Analyzing the Robustness of Open Source Software Ecosystems to the Loss of Contributors: A Case Study

Zhendong Sha, Alice Petrov, Yuan Tian, Ting Hu

School of Computing, Queen's University, Canada

Abstract

Context: The health and sustainability of an open-source software (OSS) ecosystem depends on its contributors' initiative. Unfortunately, studies have shown that OSS projects often suffer from high contributor turnover rates due to their open-source nature. High contributor turnover rates can have negative impacts on the community involved in individual projects and the ecosystem that relies on such projects.

Objective: Propose a computational model to quantify the robustness of a software ecosystem to contributor loss and perform a case study on two OSS ecosystems, i.e., Ruby and PyPI.

Method: We utilize a simulation method to analyze project extinction risk due to contributor loss at the ecosystem-level. We apply the proposed method on 12,267,933 and 1,459,322 commits scraped from 102,447 Ruby projects and 69,311 PyPI libraries respectively, hosted on GitHub. To identify the factors that influence the robustness of ecosystems, we propose an ecosystem simulation method to generate artificial ecosystems with different control parameters.

Results: We find that contributor turnover and project abandonment frequently happen in Ruby and PyPI ecosystems. Moreover, the preference of developers to contribute to large projects in an OSS ecosystem negatively affects the robustness of the ecosystem. Both studied ecosystems are less robust to the loss of active contributors who either intensively or extensively contribute to the ecosystem than the random loss of contributors.

Conclusion: Our proposed methods can be leveraged to analyze the robustness of a software ecosystem to contributor loss over time. Moreover, we provide a ecosystem simulation method to analyze how various factors determine the robustness of an OSS ecosystem, demonstrating the potential of

testing a hypothesis without empirical data for ecosystem robustness analysis.

Keywords: Software Ecosystem, Computational Model, Robustness of OSS Ecosystems, Developer Turnover, Case Study, Simulation Analysis.

---

1. Introduction

Over 80 percent of the software used in any technology product or service today is open source, and this trend is growing (Zemlin, 2017). The expanding open source software (OSS) community attracts many developers who eagerly contribute to OSS projects. However, despite the importance of OSS projects to the software industry and the increasing number of contributors, many OSS projects fall short of active maintainers (Marsan et al., 2018) and thus bring risk and have severe negative impacts on other software systems that rely on them (Schneider, 2018; Apache Log4j, 2021). Additionally, OSS projects suffer from a higher *contributor turnover* rate when compared to closed software projects since most OSS contributors are not paid for their contributions (Miller et al., 2019). Typical reasons for leaving an OSS project include major life changes, lacking peer support, losing interest, etc. (Miller et al., 2019). Recent studies have found that a high contributor turnover rate can lead to knowledge loss (Nassif and Robillard, 2017; Rigby et al., 2016), quality degradation (Foucault et al., 2015), and project extinction (Coelho and Valente, 2017).

In the literature, researchers have mainly analyzed the prevalence and impact of contributor turnover on individual projects (Izquierdo-Cortazar et al., 2009; Foucault et al., 2015; Rigby et al., 2016; Nassif and Robillard, 2017), while studies of contributor turnover at the OSS ecosystem level are more rare (Constantinou and Mens, 2017a). Moreover, no methodology exists to quantify the impact of contributor loss at the ecosystem level, not to mention statistical methods that can help identify factors that potentially influence the robustness of a software ecosystem to contributor loss.

To fill the gap, this study aims to quantify the robustness of a software ecosystem to contributor loss and perform a case study on representative OSS ecosystems. A *loss of a contributor* in an OSS ecosystem refers to a contributor who stops contributing to *all* projects in the considered ecosystem. The main reason for studying this topic is because local (project level) contributor turnover will eventually have cascading effects on the entire ecosystem due

to the sharing of code and contributors at the ecosystem level (Constantinou and Mens, 2017b; Valiev et al., 2018; Blincoe et al., 2019). Thus, it is crucial for an OSS community and its members to be aware of the ecosystem's *robustness to the loss of contributors*, i.e., the ability to remain stable (e.g., retain active projects) despite contributor loss.

OSS communities can be formed in different ways. In this study, we consider all contributors who have ever committed to the code base of any projects within an OSS ecosystem as members of an OSS community. Following the definition provided by Lungu (Lungu, 2008), we refer to *software ecosystems* as collections of software projects that are developed and evolve together in the same environment. Sample software ecosystems include libraries of programming languages (e.g., packages in npm, PyPI), distributions of operating systems (e.g., Ubuntu and Debian), and applications in mobile app stores (e.g., Google Play Store).

Inspired by a well-known model (Pocock et al., 2012) proposed to analyze the robustness of an ecological ecosystem to the loss of species, we propose the first computational framework that can quantify and analyze an OSS ecosystem's robustness to the loss of contributors. The computational framework models an OSS ecosystem during a specified period using a *contributor-project network*, where each node represents a contributor or a project in the ecosystem and edges between contributors and projects represent project contribution relationships. The framework then simulates the loss of contributors in the ecosystem by removing contributor nodes one by one from the contributor-project network. The sequence of removed nodes can be randomly generated or follow a specific mechanism, e.g., removing contributors who contribute the most in the ecosystem first. Each simulated contributor removal process results in a curve representing the cascading negative effects of contributor loss on the ecosystem in terms of project loss[1]. The area under the curve is calculated to represent the robustness of the ecosystem. To further answer questions regarding which factors influence the robustness of an OSS ecosystem to the loss of contributors, we propose a new bipartite graph generation model where we utilize parameters to control the topologies of contributor-project networks and perform robustness analysis. Our proposed generation model complements empirical studies which

[1]Project loss in one time interval means that the project receives no commits in the given time interval, i.e., the project is inactive.

mainly rely on observed empirical data.

We apply our methods to analyze the contributor loss and project extinction in two popular OSS ecosystems, i.e., the Python PyPI [2] ecosystem and the Ruby ecosystem on GitHub. Specifically, we analyze the robustness of the Ruby and PyPI ecosystems to the random loss of contributors (**RQ1**), and the robustness of synthesized OSS ecosystems with different topological properties to the random loss of contributors (**RQ2**). We demonstrate how our computational framework can be extended to analyze the impact of two specific types of contributor loss (random loss weighted by a contributor's commit number and project number) on the robustness of an OSS system (**RQ3**). To answer the above research questions, we collected 12,267,933 and 1,459,322 commits[3] from 102,447 Ruby projects and 69,311 PyPI libraries respectively, hosted on GitHub.

The main contributions of our work are as follows:

- We propose the first computational model to quantify software ecosystem robustness with respect to the random loss of contributors. We also demonstrate how our model can be adapted to analyze the impact of specific types of contributor loss at the ecosystem level.
- We propose a bipartite graph model that can generate synthetic OSS ecosystems and analyze how the topological properties of an OSS ecosystem can affect its robustness to the random loss of contributors.
- We conduct case studies on the robustness of OSS ecosystems to the loss of contributors in the PyPI and Ruby ecosystems by analyzing ten years of evolutionary data.

We provide a replication package[4] with our dataset and source code as a means to enable more in-depth and varied studies in this new research area.
Paper organization: The rest of the paper is organized as follows. Section 2 introduces the background related to the study of OSS ecosystem analysis and simulation. Section 3 introduces the design details of our proposed method for quantifying and analyzing the robustness of an OSS ecosystem to the loss of contributors. Section 4 presents our design for a case study on

[2]PyPI is the official third-party registry for Python packages
[3]Commits were collected in March 2020.
[4]`https://figshare.com/s/12130214f4824442e4d1`

the Ruby and PyPI ecosystems. Section 5 discusses the results of the case study. Section 6 discusses the limitations of this work. Finally, Section 7 concludes the paper.

2. Background

2.1. Developer Turnover in Software Projects

Developer turnover and contributor loss are critical threats to software development teams and OSS ecosystems (Schilling et al., 2012; Ehls, 2017). To help companies and OSS project maintainers retain developers and mitigate the risk introduced by developer turnover, studies have been conducted to understand why developers leave commercial or open-source software projects (Miller et al., 2019), and explore developer turnover patterns (Constantinou and Mens, 2017a; Lin et al., 2017).

Miller et al. (Miller et al., 2019) conducted a survey with developers to understand why they gave up on OSS projects. They identified three reasons: occupational, social, and technical. For instance, the most mentioned occupational reasons are: a new job that does not support OSS; a change of role/project; leaving a job where they contributed to OSS; a lack of time due to a new job; a lack of time due to an existing job; the use of OSS in school but not in professional work; too much code at work.

When analyzing the risk of contributor loss on OSS projects, a metric known as *Truck Factor (TF)* is frequently adopted to identify the concentration of knowledge in software development environments (Williams and Kessler, 2003; Zazworka et al., 2010; Torchiano et al., 2011; Cosentino et al., 2015; Mens, 2016). TF is defined as the minimum number of developers who have to be hit by a truck (or leave the team) to put the project in trouble. TF is also known as Bus Factor/Number or Lottery Factor. A low TF value means that the project's knowledge is concentrated in a few team members, and the project faces a serious risk of discontinuation should these developers leave. In contrast, a high TF value means that every developer is contributing to the project in similar terms and the risk for project discontinuation is low. TF is also considered as a metric for identifying the most important contributors in a software project (Pfeiffer, 2021). Avelino et al. (Avelino et al., 2019) found that the majority of their studied OSS projects do not survive when Truck Factor developers disengage and no other developers replace them. To overcome the challenge of knowledge concentration, i.e., having a low TF value, Etemadi et al. (Etemadi et al., 2022) proposed a

self-adaptive task assignment (SATA) approach that adaptively switches between cost-oriented and diffusion-oriented strategies over subsequent rounds of task assignments proposed.

Lin et al. (Lin et al., 2017) explored factors which can impact the will of developers to stay in OSS projects. They collected and analyzed the code commit history of five industrial OSS projects. They found that developers who start contributing to the project earlier, balance maintaining files created by themselves with files created by others, and mainly modify files tend to stay longer, i.e., leave the project later. In a similar manner, we leverage code commits as the main contribution activity in an OSS ecosystem. Ali et al. (Ali et al., 2020) apply survival analysis methods on 3,052 popular Python projects to examine the attributes that might lead to their inactivity over time. They found that projects with repositories on multiple hosting services, a timeline of publishing major releases, and a good network of developers remain healthy over time.

Constantinou et al. (Constantinou and Mens, 2017a) extended Lin et al.'s work to analyze both social and technical factors that impact developer turnover. Their work is also the first analyzing developer turnover at the ecosystem level. They found that developers who do not engage in discussions with other developers, do not have strong social and technical activity intensity, communicate or commit less frequently, and do not participate in both technical and social activities for long periods of time are more likely to abandon an ecosystem.

Overall, our study is different from most existing work on developer turnover because we conduct an ecosystem level analysis. Unlike the only ecosystem level study by Constantinou et al. (Constantinou and Mens, 2017a), we do not investigate the characteristics of developers who leave an ecosystem. Instead, we measure the impact of their loss on the ecosystem.

2.2. Robustness Analysis

The concept of robustness is well developed in engineering, referring to the maintenance of system performance when subject to unpredictable external perturbations, or when there is uncertainty about the values of internal design parameters (Carlson and Doyle, 2002). Depending on the application discipline, the definition and analysis of robustness may vary. For instance, in social sciences and economics researchers explore the stability of human societies in the face of disrupting forces (Cutter et al., 2008). In engineering, designers of communication systems conduct robustness analysis to ensure

their system can function despite occasional failures, and even under attack (Wang et al., 2014).

System robustness analysis is often conducted via analyzing the robustness of the networks representing the crucial relationships in a system. For instance, researchers in ecology have investigated the robustness of networks capturing the interactions between species (e.g., feeding relationships) in ecological ecosystems (Pocock et al., 2012; Cai and Liu, 2016). Pocock et al. (Pocock et al., 2012) propose the first and widely adopted (Pilosof et al., 2017) computational framework that can quantify the robustness of an ecological ecosystem to the loss of species and identify the keystone species in an ecosystem which are vital for ecosystem restoration planning. Cai and Liu (Cai and Liu, 2016) further extend the model to analyze the robustness of such ecosystems to the loss of multiple species in a community (rather than individual species). We believe their methodology can be applied to analyze the robustness of OSS ecosystems to the loss of contributors, as a similar "feeding" relationship exists in the OSS contribution process, i.e., the loss of contributors may lead to the extinction of a project and eventually affect the whole ecosystem that contains the project. Moreover, ecological models and theories from natural ecosystems have been adapted and adopted to understand and better explain the evolution of OSS ecosystems in previous work (Mens et al., 2014).

2.3. Simulation Study

Empirical analysis is widely adopted by software engineering research to obtain knowledge from data describing various software systems. However, the data dependency of empirical analysis can not always be satisfied. Therefore, it has been proposed to analyze the problems encountered in software engineering by using simulation studies (Stol and Fitzgerald, 2018). By constructing various simulation models, we are able to reduce the dependency of analysis on real data and facilitate a more detailed analysis about the problem by adjusting the generation parameters. Typically, simulation studies can be divided into the following two categories.

The first simulation method, known as experimental simulation (Stol and Fitzgerald, 2018), generates different artificial systems to allow for a more detailed understanding concerning the actual situation of the system. For example, the green house compared to the wild environment can fix certain system parameters (i.e. temperature), allowing the researcher to conduct a more focused analysis (Runkel and McGrath, 1972).

The second type of simulation method is known as computer simulation (Stol and Fitzgerald, 2018). This type of simulation focuses on the evaluation of the system by simulating the external changes that the system may experience (Stol and Fitzgerald, 2018). In the field of public safety, this approach is used to evaluate the safety of buildings in emergency situations (Tsigkanos et al., 2014). In the biology domain, through simulating the extinction of endangered plants and animals, we can measure the robustness of ecosystems (Pocock et al., 2012). In software engineering domain, simulation experiments also enable us to evaluate the response of systems under various situations of interest (Setamanit et al., 2007).

3. Methodology

3.1. Terminology

In the remainder of this paper, we refer to the robustness of an OSS ecosystem to the loss of contributors as the *OSS's robustness*. When capturing the contribution relationship between developers and projects within one ecosystem, we consider the submission of code commit(s) as a *contribution*, similar to existing work (Constantinou and Mens, 2017b; Bao et al., 2019).

To report and analyze the dynamics of OSS robustness, we create multiple snapshots of each target ecosystem, where each snapshot contains commits created by the contributors of the target ecosystem in a given time interval. Projects and contributors are categorized into three groups indicating their status within each time interval: the new immigrants (*NewProjects* and *Joiners*) – projects and contributors that are newly created and join the ecosystem within the current time interval; the residents (*ActiveProject* and *ActiveContr*) – projects and contributors that are active in the current intervals; the emigrants (*AbandonedProjects* and *Leavers*)– the projects and contributors that are active in the current time interval, but no longer active in the next time interval, i.e., those projects and contributors which are leaving in the current interval. We summarize the definitions of these six terms in Table 1.

3.2. Modeling Contribution in OSS Ecosystems

Before we dive into the details of our OSS ecosystem robustness model, we first introduce two important graphs we utilize to capture the relationship among projects within an OSS ecosystem and their contributors.

Table 1: Definitions of contributor and project metrics for each time interval.

Metrics	Definitions
$ActiveProjects(t)$	$\{p \mid isActive(p,t)\}$
$ObsoleteProjects(t)$	$\{p \mid isActive(p,t) \bigwedge \forall i > t, \neg isActive(p,i)\}$
$NewProjects(t)$	$\{p \mid isActive(p,t) \bigwedge \forall i < t, \neg isActive(p,i)\}$
$ActiveContr(t)$	$\{c \mid isContr(c,t)\}$
$Leavers(t)$	$\{c \mid isContr(c,t) \bigwedge \forall i > t, \neg isContr(c,i)\}$
$Joiners(t)$	$\{c \mid isContr(c,t) \bigwedge \forall i < t, \neg isContr(c,i)\}$

$isActive(p,t)$ means project p reveives commits in time interval t;
$isContr(c,t)$ means contributor c authors commits in time interval t.

Definition 3.1. Project Contributor Graph: A bipartite graph

$$G = (N_C, N_P, E) \tag{1}$$

is utilized to summarize the historical commits in the ecosystem, where N_C and N_P are two sets of nodes representing active contributors and projects in the ecosystem and E is a set of edges, wherein edge $(C_i, P_j) \in E$ if and only if project $P_j \in N_P$ receives at least one commit from contributor $C_i \in N_C$. An example G containing five projects and five contributors is shown in Figure 1.

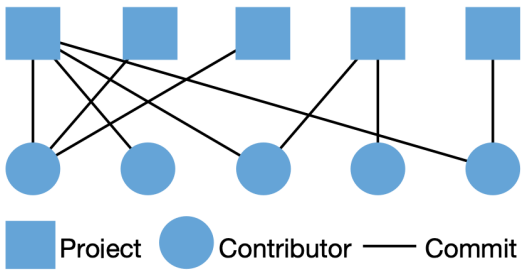


Figure 1: An example project contributor graph. The graph describes a software ecosystem consisting of five projects and five contributors.

Formulating all commit activities as one graph neglects the temporary characteristics of an evolving software ecosystem. To address this issue, we define a *temporary project contributor graph* G_t to capture the characteristics of an ecosystem in a specific period.

Definition 3.2. Temporary Project Contributor Graph: A bipartite graph

$$G_t = (N_C(t), N_P(t), E(t)) \tag{2}$$

is utilized to summarize historical commits that took place within the time interval specified by interval t. Wherein a node set $N_C(t)$ represents active contributors that make at least one commit in the time interval t, a node set $N_P(t)$ represents active projects that receive at least one commit in the time interval t and $(C_i, P_j) \in E(t)$ if and only if project $P_j \in N_P(t)$ receives at least one commit from contributor $C_i \in N_C(t)$.

3.3. Quantifying OSS Ecosystem Robustness to the Loss of Contributors

We evaluate the robustness of the OSS ecosystem to the loss of contributors **by simulating the random removal** of contributors from the contributor node-set N_C.

The reasons for analyzing the random loss of contributors are threefold. Firstly, it is impossible to predict and model which contributors will leave an ecosystem and in what order with 100% accuracy. Since we cannot model the actual contributor loss, any robustness metric is only an estimation of the actual situation. Next, in practice, we never know the underlying reasons for all contributor losses. For instance, given the sudden occurrence of COVID-19, many OSS contributors may leave their community. Without verifying the reason for each of those who left, it is not possible to conclude whether a particular set of contributors was lost because they were impacted by COVID-19, not to mention modeling such loss prior to the loss. Moreover, a mixed set of factors would contribute to the observed contributor loss at the ecosystem level, as opposed to a mere one or two factors. Thus, we simulate contributor loss in a random order mimicking a scenario for which we do not have the data (e.g., gender, age, working status, etc.) to determine who would be more likely to leave the community first. Last but not the least, robustness to random loss can be regarded as a baseline when analyzing whether an ecosystem is sensitive to particular types of contributor loss. We elaborate more on how our model can be easily adapted to analyze pre-defined types of contributor loss in Section 5.3.

Under our model, a project is considered inactive upon the loss of all its contributors. We refer to the removal of contributors as primary extinction simulation and the resulting failure of the project (i.e., the project being inactive) as secondary extinction (Pocock et al., 2012).

Definition 3.3. Primary Extinction Simulation: We simulate order of contributor removal using random permutation

$$\text{per}(w) = \{u \in \Sigma^* || w|_a = |u|_a, \text{for } a \in \Sigma\}, \tag{3}$$

where w is a string enumerating all ecosystem contributors in N_C.

The cascading failure of projects is deepened when we remove contributors in the order of removal generated by a primary extinction simulation, in which a project becomes inactive when all contributors are removed (secondary extinction) (Pocock et al., 2012). An example of a primary extinction simulation is shown in Figure 2 (left).

The robustness of the OSS ecosystem $\overline{R}$ is gathered from multiple primary extinction simulation runs, in which all contributors are removed according to the order determined by the primary extinction simulation. We remove all contributors, rather than removing a proportion of the contributors, since software ecosystem robustness is defined as the area under the curve (AUC) for project survival probability (Pocock et al., 2012; Burgos et al., 2007).

Definition 3.4. Software Ecosystem Robustness: $\overline{R}$ is calculated by averaging the area under the curve AUC of the function of secondary extinctions against primary extinction across multiple primary extinction simulation runs. The area under the curve

$$\text{AUC} = \int_0^1 f(p)dp, \tag{4}$$

with p representing the proportion of active contributors that are not removed by primary extinction simulation and $f(p)$ representing the proportion of active projects surviving secondary extinction (Pocock et al., 2012).

An illustrative example for the calculation of software ecosystem robustness $\overline{R}$ is provided in Figure 2. The distribution bar chart in Figure 2 (right) shows that the AUC values derived from 1,000 primary extinction simulations follow a bell-shaped distribution, indicating the AUC values gathered across different primary extinction simulations are spread out around the mean.

3.4. A Generative Network Model for OSS Ecosystems

We propose a generative network (i.e., bipartite graph) model with a power-law degree distribution to formulate artificial software ecosystems.

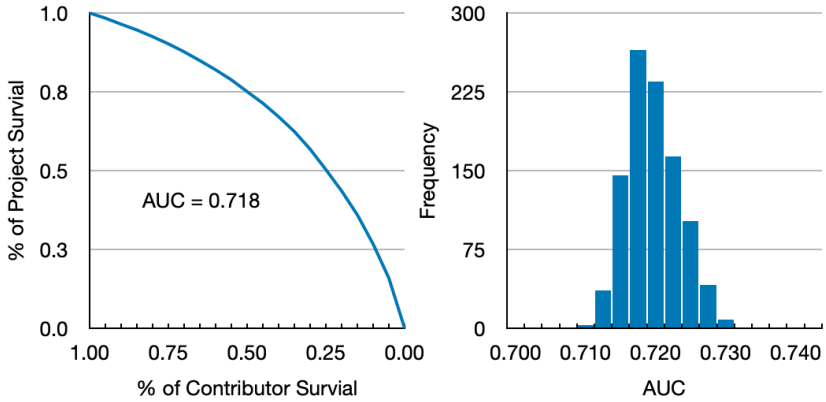


Figure 2: An example of a primary extinction simulation. The plot on the left shows the results of one primary extinction simulation, plotting the fraction of project survival (x-axis) against the fraction of contributor survival (y-axis). The plot on the right shows the distribution of the area under the curve of project survival (secondary extinction) in 1,000 primary extinction simulation runs.

The random network is generated according to a power-law degree distribution because the power-law distribution is observed in real software ecosystems. This is reflected in the distribution of the number of different contributors working on a project (Figure 3) and the distribution of the number of projects worked on by each contributor (Figure 4). By using the exponential function $f(x) = cx^{\lambda}$ to fit this distribution, we can obtain a constant c and the parameter λ reflecting the degree of the power-law distribution, the larger the value of λ the more uneven the distribution.

There are existing random graph models to generate random bipartite graphs (Guillaume and Latapy, 2006). However, these models appear to lack parameters to manipulate the simulation from aspects such as the degree of power-law distribution. Thus, we develop a novel generative bipartite graph model from scratch.

The novel generative model has one generation parameter – the curving parameter c to simulate the probabilities of ecosystem contributors participating in projects with different existing contributor numbers, making the resulting networks follow the power-law degree distribution. For any ecosystem contributor, the probability of participating in project i is determined by the following probability function:

$$p(i) = \frac{n_i^c}{\sum_P^j n_j^c} \tag{5}$$

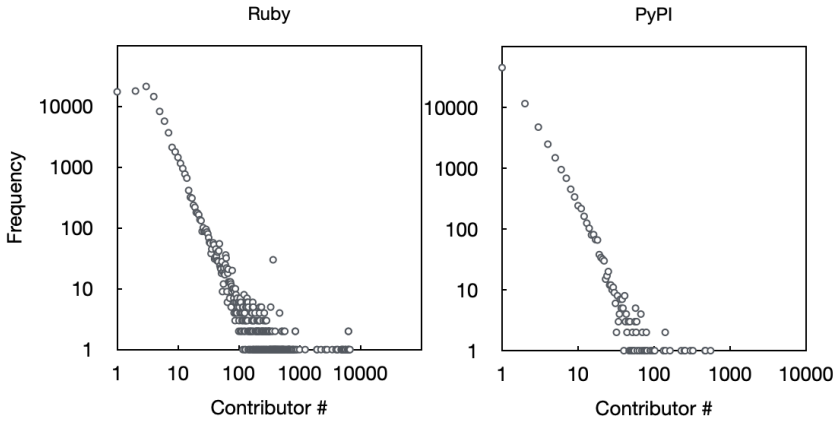


Figure 3: The distribution of projects' active contributor numbers for the Ruby and PyPI ecosystems. The x-axis represents the number of contributors and the y-axis represents the corresponding frequency of projects. Both distributions follow the power-law distribution, indicating the majority of projects have a small number of contributors.

where n_i is the number of existing contributors of project i and c is the curving parameter that will manipulate the degree of preferential participants (Barabási et al., 2016). If $c = 1$ then $p(i)$ is called linear preferential attachment. If $c > 1$ then $p(i)$ is a superlinear preferential attachment. The super-linear relationship will increase the possibility of contributors' participation in large projects and exacerbate the phenomenon of the richer-get-richer. If $c < 1$ then $p(i)$ is sublinear preferential attachment, it will mitigate the contributors' tendency to participate in big projects, mitigating the phenomenon of the richer-get-richer.

The novel generative model has one controlling parameter–the project per contributor ratio (PPC). The PPC is simply the the number of active projects included in the ecosystem divided by the number of active contributors. By leveraging PPC, we are able to control the scale of the ecosystem simulations derived from the novel generative model.

The artificial ecosystem generation process consists of three steps. The first step is to initialize the ecosystem entities based on the number of active contributors and the number of active projects specified by the PPC. In this process, each project is randomly assigned a contributor to secure its activeness in the ecosystem simulation. In the second step, we use the exponential random variable $f(x) = \lfloor e^{-x} \rfloor$ for $x \geq 0$, to determine the number of projects that the contributors work on. The function $f(x)$ generates ran-

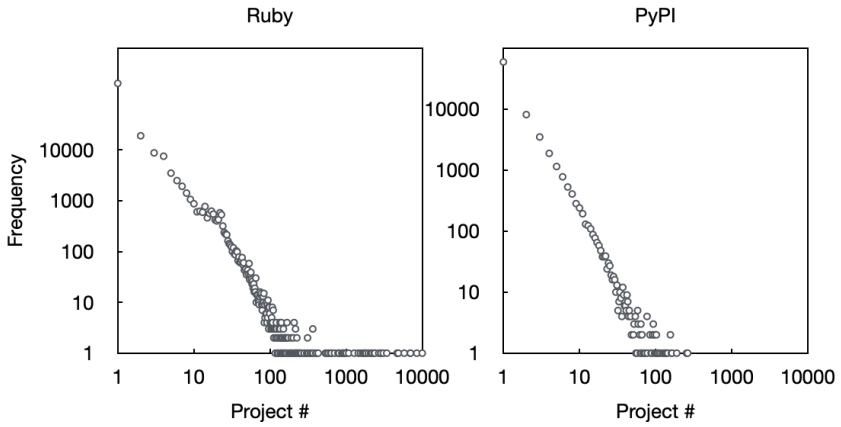


Figure 4: The distribution of contributor's active project numbers for the Ruby and PyPI ecosystem. The x-axis represents the number of contributor's active projects and the y-axis represents the frequency of such type of contributor in the ecosystem. Both distributions follow power-law, indicating the majority of contributors have a small number of contributors while the rest have a very large number of working projects.

dom numbers following the power-law distribution. This distribution stems from the empirical results we observe in real OSS ecosystems, which show the distribution of the contributor working project number follows power-law (Figure 4). In the third step, each contributor utilizes $p(i)$ to determine its working projects. This is a random process, in which the working projects are determined for each contributor in an iterative fashion.

4. Case Study Overview

4.1. Dataset

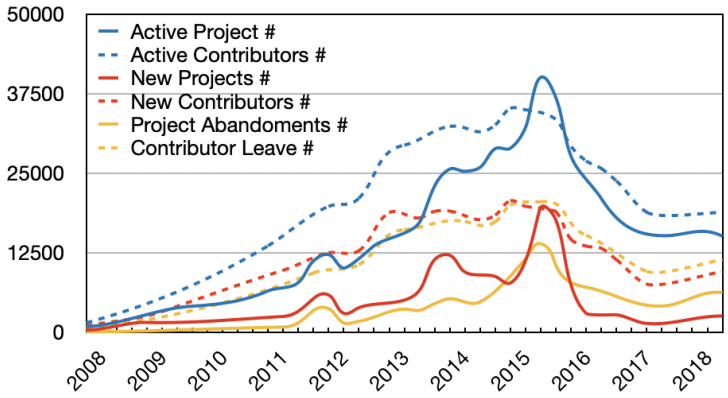
To collect projects and contributors for the Ruby ecosystem, we first obtained a list of GitHub projects having Ruby as the main programming language from GHTorrent's (Gousios, 2013) latest data dump (release 2019-06-01). The Github projects used for the PyPI ecosystem were acquired from the dataset provided by Valiev et al. (Valiev et al., 2018), which was collected as of March 2018 from the PyPI registry of Python packages. Similar to Constantinou and Mens (Constantinou and Mens, 2017b), we applied filters on the Ruby projects acquired from GitHub. More specifically, we 1) eliminated all forked projects, as many of them are created for contributing back to forked projects (Jiang et al., 2017), and 2) eliminated projects with less than three contributors and/or containing less than five commits, because we

would like to focus on non-trivial (e.g., not personal or experimental) projects in the ecosystem. In the end, we cloned a total of 102,447 Ruby projects and 69,311 PyPI projects and collected data (i.e., author, commit date, project) on $12,267,933$ commits from projects in the Ruby ecosystem and $1,459,322$ commits from projects in the PyPI ecosystem. These commits involve a total of $259,573$ contributors in the Ruby ecosystem and $77,537$ contributors in the PyPI ecosystem.

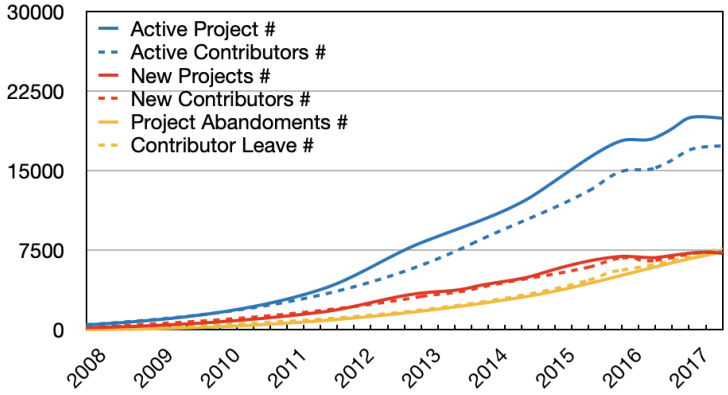
In this work, we focus on the dynamics and statistics of the two considered ecosystems in two selected time periods, i.e., 2008-01-01 to 2018-12-31 for the Ruby ecosystem, and 2008-01-01 to 2017-12-31 for the PyPI ecosystem. We do not study the Ruby ecosystem after 2019 because our projects are selected based on the 2019-06-01 GHTorrent data dump, i.e., we do not have all the projects created after 2019-06-01. Moreover, we need a period long enough to ensure the status of projects in each considered period is correct. For instance, determining whether a project was abandoned in May of 2019 relies on observation after 2019-06-01, which is not included in our dataset. For similar reasons, we only report the statistics of the PyPI ecosystem before 2018. All commits in the two target time periods are split into time intervals of six-months, where two adjacent time intervals have an overlap of three months. For instance, the first four time intervals for the Ruby ecosystem capture the ecosystem's statistics based on commits submitted between 2008-01-01 to 2008-06-31, 2008-04-01 to 2008-09-31, 2008-07-01 to 2008-12-31, and 2008-10-01 to 2009-03-31, respectively. The overlapping intervals allow us to mitigate the influence of commits submitted near the boundary of time intervals. Based on the above setup, we created 43 time intervals for the Ruby ecosystem (three in each year), and 39 time intervals for the PyPI ecosystem.

4.2. Prevalence of Contributor Loss and Project Extinction in the Ruby and PyPI Ecosystems

Figure 5a and Figure 5b show the dynamics of basic project and contributor statistics for the Ruby and PyPI ecosystems, based on our collected commits. We observe a clear rise-and-drop pattern in terms of active projects and contributors in the evolution of the Ruby ecosystem. On the other hand, the PyPI ecosystem grows rapidly before 2016 and becomes stable afterwards. The results also show that in both ecosystems there is a large proportion of active contributors and projects leaving the ecosystem in each time interval.



(a) The evolution of the Ruby ecosystem.



(b) The evolution of the PyPI ecosystem.

Figure 5: The evolution of the Ruby and PyPI ecosystems. The x-axis represents time intervals of six months (time intervals are displayed every three intervals), and the y-axis represents quantities of the displayed metrics. The legend of the figure outlines six ecosystem metrics.

Figure 6 shows how contributor turnover ratio and project extinction ratio change over time in the Ruby and PyPI ecosystems. We observe that contributor turnover happens frequently with a 20%-60% turnover ratio in both ecosystems. We also observe that the two target ecosystems respond to the loss of their contributors differently in terms of project extinction. We observe that the Ruby ecosystem has a much higher contributor turnover rate (ref. solid lines in Figure 6) but a similar project abandonment rate (ref. dashed lines in Figure 6) compared to the PyPI ecosystem.

We also apply the robustness model defined in Section 3 on the 43 time

intervals of the Ruby ecosystem and 39 time intervals of the PyPI ecosystem to capture the evolution of the two ecosystems over ten years. The commits in each time interval are the source for creating the temporary project contributor graph.

Our robustness model requires two inputs, i.e., an order of contributor removal and the number of runs of the primary extinction simulation. The robustness value is the averaged area under the curve generated by repeating the primary extinction simulation $1,000$ times. The number $1,000$ is chosen following previous studies (Dunne et al., 2002), as well as our observation that the robustness value becomes stable after $1,000$ runs of the primary extinction simulation.

The red lines in Figure 11 describe the ecosystem robustness value under random contributor removal for the Ruby and PyPI ecosystems. The value characterizes the robustness of the ecosystem to the random loss of contributors. The results indicate that the Ruby ecosystem has a higher robustness compared to PyPI. Ruby's higher robustness value explains why, although the fraction of contributor loss is always higher in the Ruby ecosystem than in PyPI, Ruby can maintain the same level of project extinction as PyPI (Figure 6).

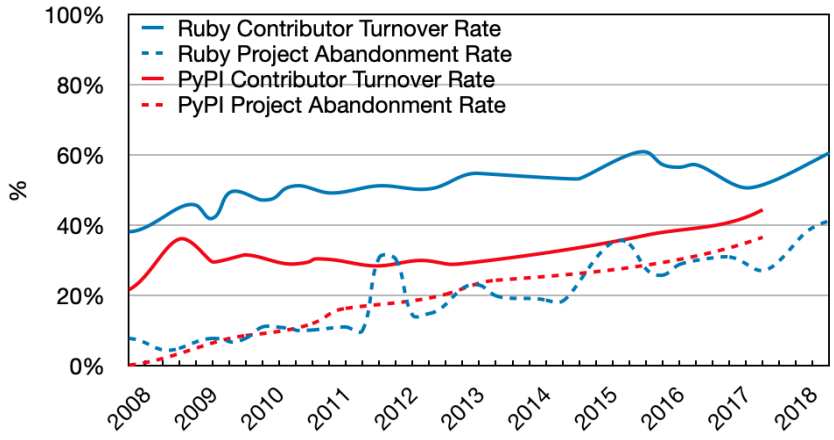


Figure 6: The evolution of ecosystem-level contributor turnover/project abandonment rate. The x-axis represents time intervals of six months (time intervals are displayed every two intervals), and the y-axis represents ratios which are calculated as follows. For a specific time interval t, the contributor turnover rate is defined as $Leavers(t)$ divided by $ActiveContr(t)$ and the project turnover rate is defined as $ObsoleteProjects(t)$ divided by $ActiveProjects(t)$.

The above analysis shows that contributor turnover and project abandonment frequently happen in the Ruby and PyPI ecosystems. Moreover, two considered ecosystems show different robustness to contributor loss in terms of project extinction.

4.3. Research Questions

To demonstrate the usage of the proposed analytical framework characterizing the robustness of OSS ecosystems, we propose the following three research questions to guide our case study on the Ruby and PyPI ecosystems.

RQ1: How robust are Ruby and PyPI ecosystems to the random loss of contributors? The loss of contributors is a widespread phenomenon in OSS ecosystems (ref. Figure 6). This phenomenon usually accompanies the fraction of ecosystem projects that are at risk of being left unmaintained. In this RQ, by leveraging our proposed computational framework in Section 3, we aim to answer how the robustness of two target OSS ecosystems changes over time. We also evaluate the impact of excluding short-term contributors on the robustness results.

RQ2: How will the topological properties of an OSS ecosystem affect its robustness to the random loss of contributors? A subsequent question of RQ1 is to investigate which factors can influence ecosystem robustness. As mentioned in Section 2 and Section 3, a network's overall topological characteristics determine how it reacts to contributor loss. Thus, in this RQ, we further examine the extent to which OSS ecosystem robustness can be affected by these properties (i.e., PPC and preferential attachment degree) via simulation studies.

RQ3: How robust are Ruby and PyPI ecosystems to specific types of contributor loss? The above experiments assume the departure of contributors is purely random, whereas this assumption is not consistent with reality. Existing work has been done demonstrating that contributors with lower contribution intensity may leave the ecosystem earlier. Therefore, in this RQ, we extend the random process of contributor removal to reflect some of the contributor's characteristics (e.g., number of commits and number of active projects). This extension allows us to quantify the association between each characteristic of the contributor and the ecosystem robustness, allowing us to distinguish critical contributor characteristics from others.

5. Results and Analysis

5.1. RQ1: How robust are Ruby and PyPI ecosystems to the random loss of contributors?

Methods: We apply the robustness model defined in Section 3 on the 43 time intervals of the Ruby ecosystem and 39 time intervals of the PyPI ecosystem to capture the evolution of the two ecosystems over ten years. The commits in each time interval are the source for creating the temporary project contributor graph.

Our robustness model requires two inputs, i.e., a mechanism determining the order of removal and the number of runs of the primary extinction simulation. In this RQ, the order of contributor removal for each primary extinction simulation is formed by randomly permutating all active contributors in the ecosystem. We then calculate the robustness of each OSS ecosystem based on 1,000 runs of the primary extinction simulation. The number 1,000 is picked following previous studies (Dunne et al., 2002), as well as our observation that the robustness value follows a normal distribution after 1,000 runs of the primary extinction simulation.

Results: We present the evolution of the software ecosystem robustness of the Ruby and PyPI ecosystems in Figure 7 (Original). Figure 7 indicates that the robustness of the two ecosystems evolves over time. Both ecosystems saw a decrease in robustness from 2013 onward. We also observe that the Ruby ecosystem has larger robustness values than the PyPI ecosystem across all considered time intervals, i.e., the PyPI ecosystem is more fragile to the loss of its contributors (in terms of retaining active projects). This result explains why the PyPI ecosystem has a much higher contributor turnover rate than the Ruby ecosystem, but has a similar project abandonment rate (Figure 6).

When applying the proposed OSS robustness computational framework to the PyPI ecosystem and the Ruby ecosystem, we include short-term contributors and projects by default. However, in practice, researchers and OSS maintainers may have different preferences in keeping or discarding short-term contributors and projects. Thus, to investigate the potential impact of removing short-term contributors and projects, we generate four more datasets per ecosystem, each containing commits after the removal of short-term contributors or short-term projects. The results are shown in Figure 7 using dotted lines and dashed lines. We define short-term contributors/projects based on the time duration between their first and last observed commit in our dataset. Our experimental results show that removing short-

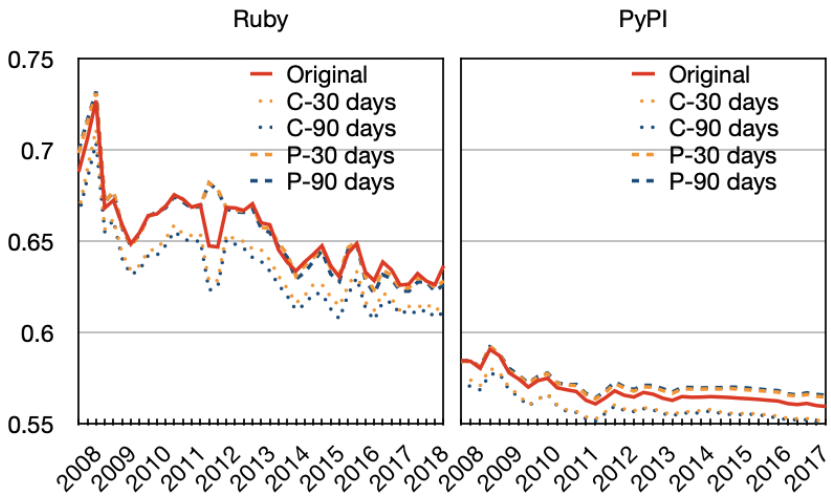


Figure 7: The evolution of the robustness value for the Ruby and PyPI ecosystems. The yellow dotted line (C-30) represents ecosystem robustness when considering only contributors with an active time greater than 30 days, the blue dotted line (C-90) represents ecosystem robustness when considering only contributors with an active time greater than 90 days. Similarly, the yellow dashed line (P-30) represents ecosystem robustness when considering only projects with an active time greater than 30 days, and the blue dashed line (P-90) represents ecosystem robustness when considering only projects with an active time greater than 90 days.

term contributors and projects does not affect the evolution trends of the two ecosystems' robustness values.

> Summary: The robustness of the Ruby and PyPI ecosystems to the loss of contributors is different. The project abandonment rate reflects that a robust ecosystem can afford to lose more contributors than a less robust ecosystem. Our results indicate that the proposed simulation-based metric can reflect the difference in robustness effectively.

5.2. RQ2: How will the topological properties of an OSS ecosystem affect its robustness to the random loss of contributors?

Methods: Determining the impact of network properties on ecosystem robustness requires obtaining data with various such properties. However, this requirement is not always possible. We therefore generate synthetic data, thereby reducing the study's dependence on real data. As opposed to em-

pirical analysis, this type of research does not require ecosystem data. We first need insights into how ecosystems are formed. Based on these insights, synthetic ecosystems can be produced using the generative process defined in Section 3.4.

We set up two ecosystem simulations to verify how different ecosystem properties affect ecosystem robustness. Experiment one verifies the relationship between PPC and the robustness of the ecosystem, we fix the number of active projects to $10,000$. By setting the number of active contributors to 6,000, 7,000, 8,000, 9,000, 10,000, 11,000, 12,000, 13,000, and 14,000, we obtain nine different PPC parameters $\{\frac{10,000}{6,000}, \frac{10,000}{7,000}, \frac{10,000}{8,000}, ..., \frac{10,000}{14,000}\}$. At the same time, we look at five curving parameter settings $\{0.6, 0.8, 1.0, 1.2, 1.5\}$. The Cartesian product of PPC and curving parameter yields $9 \times 5 = 45$ parameter combinations in total. For each parameter combination, we produce 32 artificial ecosystems and measure the corresponding robustness values using 1000 primary extinctions.

Experiment Two verifies the relationship between the degree of preferential project participation of contributors and the robustness of the ecosystem. We adjust the curving parameter $\{0.1, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0\}$ to manipulate the power-law distribution of preferential project participation. At the same time, we repeat the process for each of three PPC values, obtained from a project number of 10,000 and contributor numbers $\{8000, 10000, 12000\}$, to confirm whether the correlation is consistent under different PPC conditions.
Results: The results of the two experiments indicate the increase of either PPC (Figure 8-A) or Curving parameters (Figure 8-B) decreases the robustness of the OSS ecosystem against the loss of contributors. How do these two factors work to affect ecosystem robustness?

The results in Figure 8-C suggest the PPC parameter is positvely correlated with the power-law distribution coefficient λ. Considering the increase of λ leads to the decreases of the overall ecosystem robustness (Figure 8-D), the rise of PPC could also lead to the decreases of the overall ecosystem robustness. This result provides a new insight into how the growth of ecosystem scale could impact ecosystem robustness. A dramatic increase in the number of projects in the ecosystem leads to a significant increase in the average number of projects that contributors need to maintain. Compounded by the fact that contributors prefer larger projects when selecting projects, smaller projects are more vulnerable to the increase of PPC and eventually compromise overall ecosystem robustness.

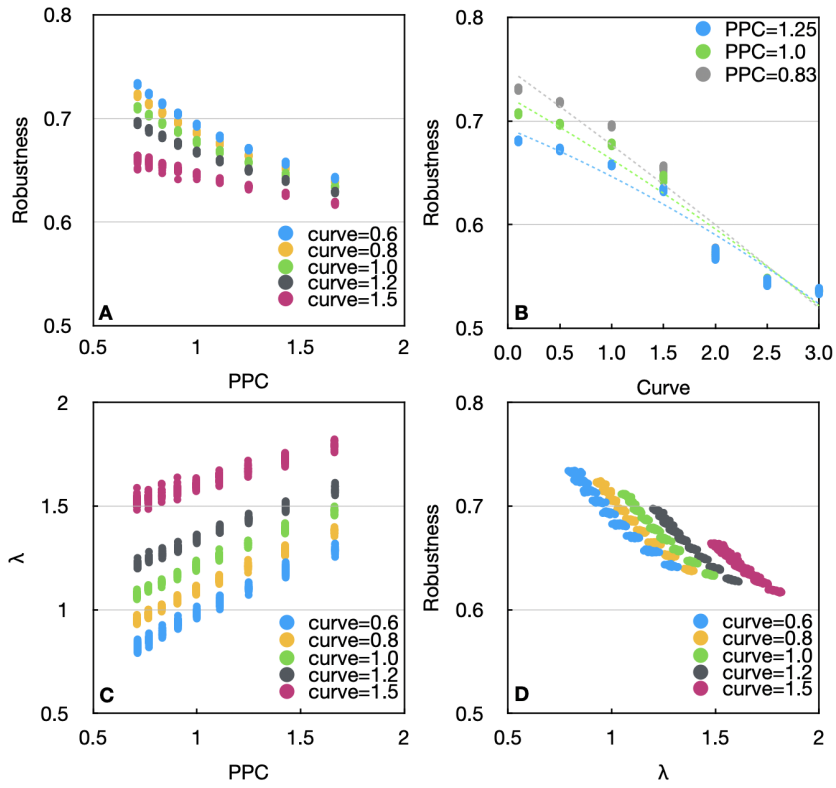


Figure 8: Results of the software simulation study. A) Ecosystem averaged workload, indicated by PPC, mitigates ecosystem robustness. B) Curving parameter compromises ecosystem robustness. C) PPC and the curving parameter promote the λ coefficient. D) The increase of the λ coefficient reduces the robustness of the system

Further ecosystem simulations are performed to verify the impact of ecosystem growth in scale, wherein the number of projects and contributors grow at the same time. Instead of growing ecosystem scale while changing PPC, this experiment grows the ecosystem while maintaining the PPC parameter. The results in Figure 9 show the averaged robustness across 32 simulation runs is consistent for ecosystems at different scales, and the consistency is not disrupted by the tuning of the curving parameter.

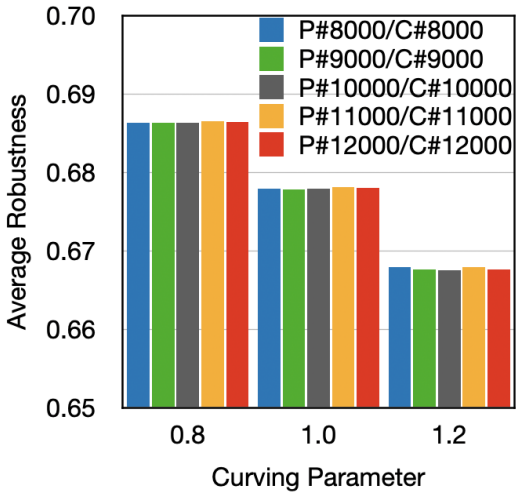


Figure 9: The scale of the OSS ecosystem will not affect robustness. For each curving parameter in $\{0.8, 1.0, 1.2\}$ (represented in x-axis), we simulate five ecosystems at different scales (differentiated by color). In the figure, $P\#$ represents the number of projects and $C\#$ represents the number of contributors. The PPC values for all five ecosystems equal to 1, indicating the averaged workload of all ecosystem simulations are identical. The result shows the impact of the average workload of ecosystem contributors is not varied across ecosystems of different scales.

Summary: By using ecosystem simulation studies, we confirm two network characteristics that jeopardize ecosystem robustness: the increase of preferential project participation and the growth of contributors' average project number. The increased preference of contributors to participate in large projects is the fundamental reason for reducing ecosystem robustness. An increase in contributors' average project number negatively impacts ecosystem robustness through this fundamental cause.

5.3. RQ3: How robust are Ruby and PyPI ecosystems to specific types of contributor loss?

Methods: The role of randomness in OSS environments is undoubtedly significant, but it's not strictly random. In this research question, we analyze the robustness of OSS ecosystems to specific types of contributor loss.

We evaluate the characteristics of contributors who leave and those who stay using statistical techniques. We first categorize contributors into two

groups: those who exit the community ($Leavers(t)$) and those who remain inside the community ($ActiveContr(t) - Leavers(t)$) during the next time interval. We then evaluate how "leavers" and "stayers" vary in the numbers of projects and commits they make to the ecosystem by running a Mann-Whitney test. The results shown in Table 2 indicate the numbers of projects and commits in the "leaver" group are both significantly lower than those in the "stayer" group ($p < 0.001$, Mann-Whitney U test).

Table 2: Mann-Whitney test for the Ruby and PyPI ecosystems from 2017-7 to 2017-12. For the Mann-Whitney test, effect size is given by the rank biserial correlation. For all tests, the alternative hypothesis specifies that group $ActiveContr(t) - Leavers(t)$ is greater than group $Leavers(t)$.

Ecosystem	Variable	p	Hodges-Lehmann Estimate	Rank-Biserial Correlation
PyPI	Project#	$< .001$	5.330e-5	0.240
	Commit#	$< .001$	2.000	0.364
Ruby	Project#	$< .001$	7.276e-5	0.253
	Commit#	$< .001$	3.000	0.396

Motivated by the above results, we propose two *weighted-random contributor loss generation* methods to allow our robustness computational framework to consider specific characteristics of contributors while performing primary extinction simulations. When we generate a random contributor removal order, we manipulate the generation of the random sequence by setting weights for each contributor.

Here, we investigate the impact of two contributor characteristics, including the number of commits and the number of projects. We use these two measures as contributor weights in order to generate random sequences while considering weights. Individuals with higher (or lower) weights are given a higher probability of being removed first. We summarize all mentioned contributor removal methods, including weighted-random contributor loss generation methods and random contributor removal, in Table 3. For each characteristic, we generate two modes of weighted random removal. The first mode is high-yield prioritized, where contributors with higher weights have a higher probability of being removed first, and the second mode is low-yield prioritized, where contributors with lower weights have a higher likelihood of being removed first.

Figure 10 compares the robustness of the Ruby and PyPI ecosystems un-

Table 3: Five contributor removal generative methods.

Index	Characteristic	Mode	Abbreviation
1	Project Number	High-yield prioritized	P# H to L
2	Project Number	Low-yield prioritized	P# L to H
3	Commit Number	High-yield prioritized	C# H to L
4	Commit Number	Low-yield prioritized	L# L to H
5	-	Random	R

der five contributor loss mechanisms using one snapshot ($2017-07-01$ to $2017-12-31$). The results indicate that secondary extinctions of the two ecosystems vary among different types of contributor removal sequences when contributors are removed according to a weighted-random process. Specifically, we find (1) both the Ruby and PyPI ecosystems are more robust to the random removal of contributors than to the active contributor prioritized removal method, (2) removing the least active contributors first results in minimal secondary extinctions, which are even lower than those seen with random contributor removals. We replicate the analysis over all time intervals. As shown in Figure 11, the divergence under different contributor removal strategies is consistent with the result derived from a single time interval.

We observe that applying different modes of removal with regard to the same characteristic produces different robustness values, which we believe is a reflection of the correlation between the selected characteristic and ecosystem robustness. Random contributor removal provides a baseline for robustness analysis due to its purely random nature. If weighted contributor removal generated by a characteristic yields a similar robustness value to this baseline, this characteristic cannot be used to discover contributors with a higher impact on robustness because the weights generated by the characteristic are no different from complete randomness. Conversely, deviation from the baseline indicates that the characteristic is relevant to ecosystem robustness because the weights generated by the characteristic can reduce ecosystem robustness values by prioritizing the removal of high-value contributors. The experimental results in Figures 10 and 11 suggest that the two characteristics we investigate produce large deviations from baseline robustness in both ecosystems, implying that a high commit number and a high project number are common characteristics of key contributors in maintaining ecosystem robustness. In addition to the two features mentioned above, this robustness

evaluation approach based on weighted random removal has the potential to be applied to analyze other characteristics in order to verify whether a certain contributor characteristic is associated with the robustness of the ecosystem to contributor removal.

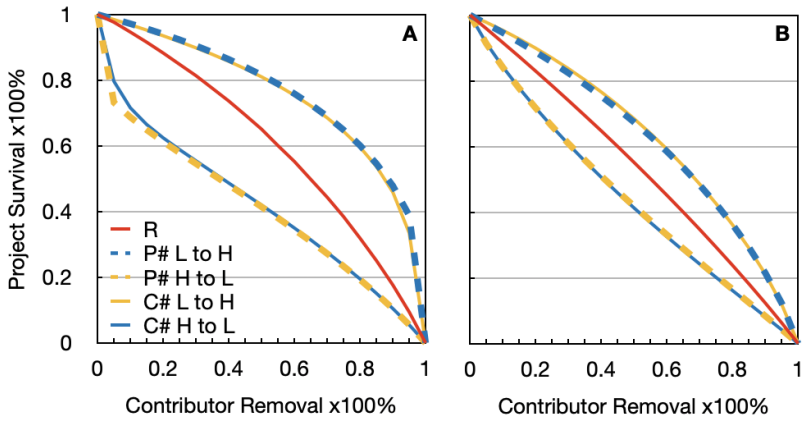


Figure 10: Results of primary extinction simulations under five types of contributor removal mechanisms for the A) Ruby and B) PiPy ecosystems, including random order removal (red solid line), project number weighted random order removal (blue and yellow dashed lines), and commit number weighted random order removal (blue and yellow solid lines). "H to L" indicates that contributors with a higher commit/project number are more likely to be removed first and "L to H" indicates contributors with a lower commit/project number are more likely to be removed first.

We define two types of contributor removal methods, including random contributor removal and weighted random contributor removal with regard to contributor characteristics. Which method best represents actual ecosystem robustness? We find that both ecosystems are less robust to the prioritized removal of high-yield contributors than to the prioritized removal of low-yield contributors. Since random contributor removal removes high- and low-yield contributors in equal proportions, the robustness of the ecosystem to random contributor removal falls somewhere between the robustness produced by the weighted-random contributor removal methods. The statistical test results recorded in Table 2 indicate that the actual workload of contributors leaving the ecosystem, including the number of projects or commits, is significantly lower than that of contributors staying in the ecosystem. Therefore, we can conclude that the actual robustness is higher than the robustness value generated by the random contributor removal model and lower than the robustness

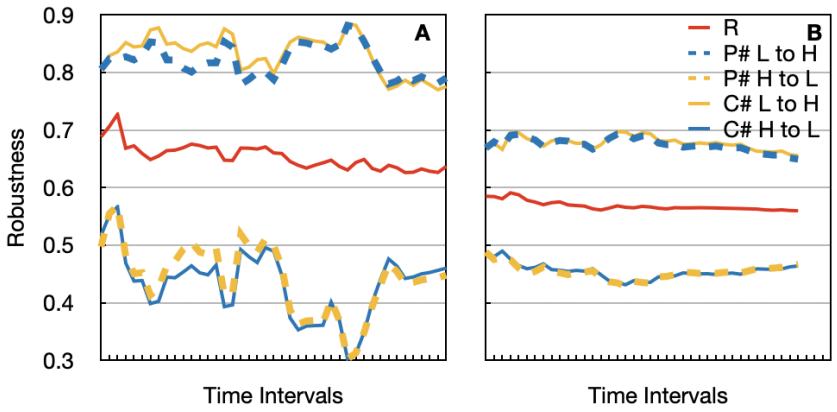


Figure 11: The evolution of the robustness values of five contributor loss mechanisms. Five contributor removal mechanisms are investigated for the A) Ruby and B) PyPI ecosystems, including random order removal (red solid line), project number weighted random order removal (blue and yellow dashed lines), and commit number weighted random order removal (blue and yellow solid lines).

generated by the priority removal of low-productivity contributors.

For practical analysis, we suggest using the ensemble of system robustness based on multiple contributor removal methods in conjunction with ecosystem contributor loss characteristics. For instance, we estimate the robustness of the Ruby and PyPI ecosystems to be between the robustness produced by the random and low-productivity prioritized contributor removal methods, because the workload of contributors who actually leave the ecosystem is significantly less than that of other contributors. Suppose the workload of the contributor who actually leaves the ecosystem is significantly larger than that of the other contributors. In this case, we suggest it is more sensible to estimate the robustness of the ecosystem to fall between the robustness produced by the random and the high-yield contributor prioritized contributor removal methods.

Summary: Weighted random contributor removal methods allow us to measure the robustness of the ecosystem under different contributor characteristics. We test two ecosystem contributor characteristics, namely the number of projects and the number of commits made. The results suggest that ecosystem robustness is very sensitive to the loss of contributors with a high number of projects or a high number of commits.

6. Threats to Validity

Internal Threats. One of the main internal threats to validity arises from splitting the commits data into six-month intervals. Although this setup has been previously used in evolution studies (Miller et al., 2019), varying the time interval duration might impact the findings reported in RQs. To mitigate this threat, we set a three-month overlapping period between two consecutive time intervals. In future work, we would investigate the potential impact of this setup on our findings.
External Threats. Our findings are based on the exploratory study of two ecosystems. Thus, they might not represent characteristics of robustness to contributor loss in other ecosystems. We plan to apply our analysis methodology to other ecosystems in future work. Nevertheless, we believe the network simulation experiments in RQ2 are general and the proposed methodology can be easily extended or adapted for a similar purpose. Following the perils of mining GitHub data outlined in (Kalliamvakou et al., 2016), we also applied several filters to filter out projects based on their commit number and contributor number. Although we applied several manually defined preprocessing steps on raw data, these metrics are often considered to remove random and trivial projects when analyzing OSS projects on GitHub (Constantinou and Mens, 2017b).

7. Conclusion

In this paper, we introduce a computational framework that can quantify an OSS ecosystem's robustness to the loss of contributors. We also design a novel approach to synthesize bipartite graphs that mimic the relationship between contributors and projects in an OSS ecosystem.

We conduct an exploratory study on two popular OSS ecosystems, the Ruby ecosystem and the PyPI ecosystem, based on a ten year span of commit history extracted from 102,447 Ruby projects and 69,311 PyPI projects hosted on GitHub. We show that (1) contributor turnover and project abandonment happen frequently in both ecosystems, (2) the Ruby ecosystem is more robust than the PyPI ecosystem to the random loss of contributors, (3) the robustness of the two ecosystems has evolved over time, (4) the preference of developers to contribute to large projects in an OSS ecosystem negatively affects the robustness of the ecosystem, and (5) both ecosystems are less robust to the loss of active contributors who either intensively or extensively contribute to the ecosystem than the random loss of contributors.

In the future, we would like to extend our methodology to analyze more ecosystems and investigate the impact of other factors on OSS ecosystem maintenance and resilience when subject to different types of risk.

References

Ali, R.H., Parlett-Pelleriti, C., Linstead, E., 2020. Cheating death: A statistical survival analysis of publicly available python projects, in: Proceedings of the 17th International Conference on Mining Software Repositories, pp. 6–10.

Apache Log4j, 2021. Apache Log4j Security Vulnerabilities. `https://logging.apache.org/log4j/2.x/security.html?spm=a2c4g.11174386.n2.4.56b74c07jouc89`. Online; accessed 20 Feb 2022.

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

Bao, L., Xia, X., Lo, D., Murphy, G.C., 2019. A large scale study of long-time contributor prediction for github projects. IEEE Transactions on Software Engineering .

Barabási, A.L., et al., 2016. Network science. Cambridge university press.

Blincoe, K., Harrison, F., Kaur, N., Damian, D., 2019. Reference coupling: An exploration of inter-project technical dependencies and their charac-

teristics within large software ecosystems. Information and Software Technology 110, 174–189.

Burgos, E., Ceva, H., Perazzo, R.P., Devoto, M., Medan, D., Zimmermann, M., Delbue, A.M., 2007. Why nestedness in mutualistic networks? Journal of theoretical biology 249, 307–313.

Cai, Q., Liu, J., 2016. The robustness of ecosystems to the species loss of community. Scientific reports 6, 35904.

Carlson, J.M., Doyle, J., 2002. Complexity and robustness. Proceedings of the national academy of sciences 99, 2538–2545.

Coelho, J., Valente, M.T., 2017. Why modern open source projects fail, in: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, pp. 186–196.

Constantinou, E., Mens, T., 2017a. An empirical comparison of developer retention in the rubygems and npm software ecosystems. Innovations in Systems and Software Engineering 13, 101–115.

Constantinou, E., Mens, T., 2017b. Socio-technical evolution of the ruby ecosystem in github, in: 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), IEEE. pp. 34–44.

Cosentino, V., Izquierdo, J.L.C., Cabot, J., 2015. Assessing the bus factor of git repositories, in: 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), IEEE. pp. 499–503.

Cutter, S.L., Barnes, L., Berry, M., Burton, C., Evans, E., Tate, E., Webb, J., 2008. A place-based model for understanding community resilience to natural disasters. Global environmental change 18, 598–606.

Dunne, J.A., Williams, R.J., Martinez, N.D., 2002. Network structure and biodiversity loss in food webs: robustness increases with connectance. Ecology letters 5, 558–567.

Ehls, D., 2017. Open source project collapse–sources and patterns of failure .

Etemadi, V., Bushehrian, O., Robles, G., 2022. Task assignment to counter the effect of developer turnover in software maintenance: A knowledge diffusion model. Information and Software Technology 143, 106786.

Foucault, M., Palyart, M., Blanc, X., Murphy, G.C., Falleri, J.R., 2015. Impact of developer turnover on quality in open-source software, in: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, pp. 829–841.

Gousios, G., 2013. The ghtorrent dataset and tool suite, in: Proceedings of the 10th Working Conference on Mining Software Repositories, IEEE Press, Piscataway, NJ, USA. pp. 233–236. URL: http://dl.acm.org/citation.cfm?id=2487085.2487132.

Guillaume, J.L., Latapy, M., 2006. Bipartite graphs as models of complex networks. Physica A: Statistical Mechanics and its Applications 371, 795–813.

Izquierdo-Cortazar, D., Robles, G., Ortega, F., Gonzalez-Barahona, J.M., 2009. Using software archaeology to measure knowledge loss in software projects due to developer turnover, in: 2009 42nd Hawaii International Conference on System Sciences, IEEE. pp. 1–10.

Jiang, J., Lo, D., He, J., Xia, X., Kochhar, P.S., Zhang, L., 2017. Why and how developers fork what from whom in github. Empirical Software Engineering 22, 547–578.

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D.M., Damian, D., 2016. An in-depth study of the promises and perils of mining github. Empirical Software Engineering 21, 2035–2071.

Lin, B., Robles, G., Serebrenik, A., 2017. Developer turnover in global, industrial open source projects: Insights from applying survival analysis, in: 2017 IEEE 12th International Conference on Global Software Engineering (ICGSE), IEEE. pp. 66–75.

Lungu, M., 2008. Towards reverse engineering software ecosystems, in: 2008 IEEE International Conference on Software Maintenance, IEEE. pp. 428–431.

Marsan, J., Templier, M., Marois, P., Adams, B., Carillo, K., Mopenza, G.L., 2018. Toward solving social and technical problems in open source software ecosystems: using cause-and-effect analysis to disentangle the causes of complex problems. IEEE Software 36, 34–41.

Mens, T., 2016. An ecosystemic and socio-technical view on software maintenance and evolution, in: 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE. pp. 1–8.

Mens, T., Claes, M., Grosjean, P., Serebrenik, A., 2014. Studying evolving software ecosystems based on ecological models, in: Evolving software systems. Springer, pp. 297–326.

Miller, C., Widder, D.G., Kästner, C., Vasilescu, B., 2019. Why do people give up flossing? a study of contributor disengagement in open source, in: IFIP International Conference on Open Source Systems, Springer. pp. 116–129.

Nassif, M., Robillard, M.P., 2017. Revisiting turnover-induced knowledge loss in software projects, in: 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE. pp. 261–272.

Pfeiffer, R.H., 2021. Identifying critical projects via pagerank and truck factor, in: 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), IEEE. pp. 41–45.

Pilosof, S., Porter, M.A., Pascual, M., Kéfi, S., 2017. The multilayer nature of ecological networks. Nature Ecology & Evolution 1, 1–9.

Pocock, M.J., Evans, D.M., Memmott, J., 2012. The robustness and restoration of a network of ecological networks. Science 335, 973–977.

Rigby, P.C., Zhu, Y.C., Donadelli, S.M., Mockus, A., 2016. Quantifying and mitigating turnover-induced knowledge loss: case studies of chrome and a project at avaya, in: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), IEEE. pp. 1006–1016.

Runkel, P.J., McGrath, J.E., 1972. Research on human behavior: A systematic guide to method. Holt, Rinehart & Winston of Canada Ltd.

Schilling, A., Laumer, S., Weitzel, T., 2012. Who will remain? an evaluation of actual person-job and person-team fit to predict developer retention in floss projects, in: 2012 45th Hawaii International Conference on System Sciences, IEEE. pp. 3446–3455.

Schneider, Z., 2018. event-stream vulnerability explained. `https://schneider.dev/blog/event-stream-vulnerability-explained/`. [Online; accessed 19-July-2020].

Setamanit, S.o., Wakeland, W., Raffo, D., 2007. Using simulation to evaluate global software development task allocation strategies. Software Process: Improvement and Practice 12, 491–503.

Stol, K.J., Fitzgerald, B., 2018. The abc of software engineering research. ACM Transactions on Software Engineering and Methodology (TOSEM) 27, 1–51.

Torchiano, M., Ricca, F., Marchetto, A., 2011. Is my project's truck factor low? theoretical and empirical considerations about the truck factor threshold, in: Proceedings of the 2nd international workshop on emerging trends in software metrics, pp. 12–18.

Tsigkanos, C., Pasquale, L., Menghi, C., Ghezzi, C., Nuseibeh, B., 2014. Engineering topology aware adaptive security: Preventing requirements violations at runtime, in: 2014 IEEE 22nd International Requirements Engineering Conference (RE), IEEE. pp. 203–212.

Valiev, M., Vasilescu, B., Herbsleb, J., 2018. Ecosystem-level determinants of sustained activity in open-source projects: A case study of the pypi ecosystem, in: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 644–655.

Wang, J., Jiang, C., Qian, J., 2014. Robustness of internet under targeted attack: a cascading failure perspective. Journal of Network and Computer Applications 40, 97–104.

Williams, L., Kessler, R.R., 2003. Pair programming illuminated. Addison-Wesley Professional.

Zazworka, N., Stapel, K., Knauss, E., Shull, F., Basili, V.R., Schneider, K., 2010. Are developers complying with the process: an xp study, in: Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 1–10.

Zemlin, J., 2017. If you can't measure it, you can't improve it: Chaoss project creates tools to analyze software development and measure open source community health.