change status each month.



(a) Tier 1 **(b) Tier 2**



(c) Tier 3

Figure 6: State machines representing the evolution paths grouped by community size. State labels indicate the name of the status and the average time (in months). Transition labels indicate the probability to change status each month.

4.2 RQ2

In this research question we analyze the survival rate of the projects considered in our dataset. While the previous research question focused on the monthly evolution of projects' activity status, this research question considers how many of them were dead (i.e., with no human activity) at the end of the period of time considered in our study. We consider a project survives, i.e., it is *alive*, if it has shown activity of any kind in the last six months of the period of time considered in our study; otherwise, we consider the project did not survive and therefore is *dead*). For dead projects, the moment of death is the timestamp of the last activity in the project.

Table 5 shows the amount of dead and alive projects of our dataset, separated by ecosystem. As can be seen, approximately more than 50% of the repositories die within the period of time considered in our study for all four ecosystems. Table 6 shows an expanded analysis on dead projects, showing the number of dead projects we found per year. Note that the year column refers to the age of the project when it died. We can see that, except for the `r-package` ecosystem, the most number of deaths are recorded in the first year; and by the fourth year, 50% of the projects are dead.

With these data, we calculated the probability of survival over time per ecosystem. In particular, we computed the survivor function $S_e(t)$ for each ecosystem e, which gives the probability that a project survives longer than some specified time t. Figure 7a shows the survival curve, which is the result of the survivor function, with its confidence interval per ecosystem (see shadowed areas per ecosystem). As can be seen, except for the `r-package` ecosystem, the probability of survival when $t > 36\,months$ approaches 50%,

Topic	# Projects	Dead	Alive
npm-package	280	189 (67.50%)	91 (32.50%)
r-package	199	98 (49.25%)	101 (50.75%)
wordpress-plugin	540	360 (66.57%)	180 (33.43%)
laravel-package	108	58 (53.70%)	50 (46.30%)

Table 5: Distribution of dead and alive projects.

	Year					
Topic	1	2	3	4	5	6
npm-package	56 (20%)	50 (17.86%)	36 (12.86%)	28 (10.00%)	18 (6.43%)	1 (0.36%)
r-package	8 (4.02%)	13 (6.53%)	13 (6.53%)	27 (13.57%)	34 (17.09%)	3 (1.51%)
wordpress-plugin	144 (26.67%)	69 (12.78%)	60 (11.11%)	48 (8.89%)	33 (6.11%)	6 (1.11%)
laravel-package	21 (19.44%)	11 (10.19%)	9 (8.33%)	13 (12.04%)	4 (3.70%)	0 (0%)

Table 6: Analysis of dead projects. Second column indicates the age of the project when it died.

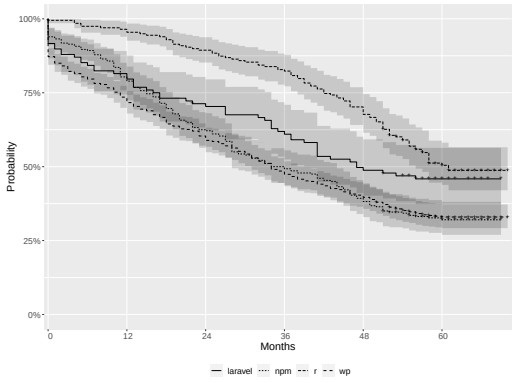
and it drops lower than 50% when $t > 48\,months$. The r-package ecosystem shows greater probability of survival over time, although it suddenly drops when $t > 60\,months$.

Additionally, we calculated the probability of survival over time per (a) project type and (b) community size, Figures 7b and 7c show the corresponding functions, respectively. The results reveal that organization projects have a higher probability of survival than personal projects, and that the bigger the community involved in the development the higher the chance of surviving. Tier 1 projects tend to die prematurely, while Tier 3 projects have a higher chance of having a longer lifespan. Usually, Tier 1 projects have a few months of activity before dying. An illustrative example of this case is the repository lehoangduc/laravel-packages of the Laravel ecosystem, which lives during approximately two months and then dies, having a lifespan of one month and six days. In fact, all repositories that die within the first year in the Laravel ecosystem are from Tier 1. On the other hand, 59.25% of all projects with a lifespan of more than five years are from Tier 3, and just 7.40% of those are from Tier 1. An example of a Tier 3 repository with a lifespan of 67 months is musonza/chat, which has activity in the last three months of the study. Figure 7d shows that these results hold for all six community/owner combinations.

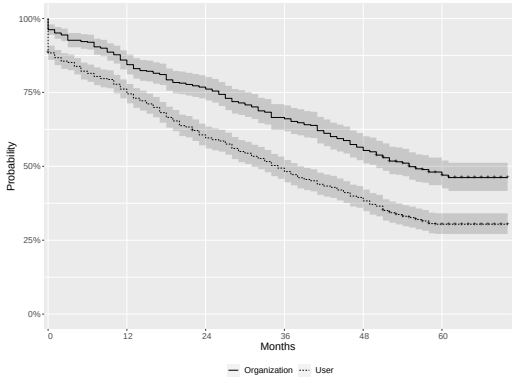
Answer to RQ2: More than a half of the analyzed projects die in their first four years. In fact, the probability of survival is lower than 50% beyond the fifth year of live. When considering project type and community size, organization projects have a higher probability of survival, and the same happens for projects with bigger community sizes.

5 DISCUSSION

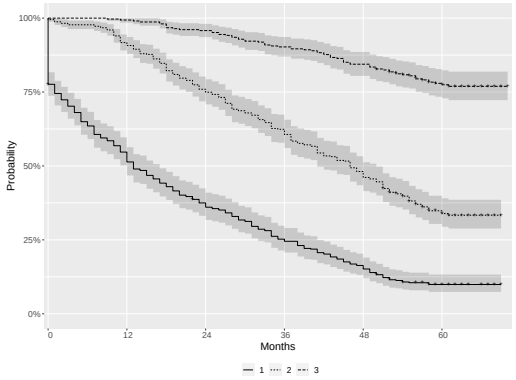
Beyond the main findings reported so far, we would like to highlight some additional insights and discussion points derived from this work.



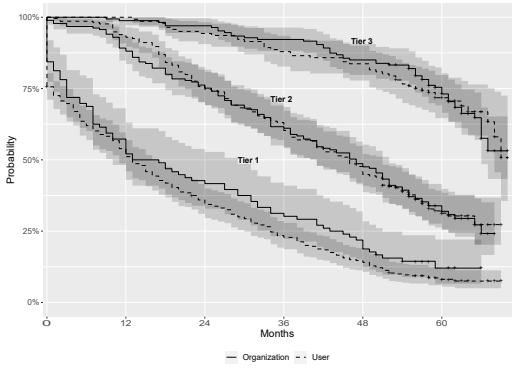
(a) Probability per ecosystem



(b) Probability per project type



(c) Probability per community size



(d) Probability per community size and project type

Figure 7: Probability of survival over time.

Analysis of a project's activity to do forecasting. The low survival rate detected in our study could motivate further analysis of potential correlations between the project's survivability and patterns in the repository activity. For instance, Figure 8 shows three examples of project's activity during its first year of life via line graphs. In this graph, the activity for each project resource is represented by a line (see *commits*, *issues* and *prs*), and also their aggregation (see *general* line). The horizontal axis shows the time since the beginning of the project (in months), while the vertical axis shows the count of the project resource measured. The examples shown in the Figure illustrate different situations that may help to forecast the project's future, specifically: (1) a long-living project with similar activity in all its resources (cf. Figure 8a); (2) a project which dies after some months of life (cf. Figure 8b); and (3) a project with intermittent commit activity (cf. Figure 8c) and eventually died at the age of three (not shown in the Figure). We believe that this kind of temporal information could help on building intelligent solutions to deploy forecast models in order to know whether a project is likely to survive given the activity during its starting months. The work by Coelho et al. [5] could be useful here. For instance, we could see to integrate some temporal data as new features in the Machine Learning (ML) model.

Recommendations to keep projects alive. Beyond a pure ML-based forecasting, we also believe that the study of the set of alive projects could help to identify some red flags that could signal the decay of a project. Or the complete opposite, factors that positively influence the chances of having a long and peaceful live. Some of these factors (i.e., entering a *zombie* status for the former, growing your community size for the latter) have been discussed before and therefore could be used as immediate recommendations.

But a more fine-grained analysis could involve techniques for measuring the similarity between time series data, such as Dynamic Time Warping (DTW). DTW is a method which takes two time series data sequences and returns the minimum distance between them. By analyzing the minimum distance between the time series of alive projects we could identify concrete temporal patterns that are shared by successful projects. These evidences could be used to build systems which monitor projects in order to send early warning messages to project communities of projects at risk.

Impact of non-coding contributors. There is a general consensus on the importance of non-coding contributors. In fact, previous work (i.e., [13]) demonstrated the relevance of non-coding contributions in a set of NPM projects in GitHub. Nevertheless, our study has shown a low impact of non-coding contributors in the project's survivability (e.g., low presence of *Zombie* state in evolution paths). We believe it is important to study (and deploy) mechanisms that help these non-coding contributors to keep the project active. Either by becoming code contributors themselves, by positively influencing code contributors or by making their life easier proactively helping in the detection, classification and prioritization of project tasks

Project's survival beyond the original repository. The development of OSS projects in online social platforms such as GitHub usually involves activity beyond the original repository, as contributors can fork it and advance the project in their own user (or organization) space. Contributions to these forks can later be sent back to the original repository via pull requests, but they can also evolve independently. While our study did consider the pull requests it did not count as surviving projects those projects where the original repository died, but a fork was still alive. We believe it would be interesting to study the potential impact of forks on our survival rate results.

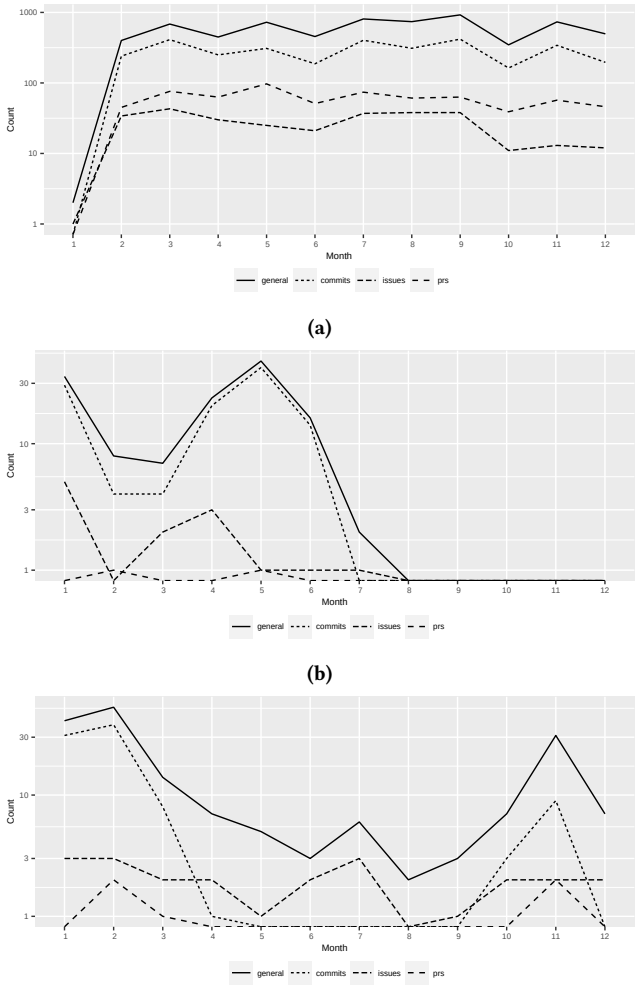


Figure 8: Examples of evolution of activity in GitHub repositories during the first months of life.

Impact of bots in the survival rate In the study performed in RQ2, where we studied the survival analysis according to the origin of the status of a project, we filter out the bot activity of a project (see also Section 7). Projects with no human activity are considered dead at the end of the study even if bots may be still updating some project dependencies or automatically performing some other minor maintenance tasks. Given the growing popularity of bots in all areas, we believe it is worth to have an open discussion on the

future role of bots in software development and how they should be counted in empirical software analysis works.

The sustainability challenge. Being the core of the digital infrastructure, the evolution of Open Source projects has a significant impact on the future of our society. However, the low survival rate reported by our findings put at risk this balance and affects the sustainability of Open Source. We believe that the low survival rate reported by our findings emphasizes the need for more sustainability efforts in the long term. On top of some of the discussion points above, initiatives such as SustainOSS[7] or CHAOSS project[8] could also help on contextualizing our results and exploring how to address this issue.

6 REPLICABILITY PACKAGE

To facilitate the replication of our study, we have prepared a GitHub repository[9] for researchers interested in repeating or complementing our evaluations. The repository includes the main elements of our dataset (i.e., graphs in different formats) together with the data used in the study.

7 THREATS TO VALIDITY

Our work is subjected to a number of threats to validity, namely: (1) internal validity, which refers to the inferences we made; and (2) external validity, which is related to the generalization of our findings.

Regarding the internal validity, in this study we relied on the collaboration graphs provided by SourceCred, which uses the information available via the GitHub API. The data provided by this API may not be complete, which causes either missing edges or the generation of dangling edges in the graph. In our study, we have detected that missing edges appear when repository assets (e.g., a commit, issue, pull request, review or comment) are not connected to the corresponding author, which usually happens when the author does not exist anymore because he/she abandoned the platform. Nonetheless, we checked that the ratio of missing edges is on average 6% for the ecosystems considered. Regarding the dangling edges, we checked that our study was not affected by this issue.

Another threat which affects the internal validity is the right-censoring of the data, which appears when it is not possible to know the exact survival time of all the participants in the study, e.g., due to time limits in the study. This means that it may happen that a project classified as dead in our study may reactivate after the six-year period considered in our work. As most survival analyses, it is important to consider this analytical problem when interpreting the results of the study.

Related to the data quality, the identification of actual software developing projects (i.e., engineered software projects) and not toy projects (e.g., projects addressing homework assignments) is a challenging task [22]. Projects in our dataset are collected from a set of ecosystems devoted to create packages or plugins for well-known platforms, which should restrict the number of toy projects collected. Nevertheless, we believe that this threat as those regarded as toy ones may later be reused (e.g., as templates) to create packages or plugins, with the consequent impact if the toy project is not maintained. Also, to distinguish between users and bots SourceCred also relies on the GitHub API. However, this task is far from trivial [10] and sometimes not all the information is available via the API, thus we may have missed some bots, which could have influenced in showing some project as alive when the only activity is due to bots.

Another threat is related to our choice of methods and techniques. To minimize this, we have carefully reported each step of our study and also provided a companion package to promote replicability (cf. Section 6).

As for the external validity, note that our dataset is based on the set of GitHub projects tagged with specific labels to identify their ecosystem, which was captured as of October 15^{th} 2021; and therefore our results should not be directly generalized to other types of Open Source projects without proper comparison and validation. Note that we consider the topic of a repository in GitHub as a source of truth, but it may be possible that some repositories are incorrectly classified. Also, the results of this paper should not be generalized as a measure of success (or failure) for GitHub projects. We acknowledge that there may be a complex interrelationship between survival analysis and such a measure, but further work is needed.

8 RELATED WORK

The study of OSS software development is a broad area of research which has been studied from a number of different perspectives [6, 8, 14]. In this section we specifically review previous related work on survival and abandonment analysis, as well as temporal analysis in GitHub.

8.1 Survival and Abandonment Analysis

The work by Samoladas et al. [23] presented a framework for assessing the survival probability of a FLOSS project, and applied it to a set of OSS projects. The results on survival probability are similar to ours. Unlike our work, their framework relied only on commit activity, and they categorized the projects according to their purpose (e.g., database, multimedia, etc.).

Other than the previous work, the study of survivability of OSS projects has usually been related to the application of specific contributor activity metrics. Avelino et al. [1] studied the Truck Factor Developer Detachment (TFDD) in Open Source, that is, what happens when influential developers abandon the project. They performed an empirical study with 1,932 GitHub projects and studied how TFDD affected their survival rate. They concluded that TFDD happen in Open Source projects and identified enablers and barriers to overcome the potential abandonment of a project.

The work by Bao et al. [2] investigated whether newcomers will become Long-Term Contributors (LTCs) in GitHub projects, that is, contributors who stay within the project for a long time and therefore play important roles in the success of the project. They built a dataset of 917 projects and used several time intervals to build a predictive model. Related to the LTCs metric, the work by Kumar Eluri et al. [9] applied machine learning methods to GitHub activity data to predict whether a contributor will become a LTC of the project. Their results reveal that random forest is the most

[7] https://sustainoss.org/
[8] https://chaoss.community/
[9] http://hdl.handle.net/20.500.12004/1/C/MSR/2022/395

effective classifier and that the number of followers is the most important feature to predict the LTC status.

Recently studies of contributor disengagement have been conducted by Miller et al. [20] and Iaffaldano et al. [12]. The former studied the reasons why contributors disengage by performing a survey and building a survival model to quantify factors which predict disengagement. The latter is a position paper which explored the active/inactive/abandon cycles in the contributor activity track.

Krishnamurthy [17] studied whether the development process of the top 100 mature projects in Sourceforge is driven either by a small set of core developers or by an organized community. Among its results, the study reveals that the most of the projects are developed by individuals, rather than communities. Calefato et al. [3] analyzed the lifecycle of OSS project developers concerning their breaks and disengagement. Their results show that breaks are rather common, specially in core contributors; and that once a contributor disengages it is less likely to rejoin the project. A similar study was conducted by Coelho et al. [4], who performed a survey on a set of deprecated Open Source GitHub systems to analyze the reasons of the failure of projects.

Unlike previous works, ours does not focus on specific contributor activity metrics, but on the overall activity of the project. Nevertheless, these previous works could complement our approach, e.g. we could try to correlate the presence of some specific types of contributors with the project survival.

Finally, the work by Khondhu et al. [15] studied the inactivity levels in OSS projects to characterize the project's maintenance status when developers hand off the project to other developers. They define project's status similar to ours (e.g., dormant or active in their case) and study the impact in the project's maintainability index. Although they did not aim at studying projects' survivability, their approach may help to better characterize the survival of OSS projects. On the other hand, the work by Coelho et al. [5] proposed an approach to measure the level of maintenance activity of projects and trained a machine learning model to infer such maintenance status in GitHub projects. Their work also performed a survival analysis of OSS projects, but using different dimensions to ours and based on archived or self-declared unmaintained projects while our analysis performs a more exhaustive analysis of all available projects in the targeted ecosystems. Furthermore, the survival determination analysis relied on their ML trained model while we used the actual data from project's repositories.

8.2 Temporal Analysis in GitHub

GitHub has been subject of a limited number of studies with a focus on temporal analysis. For instance, Mitropoulos et at. [21] studied different temporal aspects in the JavaScript code of 10,000 websites, including the development pace, dependency evolution and quality of code evolution. They concluded that the lifespan of most of the JavaScript code is short, thus indicating a high development pace. These results seem to be aligned with the results of our work.

The relationship and evolution of different software popularity measures in GitHub was studied by Zerouali et al. [26]. In particular, they performed a correlation analysis of popularity measures in libraries.io, NPM and GitHub; both within the repository and cross repositories. Their results revealed that many popularity metrics are not strongly correlated, thus implying that the use of different metrics may produce different outcomes.

Varuna et al. [25] proposed a model for predicting trends in GitHub, in particular, the repository, language and domain trends. Using the main project's activity events, they proposed a predictor with time series forecasting using Long Short Term Memory (LSTM) model.

A study of the reasons to fork in GitHub was performed by Zhou et al. [27]. They classified forks into social ones, when done using GitHub forking feature; and hard ones, when the fork is done making a separate copy of the repository. Their results reveal that hard forks often evolve out of social forks rather than being planned deliberately. Our study does not include forked projects, however, it could be extended in the future to consider forked projects as extensions of the lifetime of the original project.

Overall, we believe our study covers a gap in the systematic study of survival rate and early development dynamics in open source projects by performing a systematic study on top of four popular project ecosystems. We hope our results can be used as the basis of additional analysis that help improving the survival rate, as we also discussed in Section 5.

Last but not least, survival analysis may also be related to the study of project's success, although we believe such relationship should be studied carefully (cf. Section 7). The works by Midha et al. [19] or Subramaniam et al. [24] identify project's characteristics which may influence its success, and could be helpful to explore this relationship, in particular, time-dependent characteristics.

9 CONCLUSION

We have conducted an empirical study on the survivability of 1,127 GitHub projects, collected from four ecosystems, specifically: NPM packages, R packages, WordPress plugins and Laravel packages. Our results reveal that the prototypical development process consists of intensive active periods, where the main activity is focused on coding tasks; followed by long periods of inactivity. An analysis of the survival rate showed that many projects die in their first year of life and more than a half of them does not live more than four years though projects developed by an organization or that managed to attract a large number of contributors have better chances of survival.

We believe these results are useful to understand the dynamics of Open Source in online platforms such as GitHub. These results are also useful to raise awareness on the volatility of Open Source projects and the risks incurred by many organizations that heavily rely on Open Source projects without paying attention to their health nor contributing to their sustainability. Note that it could be argued that some projects are small and focused, and therefore their development naturally stops shortly after its creation. We believe this may be true only for a few number of projects as at least some minimal maintenance work (e.g., to fix potential security vulnerabilities) is generally required, as it happens in the projects of our dataset, where projects are packages (or libraries) from an evolving software ecosystem.

This is part of our future work, as well as expanding the number of projects considered in the study to cover other ecosystems. We are also interested in studying how to consider the evolution of project's community size in survival analysis, as current approach

considers the community size as fixed, but it actually evolves along the lifespan of the project. Other interesting topics to consider as further work have been commented in Section 5 and include, for instance, the construction of intelligent systems able to forecast the survivability of OSS projects and to flag those projects at risk of dying soon (e.g., projects entering the zombie state) so that corrective community actions can be put in place.

ACKNOWLEDGEMENTS

This work has been partially funded by the Spanish government (LOCOSS project - PID2020-114615RB-I00).

REFERENCES

[1] Guilherme Avelino, Eleni Constantinou, Marco Tulio Valente, and Alexander Serebrenik. 2019. On the Abandonment and Survival of Open Source Projects: an Empirical Investigation. In *Int. Symp. on Empirical Software Engineering and Measurement*. 1–12.

[2] Lingfeng Bao, Xin Xia, David Lo, and Gail C. Murphy. 2021. A Large Scale Study of Long-time Contributor Prediction for GitHub Projects. *IEEE Trans. Software Eng.* 47, 6 (2021), 1277–1298.

[3] Fabio Calefato, Marco Aurélio Gerosa, Giuseppe Iaffaldano, Filippo Lanubile, and Igor Steinmacher. 2021. Will you Come Back to Contribute? Investigating the Inactivity of OSS Core Developers in GitHub. *Empir. Softw. Eng. (to appear)* (2021).

[4] Jailton Coelho and Marco Tulio Valente. 2017. Why Modern Open Source Projects Fail. In *Joint Meeting on Foundations of Software Engineering*. 186–196.

[5] Jailton Coelho, Marco Tulio Valente, Luciano Milen, and Luciana Lourdes Silva. 2020. Is this GitHub Project Maintained? Measuring the Level of Maintenance Activity of Open-source Projects. *Inf. Softw. Technol.* 122 (2020), 106274.

[6] Valerio Cosentino, Javier Luis Cánovas Izquierdo, and Jordi Cabot. 2017. A Systematic Mapping Study of Software Development with Github. *IEEE Access* 5 (2017), 7173–7192.

[7] Kevin Crowston and James Howison. 2006. Assessing the Health of Open Source Communities. *Computer* 39, 5 (2006), 89–91.

[8] Kevin Crowston, Kangning Wei, James Howison, and Andrea Wiggins. 2012. Free/libre Open-source Software Development: what we Know and what we do not Know. *ACM Comput. Surv.* 44, 2 (2012), 7:1–7:35.

[9] Vijaya Kumar Eluri, Thomas A. Mazzuchi, and Shahram Sarkani. 2021. Predicting Long-time Contributors for GitHub Projects Using Machine Learning. *Inf. Softw. Technol.* 138 (2021), 106616.

[10] Mehdi Golzadeh, Damien Legay, Alexandre Decan, and Tom Mens. 2020. Bot or Not?: Detecting Bots in Github Pull Request Activity Based on Comment Similarity. In *Int. Conf. on Software Engineering*. ACM, 31–35.

[11] Hideaki Hata, Nicole Novielli, Sebastian Baltes, Raula Gaikovina Kula, and Christoph Treude. 2022. Github Discussions: an Exploratory Study of Early Adoption. *Empir. Softw. Eng.* 27, 1 (2022), 3.

[12] Giuseppe Iaffaldano, Igor Steinmacher, Fabio Calefato, Marco Aurélio Gerosa, and Filippo Lanubile. 2019. Why do Developers Take Breaks from Contributing to OSS Projects?: a Preliminary Analysis. In *Int. Workshop on Software Health*. 9–16.

[13] Javier Luis Cánovas Izquierdo and Jordi Cabot. 2022. On the Analysis of Non-coding Roles in Open Source Development. *Empir. Softw. Eng.* 27, 1 (2022), 18.

[14] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. Germán, and Daniela E. Damian. 2016. An In-depth Study of the Promises and Perils of Mining Github. *Empir. Softw. Eng.* 21, 5 (2016), 2035–2071.

[15] Jymit Khondhu, Andrea Capiluppi, and Klaas-Jan Stol. 2013. Is it all Lost? a Study of Inactive Open Source Projects. In *Int. Conf. on Open Source Software: Quality Verification*, Vol. 404. 61–79.

[16] David G. Kleinbaum and Mitchel Klein. 2005. *Survival Analysis: A Self-Learning Text*. Springer Science and Business Media, LLC.

[17] Sandeep Krishnamurthy. 2002. Cave or Community?: an Empirical Examination of 100 Mature Open Source Projects. *First Monday* 7, 6 (2002).

[18] Josianne Marsan, Mathieu Templier, Patrick Marois, Bram Adams, Kevin Carillo, and Georgia Leida Mopenza. 2019. Toward Solving Social and Technical Problems in Open Source Software Ecosystems: Using Cause-and-effect Analysis to Disentangle the Causes of Complex Problems. *IEEE Softw.* 36, 1 (2019), 34–41.

[19] Vishal Midha and Prashant Palvia. 2012. Factors Affecting the Success of Open Source Software. *J. Syst. Softw.* 85, 4 (2012), 895–905.

[20] Courtney Miller, David Gray Widder, Christian Kästner, and Bogdan Vasilescu. 2019. Why do People Give up Flossing? a Study of Contributor Disengagement in Open Source. In *Int. Conf. on Open Source Systems*, Vol. 556. 116–129.

[21] Dimitris Mitropoulos, Panos Louridas, Vitalis Salis, and Diomidis Spinellis. 2019. Time Present and Time Past: Analyzing the Evolution of Javascript Code in the Wild. In *Int. Conf on Mining Software Repositories*. 126–137.

[22] Nuthan Munaiah, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. 2017. Curating GitHub for Engineered Software Projects. *Empir Software Eng.* 22 (2017), 3219–3253.

[23] Ioannis Samoladas, Lefteris Angelis, and Ioannis Stamelos. 2010. Survival Analysis on the Duration of Open Source Projects. *Inf. Softw. Technol.* 52, 9 (2010), 902–922.

[24] Chandrasekar Subramaniam, Ravi Sen, and Matthew L. Nelson. 2009. Determinants of Open Source Software Project Success: a Longitudinal Study. *Decis. Support Syst.* 46, 2 (2009), 576–585.

[25] T. V. Varuna and Anuraj Mohan. 2019. Trend Prediction of GitHub Using Time Series Analysis. In *Int. Conf. on Computing, Communication and Networking Technologies*. 1–7.

[26] Ahmed Zerouali, Tom Mens, Gregorio Robles, and Jesús M. González-Barahona. 2019. On the Diversity of Software Package Popularity Metrics: an Empirical Study of NPM. In *Int. Conf on Software Analysis, Evolution and Reengineering*. 589–593.

[27] Shurui Zhou, Bogdan Vasilescu, and Christian Kästner. 2020. How has Forking Changed in the Last 20 Years?: a Study of Hard Forks on GitHub. In *Int. Conf. on Software Engineering*. 445–456.