Socio-Technical Evolution of the Ruby Ecosystem in GitHub

Eleni Constantinou
Software Engineering Lab,
COMPLEXYS Research Institute
University of Mons
Mons, 7000, Belgium
eleni.constantinou@umons.ac.be

Tom Mens
Software Engineering Lab,
COMPLEXYS Research Institute
University of Mons
Mons, 7000, Belgium
tom.mens@umons.ac.be

***Abstract*—The evolution dynamics of a software ecosystem depend on the activity of the developer community contributing to projects within it. Both social and technical changes affect an ecosystem's evolution and the research community has been investigating the impact of these modifications over the last few years. Existing studies mainly focus on temporary modifications, often ignoring the effect of permanent changes on the software ecosystem. We present an empirical study of the magnitude and effect of permanent modifications in both the social and technical parts of a software ecosystem. More precisely, we measure permanent changes with regard to the ecosystem's projects, contributors and source code files and present our findings concerning the effect of these modifications. We study the Ruby ecosystem in GitHub over a nine-year period by carrying out a socio-technical analysis of the co-evolution of a large number of base projects and their forks. This analysis involves both the source code developed for these projects as well as the developers having contributed to them. We discuss our findings with respect to the ecosystem evolution according to three different viewpoints: (1) the base projects, (2) the forks and (3) the entire ecosystem containing both the base projects and forks. Our findings show an increased growth in both the technical and social aspects of the Ruby ecosystem until early 2014, followed by an increased contributor and project abandonment rate. We show the effect of permanent modifications in the ecosystem evolution and provide preliminary evidence of contributors migrating to other ecosystems when leaving the Ruby ecosystem.**

***Index Terms*—Ruby; Software Ecosystem; GitHub; Software Evolution; Socio-Technical Analysis**

I. Introduction

Most open source projects today are no longer developed in isolation [1], but co-evolve in software ecosystems, thus shifting the attention of the research community to *software ecosystems* [2]. Such ecosystems are defined by Lungu [3] as collections of software projects that are developed and evolve together in the same environment. Examples of software ecosystems include package distributions for programming languages (e.g., CRAN, CPAN, RubyGems, npm, PyPI) or operating systems (e.g., Ubuntu, Debian), and mobile app stores. Software ecosystems differ from individual project development in the sense that projects in ecosystems can share source code (e.g., shared libraries) and contributors [4].

A significant part of many popular ecosystems is developed through GitHub, a portal for distributed versioning allowing developers to create and fork projects, link these projects via dependencies, and collaborate through a pull-based development process [5], [6], [7]. Developing software in such a way is an inherently *social* activity, involving the interaction, communication and collaboration of multiple contributors to the same project and across interdependent projects. It is also an inherently *technical* activity, involving the creation, generation and modification of a multitude of software artefacts such as source code, documentation, tests, metadata, and many more.

The socio-technical evolution dynamics of software ecosystems is an emerging research subject [8], [9], [10], but current studies mainly focus on *temporary* changes of the ecosystem. For example, an analysis of temporary changes will consider a contributor who becomes temporarily inactive but continues to contribute later on. In contrast, this paper focuses on *permanent* changes in the ecosystem and measures their effect on the ecosystem's evolution. For example, a contributor may decide to leave the ecosystem, abandoning the projects he is contributing to, and perhaps leading to abandoned files. The degree of renewal and abandonment may be indicative of quality problems in projects or in the ecosystem as a whole.

Building further on a preliminary empirical evaluation of the evolution dynamics of the Ruby on Rails project in GitHub [11], this paper empirically analyses the socio-technical evolution of the entire Ruby ecosystem in GitHub containing tens of thousands of projects. The goal is to gain an understanding of the evolution dynamics of base projects and their forks in this ecosystem, as well as the long-term effect of modifications in the social interaction to the ecosystem's source code artefacts. Ruby is a popular programming language ecosystem [12], [13] and most gems use GitHub as a collaborative source code repository [13]. Thus, we focus on Ruby projects that follow a pull-based development process [6] with pull requests being merged to the base projects.

We present our results at two levels of granularity: individual projects composed of a base project and its forks; and the global ecosystem composed of these individual (often interdependent) projects. For each level, we analyse the permanent changes of project, developer and source code files activity to demonstrate the effect of changes in the the ecosystem.

Our contributions can be summarized as follows:

- C1 We study the permanent modifications of the socio-technical network of the Ruby ecosystem in GitHub.
- C2 We measure the impact on the ecosystem when contributors permanently abandon projects.
- C3 We provide preliminary evidence about contributors migrating to different ecosystems.
- C4 We illustrate how data in GitHub can be used for socio-technical analyses of software ecosystems.

The remainder of this paper is structured as follows. Section II presents our research methodology. Sections III-VI present experimental results and discuss our findings. Section VII cross-validates our results using external data sources and Section VIII discusses the actionable results this work could lead to. Section IX reports the threats to validity of our research. Section X presents related work and finally, Section XI concludes this paper and provides future research directions.

II. Method

Research Questions

Our empirical study investigates the evolution of the Ruby ecosystem in GitHub. We consider two levels of granularity, namely the global ecosystem and the evolution around individual Ruby projects. Blincoe et al. [1] observed that ecosystems in GitHub tend to revolve around central projects having many dependent projects, forming star-like patterns. By analogy, we consider multiple base projects in Ruby, and study the co-evolution with their forks. We focus on four research questions:

- RQ_1 How does the ecosystem grow over time?
- RQ_2 How do the technical artefacts of the ecosystem evolve?
- RQ_3 How does the ecosystem's contributor team evolve?
- RQ_4 How do changes in the contributor team impact the technical artefacts?

The evolution of the Ruby ecosystem is studied according to the observed activity in GitHub. The earliest date of commit activity is registered on October 2007 and from this date until September 2016, the dataset is divided into 35 quarters (i.e., three-month intervals). Since the first and last quarter do not have any past or future data, respectively, the evolution of the Ruby ecosystem is studied for 33 quarter transitions. For each quarter we register the socio-technical network activity of each ecosystem "actor" as follows:

- Projects: We record the active base projects and their active fork projects which have activity in terms of committed source code, along with the commits of each project.
- Contributors: For each active base or fork project, we identify the individual contributors who authored the commits.
- Source code files: For each commit, we record the source code files that were committed and the lines of code (LOC) of each commit. We register only the number of added or modified lines of code, thus ignoring the deleted code lines. We follow this approach to measure the actual development effort of each commit, assuming that deleting code lines requires little effort compared to adding or modifying code.

We map each source code activity to its respective contributor. More precisely, each commit or LOC activity is mapped to both the contributor who performed the changes and to the project. We use Algorithm 1 to map source code files to distinct file identities since files can be added, deleted, renamed or moved throughout a project's evolution. The case of file deletion is omitted from Algorithm 1 since it does not have any effect on the file mapping. According to Algorithm 1, if a file is renamed or moved, then the previous name of the file is retrieved, and the new file is registered with the same identity. Otherwise, if a file in a commit is added or the file was not previously encountered in commits, a new identity is assigned to the file. The status of each file in each commit, i.e., whether the file is added, removed, modified, renamed or moved, is retrieved using the GitHub API[1]. Thus, file identities are used instead of their names to ensure that activity in the same file is correctly registered even if the file has been renamed or moved.

Algorithm 1 Mapping project source code files to unique ids

```
1:  procedure FileMapping
2:      commits ← Project commits sorted chronologically
3:      map <String,Integer>  ← Mapping of files to ids
4:      id = 1
5:      for each commit c ∈ commits  do
6:          files ← source code files of c
7:          for each file f ∈ files  do
8:              if status(f)== renamed or moved then
9:                  previous ← Previous name of f
10:                 mergeId ← map(previous)
11:                 map ← (f, mergeId)
12:             else if status(f)== added or f ∉ map then
13:                 map ← (f, id)
14:                 id = id + 1
15:             end if
16:         end for
17:     end for
18: end procedure
```

Metrics Definitions

For each transition from quarter $t-1$ to quarter t during the ecosystem evolution, we record the change in activity of each ecosystem actor (i.e., projects, contributors and source code files). Projects are considered as having become *obsolete* if they are inactive in a given quarter (i.e., no commit activity is recorded) and they remain inactive in all subsequent quarters until the end of the observation period. Source code files become *obsolete* if, starting from a given quarter they are never touched again in any commit of our dataset until the end of the observation period. Contributors are considered to

[1]https://developer.github.com/v3/repos/commits/

TABLE I
DEFINITIONS OF CONTRIBUTOR (c), PROJECT (p) AND FILE (f) METRICS FOR EACH QUARTER t.

$ObsoleteProjects(t)$	$\{p \mid isActive(p,t-1) \wedge \forall\, i \geq t, \neg isActive(p,i)\ \}$
$NewProjects(t)$	$\{p \mid isActive(p,t) \wedge \forall\, i < t, \neg isActive(p,i)\ \}$
$ActiveProjects(t)$	$\{p \mid isActive(p,t) \wedge isActive(p,t-1)\}$
$ProjectRenewal(t)$	$\mid NewProjects(t) \mid\ /\ \mid \{p \mid isActive(p,t)\} \mid$
$ProjectAbandonment(t)$	$\mid ObsoleteProjects(t) \mid\ /\ \mid \{p \mid isActive(p,t-1)\} \mid$
$Leavers(t)$	$\{c \mid isContr(c,t-1) \wedge \forall\, i \geq t, \neg isContr(c,i)\ \}$
$Joiners(t)$	$\{c \mid isContr(c,t) \wedge \forall\, i < t, \neg isContr(c,i)\ \}$
$Stayers(t)$	$\{c \mid isContr(c,t) \wedge isContr(c,t-1)\ \}$
$TeamRenewal(t)$	$\mid Joiners(t) \mid\ /\ \mid \{c \mid isContr(c,t)\} \mid$
$TeamAbandonment(t)$	$\mid Leavers(t) \mid\ /\ \mid \{c \mid isContr(c,t-1)\} \mid$
$ObsoleteFiles(t)$	$\{f \mid isTouched(f,t-1) \wedge \forall\, i \geq t, \neg isTouched(f,i)\ \}$
$NewFiles(t)$	$\{f \mid isTouched(f,t) \wedge \forall\, i < t, \neg isTouched(f,i)\ \}$
$ActiveFiles(t)$	$\{f \mid isTouched(f,t) \wedge isTouched(f,t-1)\}$
$FileRenewal(t)$	$\mid NewFiles(t) \mid\ /\ \mid \{f \mid isTouched(f,t)\} \mid$
$FileAbandonment(t)$	$\mid ObsoleteFiles(t) \mid\ /\ \mid \{f \mid isTouched(f,t-1)\} \mid$

be *leavers* in a given quarter if they have not contributed in that quarter or any subsequent quarters until the end of the observation period. Contributors are considered to be *joiners* in a given quarter if they have contributed in that quarter but not in any preceding quarter.

Table I presents the metrics we used for measuring *permanent* changes concerning the activity of ecosystem actors. The metrics definitions consider a project p, a contributor c, a source code file f, and two successive quarters $t-1$ and t. The predicate $isActive(p,t)$ is true if and only if there is a source code commit for project p in quarter t; $isContr(c,t)$ is true if and only if contributor c made a source code commit in quarter t; $isTouched(f,t)$ is true if and only if file f was touched through commits in t.

We formally define $ObsoleteProjects(t)$ as the set of projects that do not have any commit activity from quarter t onward. $NewProjects(t)$ are the projects that have commit activity for the first time during quarter t, and $ActiveProjects(t)$ are those projects that have commit activity in both quarters $t-1$ and t. With respect to the ecosystem contributors, we formally define $Leavers(t)$, $Joiners(t)$ and $Stayers(t)$ as the contributors who did not have any contributions from quarter t onward, had never contributed before t, and contributed in both quarters t and $t-1$ respectively. Source code file modifications $ObsoleteFiles(t)$, $NewFiles(t)$ and $ActiveFiles(t)$ are defined as the files that were never touched again, that were never touched before, and that were touched in both quarters t and $t-1$, respectively.

For projects, we define $ProjectRenewal(t)$ as the ratio of new against active projects in quarter t, and $ProjectAbandonment(t)$ as the ratio of obsolete projects in quarter t against active projects in quarter $t-1$. Renewal and abandonment is defined similarly for teams and files.

Dataset & Preprocessing

To facilitate reproducibility of our study, all extracted data we have used is made available in a public dataset on https://bitbucket.org/econst/rubyecosystemgithub-saner17.

We used the 2016-09-05 SQL dump of the GHTorrent dataset [5] to obtain data for GitHub projects that are part of the Ruby ecosystem. GitHub has been a valuable data source for ecosystem researchers to mine information stored in GitHub for empirical studies. However, the benefits of using data obtained from GitHub can be undermined when researchers do not ensure the validity of the repositories they use in their studies. To this end, it is important to include a pre-processing step to filter out projects that can potentially introduce noise and degrade the accuracy of research results.

We present the filtering steps we carried out to select the projects of the Ruby ecosystem, in order to limit the perils of using GitHub projects that are discussed by Kalliamvakou et al. [14]. We explain the rationale behind, and benefits of, each filtering step. In Table II we match each peril of using GitHub data to the appropriate filter that we have applied to avoid or reduce the peril.

Filter 1: Eliminate non-Ruby projects. Projects that do not have Ruby as their main programming language are excluded from further analysis since we rely on the Ruby ecosystem in GitHub. The main programming language of each project is identified by the *language* column of the *projects* table of GHTorrent. Although the dataset provides a list of all the languages used in a repository (*project_languages* table of GHTorrent), we are interested in the main programming language of each project.

Filter 2: Eliminate projects with scarce commit activity. We excluded projects that are either marked as deleted, or have scarce commit activity. To identify such projects, we gathered each project's commit activity and eliminated projects with less than five commits from our dataset.

Filter 3: Eliminate isolated projects. We excluded all base projects that did not receive any merge from its forks during the considered observation period. We applied this filter since our empirical study focuses on the socio-technical interaction and co-evolution between base and fork projects.

Filter 4: Eliminate forks without merges to the base project. For each base project, we excluded all forks without any merge

TABLE II
PERILS OF MINING GITHUB [14] AND FILTERING STRATEGY

Peril	Filter
A repository is not necessarily a project	Filter 2
Low project activity	Filter 2
Inactive projects	Filter 2, 4
Non software development projects	Filter 6
Personal projects	Filter 3
Few projects use pull requests	Filter 4
GitHub's API does not expose all data	Don't use commits as the measurement unit, rather LOC of commit files
GitHub is continuously evolving	Use a recent GitHub dataset

to the base project. Such forks do not involve any interaction with the base project and are therefore not relevant to the current empirical study.

Filter 5: Eliminate short-lived contributors. Occasional or one-time contributors tend to introduce quite some noise in the dataset. To eliminate their effect, we excluded all contributors (and their activity) for whom the timespan between the first and last observed commit activity was less than three months. This filter ensures that TeamRenewal and TeamAbandonment are not overestimated since such contributors join and abandon the ecosystem in consecutive quarters and their contribution does not reflect the actual workload.

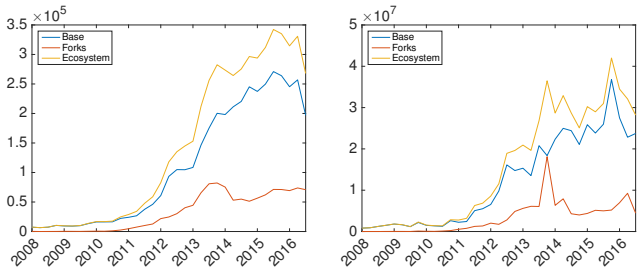
Filter 6: Only consider source code files in commits. Since our focus is on studying the effort involved in software development activity, we only considered Ruby source code files with the .rb extension.

We found the GHTorrent dataset to be incomplete with regard to additional information that is necessary for our analyses, i.e., the modifications in the source code artefacts of the studied projects. More precisely, although GHTorrent provides information about commits (author, committer, date, comments, etc), it does not provide a list of the modified files nor the patch of each commit. Therefore, we queried GitHub to obtain the commit information for all projects and all commits during the considered observation period. For each commit, we queried the GitHub API[2] and recorded the files touched in each commit according to the response to each query (in JSON format). For each file, we registered the filename, file status (modified, renamed, removed, added), number of added and deleted lines, and the patch. As mentioned earlier, we relied on the number of *added* and *modified* LOC to measure developer productivity. It should be noted that modified lines are represented by one added and one deleted line in the information provided by GitHub, meaning that the added field consists of all the new lines and the modified ones. Although it is possible to extract the exact number of modified lines of code from patches [15], this requires comparing two versions of the source code. Since our dataset only contains the git patches, this would require a time-consuming analysis of the patch to compute how many lines were modified or added. For our purposes, we assume that the effort to add or modify a line of code is similar. It suffices to rely on the added field queried from GitHub to obtain this information, avoiding the processing of git patches.

[2]https://developer.github.com/v3/repos/commits/

TABLE III
ECOSYSTEM DESCRIPTIVE STATISTICS

	Base	Forks	Ecosystem
Projects	25,611	69,008	94,619
Contributors	59,830	45,496	76,320
Touched Files	780,702	235,575	815,623
Commits	3,871,281	1,141,589	5,012,870
LOC	427,918,256	114,782,090	542,700,346



(a) Commits (b) Lines of Code (LOC)

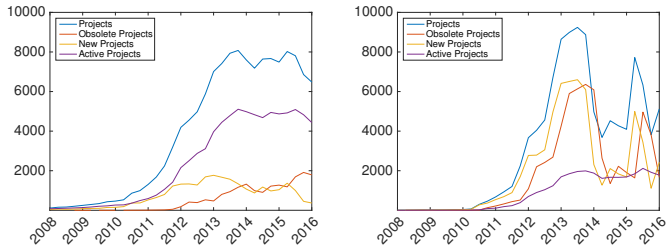
Fig. 1. Evolution of team changes

III. RQ$_1$ HOW DOES THE ECOSYSTEM GROW OVER TIME?

Table III presents descriptive statistics of the studied Ruby ecosystem. Our observation period starts on 29 October 2007 and ends on 3 September 2016, resulting in 35 quarters. Throughout the paper, all figures and discussions will present the evolution in terms of years to facilitate the interpretation of our results.

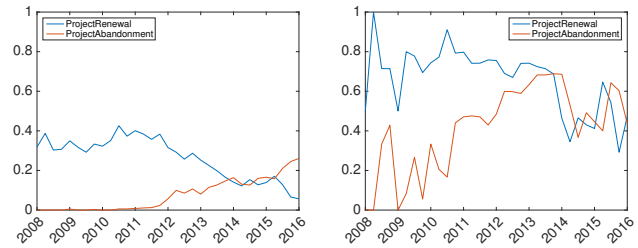
Our dataset includes the evolution history of 25,611 base projects and 69,008 forks developed around them, resulting in 94,619 projects. In total, 76,320 distinct contributors committed source code for these projects, of which 78% contributed to the base projects and 60% to the forks. 780,702 files were touched in base projects, while 235,575 files were touched in forks, with the total number of files touched in the ecosystem corresponding to 815,623. Finally, over 3.8 million commits were registered for these projects, the majority of which (77%) being recorded for the base projects. Over 542 million lines of code were added or modified in these commits for the ecosystem, 79% of which belonged to the base projects.

Figure 1 presents the evolution of the Ruby ecosystem in terms of commits and lines of code. The main development activity takes place in the base projects. After August 2011, there is a significant increase in the development effort, and this increased effort continues until the end of the observation period. After February 2015, a drop in the number of commits is observed in Figure 1(a). This is misleading, since the measurement of LOC in Figure 1(b) shows an increase for this period of time instead. The observed difference is probably due to the recommended practice of squashing commits before a



(a) Base Projects (b) Forks

Fig. 2. Evolution of base projects and forks in Ruby



(a) Base Projects (b) Forks

Fig. 3. Evolution of Renewal and Abandonment of Ruby projects

merge. The contribution guidelines of Ruby on Rails[3], one of the largest and most active Ruby projects in GitHub, instruct contributors to squash multiple commits into a single one. Hence, in the remainder of this article we will measure development effort in terms of LOC rather than commits as it provides more reliable values.

Figure 2 shows the evolution of the number of obsolete, new and active base projects and forks in the Ruby ecosystem. After May 2010, both the number of base projects and forks present a significant increase, while after February 2014 a decrease is observed. In the time period between May 2010 and February 2014, there is an increased rate of new base projects and low rate of base projects becoming obsolete. In contrast, forks have similar rate of new and obsolete projects, which can be explained by the fact that their lifecycle is limited compared to the base projects. After February 2014, new base projects are observed less often and more base projects become obsolete compared to the previous ecosystem's history. A drop in the number of new and active projects is also observed in the fork projects.

Figure 3 presents the ProjectRenewal and ProjectAbandonment for the base projects and forks of Ruby. 30-40% of the base projects in each quarter are new, while abandoned projects remain below 10% until February 2014. As expected, the new project rate in forks is over 60% until February 2014, while the increased abandonment rate of approximately 40% on average shows the short longevity of forks.

In an effort to explain the observed phenomena after February 2014, we gathered anecdotal evidence from developer blogs and Ruby's mailing lists. In November 2013, a thread in the Ruby mailing list discusses the survival of Ruby as a

[3]http://edgeguides.rubyonrails.org/contributing_to_ruby_on_rails.html

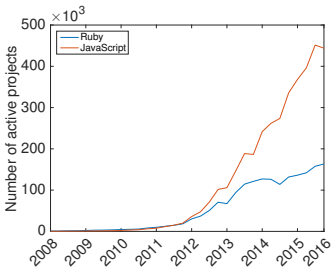


Fig. 4. Ruby and JavaScript evolution (active base projects)

programming language[4]: *"Will ruby die in the future? We have some new technologies like Node.js and angular but how ruby can challenge them?"* and the consensus from the replies of the Ruby community is that Ruby will continue to exist in the future but it needs to transform: *"From being in technology for a long time I have realized technologies never really die they just transform"*. Furthermore, we gathered anecdotal evidence about technology migration from Ruby at that period. Firstly, there is evidence of projects moving to Node.js due to performance issues[5] [6] and companies like Twitter moving to other technologies to handle increased traffic [7].

To support this anecdotal evidence, we investigated the activity in the Ruby and JavaScript ecosystems for the same timespan by measuring the number of active base projects in each quarter that are derived by their commit activity. We used the GHTorrent dataset to obtain the commit activity of projects in each programming language for each quarter without applying any filters to the project selection. The results are presented in Figure 4. Until 2012 both ecosystems have comparable growth, whereas from 2012 onward, the JavaScript ecosystem in GitHub presents a larger growth compared to the Ruby ecosystem. A possible explanation is that developers migrate to ecosystems of other technologies like JavaScript, since there is evidence that contributors involved in Ruby projects are likely to contribute to JavaScript projects and vice versa [16]. We provide empirical evidence of developer migration in the social investigation of the ecosystem in the presentation of the results of Section V. Nonetheless, further research is required to verify if the evolution dynamics of both ecosystems affect one another.

IV. RQ_2 How do the technical artefacts of the ecosystem evolve?

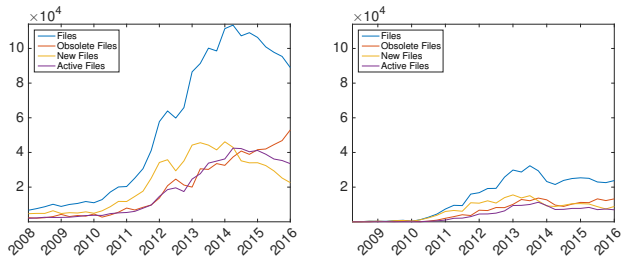
Figure 5 presents the evolution of the Ruby ecosystem with respect to the development and maintenance of source code files. Considering the growth of the ecosystem until February 2014, the increase in the number of actively developed source code files is an expected outcome. According to the results of Figure 5, the bulk of the development activity takes place in

[4]http://blade.nagaokaut.ac.jp/cgi-bin/vframe.rb/ruby/ruby-talk/411961?411838-412226

[5]http://ilikekillnerds.com/2015/02/is-ruby-on-rails-dying/

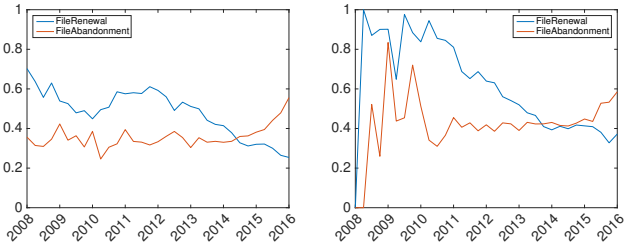
[6]http://blog.parse.com/learn/how-we-moved-our-api-from-ruby-to-go-and-saved-our-sanity/

[7]http://www.theregister.co.uk/2012/11/08/twitter_epic_traffic_saved_by_java/



(a) Base Projects (b) Forks

Fig. 5. Evolution of Ruby source code files in base projects and forks



(a) Base Projects (b) Forks

Fig. 6. Ecosystem of renewal and abandonment of Ruby source code files

the base projects, while the technical evolution shows similar attributes for both views of the ecosystem. For both views, the number of new files is comparable to the number of obsolete files, indicating the implementation of new functionality. However, after February 2014, there is a drop in the number of new files for the base projects, indicating either the contributors' focus on maintaining existing files or a reduction in development effort.

Figures 6 shows the FileRenewal and FileAbandonment for the base projects and forks respectively. For the base projects, the average renewal and abandonment correspond to 48% and 36% respectively, confirming the ecosystem growth until February 2014, where file abandonment starts to exceed file renewal. For forks, the average FileRenewal and FileAbandonment corresponds to 61% and 42% respectively. This is an expected outcome considering the pull-based development process. Since it is a common practice to add new functionality via forks prior to merging it back to the base project so as not to disturb the code in the original branch [17], we expect to see large values of FileRenewal. With respect to FileAbandonment, larger values are expected since forks are characterized by a short maintenance time and afterwards become inactive [18]. We observe increased abandonment from February 2014 onwards for forks as well.

V. RQ$_3$ How does the ecosystem's contributor team evolve?

The results of the social evolution of the Ruby ecosystem are presented in Figure 7. According to Figure 7(a), an increase in the number of Joiners and Leavers for base projects is observed after August 2010. Contributor activity is increasing until February 2014 and a drop is observed after May 2015. According to Figure 7(b), the number of contributors to fork projects shows a large increase between May 2010 and November 2013, and from February 2014 the number of contributors participating in forks suddenly drops. Considering that the total number of ecosystem contributors throughout its evolution is not greatly affected by the fork participation, this seems to imply that the majority of these contributors that left the forks continue their participation in base projects. We also observe in Figure 7(c) that the number of Leavers in the ecosystem is rapidly increasing after November 2012 and at the same time, the number of Joiners is decreasing.

Figures 7(d)-(f) show the ecosystem evolution in terms of TeamRenewal and TeamAbandonment for the base projects, forks and the global ecosystem. For all three views, TeamRenewal decreases over time, indicating that less people join the ecosystem each quarter with respect to Ruby's community size in GitHub. More precisely, on average 38% of the ecosystem development team consists of Joiners each quarter, whereas 21% of the team abandons the ecosystem each quarter. Our results are in agreement with Foucault et al. [19] who observed that newcomers do not tend to become stayers.

Finally, the abandonment rate exceeds the joining rate of the ecosystem after February 2014 and the number of active developers is reduced. Combined with our observations concerning the ecosystem projects in Section III, this reveals evidence of a possible correlation between developer abandonment and project abandonment.

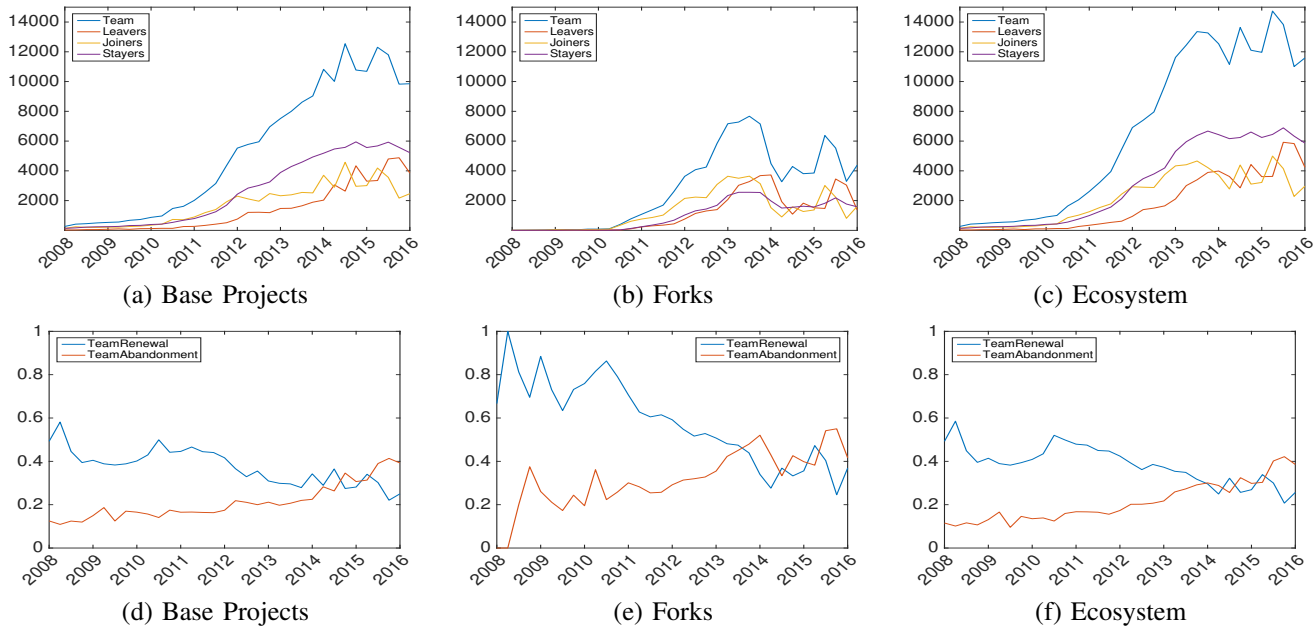
To further investigate the behavior of contributors abandoning the Ruby ecosystem, we measured their activity on GitHub projects with another main programming language *during* their active period in Ruby and *after* abandoning Ruby. Table IV summarizes the results concerning the top 10 programming languages for activity of contributors for the period when they were active in Ruby (first and second column of Table IV) and when they abandoned Ruby (third and fourth columns of Table IV).

Our findings suggest that Ruby contributors worked in parallel on JavaScript, Python and HTML projects on GitHub, and these results are confirmed by the observations in [16]. Also, the majority of Ruby contributors that used to contribute to JavaScript in parallel to Ruby, continue their activity on JavaScript projects after abandoning the Ruby ecosystem.

We consider these observations as initial evidence of a correlation between the evolution of the Ruby and JavaScript ecosystems. In future work, we aim to investigate the extent of developer migration and the characteristics of their activity in both ecosystems. In the context of this work, we conclude that there is an increasing number of contributors abandoning the ecosystem, either by becoming inactive in GitHub or leaving the Ruby ecosystem and contributing to other ecosystems.

VI. RQ$_4$ How do changes in the contributor team impact the technical artefacts?

To measure the impact of Leavers (contributors abandoning the Ruby ecosystem), we measure their diversity index in the project-contributor graph of the Ruby ecosystem, inspired by the work of Posnett et al [20]. Diversity measures are



(a) Base Projects (b) Forks (c) Ecosystem

(d) Base Projects (e) Forks (f) Ecosystem

Fig. 7. Social evolution of Ruby (for base projects, forks and the global ecosystem)

TABLE IV
MIGRATION OF RUBY ABANDONERS TO OTHER PROGRAMMING LANGUAGE ECOSYSTEMS.

Language	Active in Ruby	Language	Abandoned Ruby
JavaScript	28,322	JavaScript	20,198
Python	14,901	HTML	12,027
HTML	12,228	Python	10,777
Java	11,379	Java	7,233
C	8,877	C	5,181
PHP	8,862	Go	4,923
VimL	7,013	PHP	4,845
C++	6,635	VimL	4,839
CoffeeScript	5,766	C++	4,192
Go	4,885	CoffeeScript	3,328

borrowed from ecology and, when applied to bipartite graphs, they express the specialization of a given species with respect to the species in the other level [21]. Project and contributor diversity can be measured according to Shannon's entropy or the Simpson index, while the specialization of a species in one level relative to the species in the other level is measured by the relative entropy (a.k.a. Kullback-Liebler divergence) [10], [20]. By measuring the specialization of Leavers, we assess the relative risk they cause to the ecosystem (according to their relative contribution) until they abandoned the ecosystem. As explained in [21], the specialization of a contributor c_j expressed in terms of relative entropy is defined as:

$$S_{c_j} = \sum_{i=1}^{n} \frac{w_{ij}}{C_j}(log\frac{w_{ij}}{C_j} - log\frac{P_i}{W})$$

where n is the number of projects and m the number of contributors in the ecosystem, w_{ij} the workload of contributor c_j to project p_i counted in number of lines of code, $P_i = \sum_{j=1}^{m} w_{ij}$ the total workload of all contributors to project p_i, $C_j = \sum_{i=1}^{n} w_{ij}$ the total workload of contributor c_j on all projects she is contributing to, and $W = \sum_{i=1}^{n}\sum_{j=1}^{m} w_{ij}$ the total ecosystem workload.

We computed the contributor specialization with a time constraint on the project and ecosystem workload. More precisely, we consider the ecosystem and contributor workload for the quarters where the contributor was active in the ecosystem, that is from the quarter of her first contribution until the quarter before abandoning the ecosystem. The boxplots in Figure 8 present the specialization of Leavers of the Ruby ecosystem in each quarter for Leavers with $S_{c_j} > 0$. Considering that the bulk of Leavers have low specialization, we only focus on contributors with increased specialization, i.e., leavers who have large contributions to important projects of the ecosystem. The departure of such people from the ecosystem may present important risks for the ecosystem's sustainability. According to the results of Figure 8, the majority of Leavers of the Ruby ecosystem do not differ in terms of specialization throughout its evolution. From August 2010 onward, however, there are more and more outliers. These represent contributors with large specialization abandoning the ecosystem, indicating potential risks for Ruby's sustainability.

VII. CROSS-VALIDATION: GITHUB VS RUBYGEMS

Our empirical results reveal that the evolution of the Ruby ecosystem in GitHub is faced with a downturn from early 2014. Considering that we provide a quantitative analysis according to data gathered from GitHub, we cross-validated our results using three external data sources. First, we measured the number of unfiltered active Ruby projects in GitHub to ensure that our filtering approach did not bias the outcome of this study. Figure 9(a) displays this data, and reveals a large drop of Ruby project activity in GitHub from mid-2015. Secondly, we

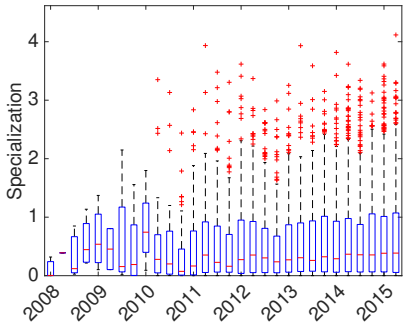
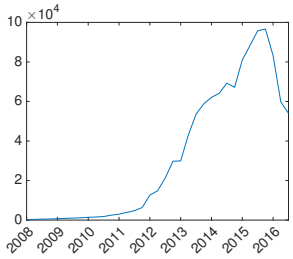
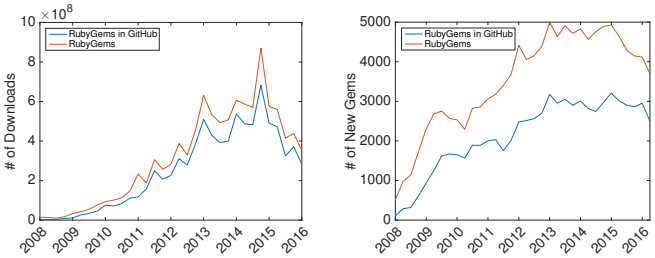


Fig. 8. Leavers specialization in the ecosystem



(a) Number of unfiltered Ruby projects in GitHub



(b) Downloads (c) New Gems

Fig. 9. Ruby ecosystem evolution

measured the number of gem downloads from the RubyGems API in order to assess if the results coincide with our observations in GitHub. We display the results in Figure 9(b) for all gems and for the subset of gems that are developed in GitHub, and observe a clear drop of gem downloads from late-2014 onward. Finally, we used the RubyGems API to measure the number of newly introduced gems in RubyGems, so as to investigate if the growth dynamics of the ecosystem align with our observations in GitHub. The results are displayed in Figure 9(c), where a decrease of new gems is observed from 2015 onward. These external measures therefore cross-validate and confirm our empirical observations that were based on Ruby projects in GitHub only.

VIII. Discussion

We focused on the evolution of Ruby projects in GitHub due to the popularity of the GitHub platform for software development and because most Ruby gems use GitHub as a collaborative source code repository [12], [13], enabling us to study the social aspect of the ecosystem.

The metrics we defined in Section II enable the measurement of permanent changes during an ecosystem's evolution. Changes of these metrics throughout the ecosystem's evolution may be used to assess risks related to ecosystem's sustainability. For example, measuring TeamRenewal and TeamAbandonment can assist in identifying social disturbances in the ecosystem early, by detecting successive drops or increases respectively. Combined with information about Leavers specialization, the magnitude of these changes could be used to assess the importance of abandoners of the ecosystem. We also showed that combining measurements of permanent social changes with external observations on the ecosystem can strengthen the confidence of increased risk for loss of sustainability of the ecosystem. More precisely, sustainability loss in an ecosystem can be detected when large social changes are accompanied by similar observations of the ProjectRenewal and ProjectAbandonment measurements in the same periods of time, showing the effect of social changes on the ecosystem evolution.

Measuring the amount and effect of socio-technical changes during ecosystem evolution can assist interested stakeholders to identify risks and help them to take corrective measures so as to reduce the negative effect of such disturbances. Core package maintainers can monitor the evolution within the developer community and identify problems regarding the attraction of new contributors, as well as their attachment to the developer community. In particular, the Ruby community can benefit from our work by identifying suitable Ruby projects for newcomers in the ecosystem, as well as by assessing the risk of reduced ecosystem sustainability when highly specialized contributors abandon the ecosystem.

IX. Threats to Validity

A threat to the validity of our work is that we assume each account in GitHub to correspond to a distinct contributor. Contributors tend to use multiple accounts across different repositories (e.g., version control system, bug tracker and mailing list), but using multiple accounts to commit source code in the same GitHub repository is less common. To eliminate the effect of this threat, we will include an identity merging preprocessing step in future work [22].

Another threat stems from the fact that we identified Ruby projects in GitHub based on the main programming language set by GitHub. GitHub uses the Linguist Library[8] to determine the programming language of each file within a project. The wrong language can be set as a project's primary language when the primary code is smaller than code files in other languages[9]. Indeed, some projects in the GHTorrent dataset appear to be written mainly in CSS. This may cause some Ruby projects on GitHub to be excluded from our dataset, or certain primarily non-Ruby projects to be included.

Additional validity threats stem from the usage of a large GitHub dataset. Section II lists the multiple filters we applied

[8]https://github.com/github/linguist

[9]https://help.github.com/articles/my-repository-is-marked-as-the-wrong-language/

to eliminate noise resulting from using such datasets. However, Table II in Section II does not consider some perils that may bias our results. The first peril concerns activity outside GitHub, which poses a risk to our approach only in the case of external development with mirrors in GitHub. Filter 2 limits the effect of this peril, but cannot guarantee that all development activity of the projects is effectively performed through GitHub. The second peril is that merges only track successful code, where multiple commits are squashed together before being merged to the main repository. We eliminate the effect of this peril by considering lines of code instead of the number of commits as an indication of the development effort. The third peril is that many merged pull requests appear as non-merged by GitHub, i.e., merges were done using git directly instead of using GitHub's pull request facilities, and heuristics must be considered to determine whether or not a pull request was merged or not. This peril is not mitigated in our approach since we rely on the GitHub pull request mechanism, while it is known that many projects use a combination of GitHub and git merge strategies [14]. The last peril is that not all activity comes from registered users. The impact of this threat to the outcome of our study is limited, since Kalliamvakou et al. [14] found that 84.4% of commits were performed by registered users.

Another threat to the validity of our work stems from the fact that we split our dataset into three-month periods to measure social and technical changes throughout the ecosystem's evolution. Although this approach has been previously used in other evolution studies [9], [10], other time intervals may be more appropriate. In future work we will determine optimal intervals using proper statistical analyses.

Finally, the filtering of the initial GitHub dataset can overestimate joiners or abandoners since they can be active in Ruby projects outside of GitHub. However, the threat of measuring only the contributor activity in the active part of the Ruby ecosystem is preferred compared to the noise of including isolated or inactive projects in our analysis.

X. Related Work

Software ecosystems in GitHub have gained the interest of the research community since different studies focus either on their identification [1], visualization [23], [24], [25], [26] and analysis [13], [12], [25], [27], [28], [29]. This section focuses on related work of three different aspects of software ecosystem research, i.e., the identification and analysis of software ecosystems in GitHub, dynamics behind ecosystem evolution and social aspects of ecosystem evolution.

A. GitHub Ecosystems

Blincoe et al. [1] present the reference coupling method that detects technical dependencies between software projects to identify software ecosystems in GitHub-hosted projects. Their findings show that most ecosystems are centered around one project and they are interconnected with other ecosystems, while the predominant type of ecosystems are tools supporting software development. From a social perspective, they found that owners' social behavior aligns with the technical dependencies, but not vice versa. It should be noted that Blincoe et al. found cross-references between projects in comments on issues, pull requests and commits that follow specific patterns (User/Project#Num or User/Project@SHA). The work of Thung et al. [30] creates a weighted project-project and developer-developer network of projects and developers in GitHub. According to their approach, projects have edges between them if they have at least one common developer contributing to both projects. However, this method does not accurately reflect relationships between projects since technical dependencies are not considered.

B. Dynamics behind Ecosystem Evolution

Rigby et al. [31] quantified the extent of abandoned source files in one industrial and one open source project. They measure the historical loss distribution of these projects, measured as Leavers' expertise, and find that projects are susceptible to losses that are more than three times larger than the expected loss. Next, they use historical simulations based on the loss distribution and discover that projects are susceptible to losses that are over five times larger than the expected loss. Overall, they found that, when tight relationships between the author and the source code exist, it is more difficult to replace the authors by newcomers since they are less productive and more prone to making errors.

Wittern et al. [32] analyzed the evolution of the npm ecosystem of JavaScript-based software packages. They found that npm is a striving ecosystem with accelerating growth of packages and increasing dependencies between them. They also found that at the same time, the developer community remains quite active with respect to the maintenance of their packages. German et al. [33] studied the R ecosystem evolution and found that a healthy community and well-maintained packages are essential to the successful evolution of the ecosystem. Decan et al. [34] also studied the R ecosystem, but they focused on the use of GitHub for the distribution of R packages and inter-repository package dependency management. The authors found an increased rate of R package hosting in GitHub and exclusively distributing them through GitHub, where a major concern is the lack of support for dependency constraints since packages are not systematically monitored by a continuous integration process like in CRAN.

C. Social Aspects

Social aspects of software ecosystem evolution have also been studied by the research community. Foucault et al. [19] studied open source projects to characterize patterns of turnover and determine the effect of turnover on software quality. They defined the external and internal turnover, where in the first case members leave or join the team and in the latter case members change their role in the team. According to their findings on five open source systems, external turnover has a negative impact on the quality of the software. However, in their study they do not distinguish the core members of the

development team, whereas we eliminate possible threats of occasional contributors by filtering our dataset.

Aué et al. [35] investigated the relationship between project growth and social diversity. Project growth is measured in terms of team, commit, pull request and comment growth, while diversity reflects the gender and geographical diversity. After rating over 3,000 projects in a 5-star scale capturing each project's success, they found a statistically significant but minor relation between project success and the two diversity metrics. Vasilescu et al. [9] presented a dataset of social diversity attributes of GitHub contributors contributing to 23,493 GitHub projects. In [10], the authors studied gender and tenure diversity and how they relate to team productivity and turnover. Their findings show that increased gender and tenure diversity are associated with greater productivity. They also found that turnover is positively associated with tenure diversity. Unlike these studies, we focus on permanent renewal and abandonment of the ecosystem's entities to investigate their long-term effects on the ecosystem evolution.

Social aspects of the Ruby ecosystem in particular have been studied in recent works. Kabbedijk and Jansen [12] identified different types of contributors in the Ruby ecosystem: networkers, lone wolfs, and one day flies. Their dataset was based on gems from RubyGems, while we used data of Ruby project activity in GitHub. Their findings showed that developers fulfil different roles in the ecosystem and that most activity is performed only in a small part of the ecosystem. Also, they concluded that it is a better practice to motivate existing developers to work on existing gems, rather than expanding the ecosystem with new ones. Syed and Jansen [36] combined social network analysis with a survey to identify clusters of contributors, and revealed that the RubyGems ecosystem consists mostly of independent developers. Syeed et al. [13] empirically studied the relationship between developer coordination activities and the project dependency structure in the Ruby ecosystem. They used a dataset with packages of RubyGems, for which they retrieved their issues from the respective GitHub repository. They found that socio-technical congruence exists among developers of the same project, but decreases at ecosystem level. In contrast to these works, we focus on the development of Ruby projects, including active gems, in GitHub and measured the effect of permanent social modifications to the ecosystem evolution.

Multiple researchers have studied the impact of *socio-technical congruence* on software development. Such congruence assumes a technical structure of the software system that *mirrors* the social structure of the developer community [37], [38]. Cataldo et al. [39] observed a positive influence of socio-technical congruence on developer's productivity. Syeed et al. [13] however, found counter-evidence of such congruence for the Ruby ecosystem. In analogy to the notion of technical debt, Tamburri et al. investigated the notion of *social debt*, identifying community smells that may be indicative of suboptimal organisational structures [40]. Identifying such community smells in software ecosystems, and relating them to technical problems, remains an open area of research.

XI. Conclusion

This paper presented an empirical study of the socio-technical evolution of the Ruby ecosystem in GitHub over a nine-year period. We studied the evolution of nearly 95K active Ruby projects, consisting of over 25K base projects and their active forks. These contributions involved more than 76K distinct contributors, totalling over 5 million commits and representing 542 million lines of added or modified Ruby source code. We measured permanent changes with regard to the ecosystem's projects, contributors and source code files and presented our findings concerning the effect of these modifications. We discussed our findings with respect to the ecosystem evolution according to three different viewpoints: (1) the base projects, (2) the forks and (3) the entire ecosystem containing both the base projects and forks.

Our results show increased growth of both the technical and social aspects of the Ruby ecosystem until early 2014, where the ecosystem is being abandoned by the development community. The abandonment of Ruby contributors is clearly affecting the ecosystem's evolution considering the downturn of Ruby projects in GitHub. We provided both anecdotal evidence and empirical evidence concerning the evolution of the Ruby ecosystem. In the latter case, we cross-validated our empirical observations with measurements from both GitHub and RubyGems. Finally, we provided anecdotal empirical evidence of contributors migrating to other ecosystems, with JavaScript, HTML and Python being the most frequent ecosystems to which Ruby developers migrate.

Following the spirit of [41], in future work we will compare the socio-technical evolution of the Ruby ecosystem to the one of other programming language ecosystems. We will also study inter-ecosystem migration of contributors and carry out more advanced socio-technical analyses [8], such as the presence of socio-technical debt [40] and the effect this has on the evolution of the ecosystem.

Acknowledgment

This research was carried out in the context of ARC research project AUWB-12/17-UMONS-3 entitled "Ecological Studies of Open Source Software Ecosystems" and FNRS crédit de recherche J.0023.16 entitled "Analysis of Software Project Survival". We express our gratitude to Philippe Grosjean and the anonymous reviewers for the very useful feedback on an earlier version of this article.

References

[1] K. Blincoe, F. Harrison, and D. Damian, "Ecosystems in GitHub and a method for ecosystem identification using reference coupling," in *Working Conference on Mining Software Repositories (MSR)*, 2015, pp. 202–207.

[2] A. Serebrenik and T. Mens, "Challenges in software ecosystems research," in *European Conference on Software Architecture Workshops*, 2015, pp. 40:1–40:6.

[3] M. Lungu, "Towards reverse engineering software ecosystems," in *International Conference on Software Maintenance (ICSM)*, 2008, pp. 428–431.

[4] W. Scacchi, "Free/open source software development: Recent research results and emerging opportunities," in *Joint Meeting on European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering: Companion Papers (ESEC-FSE companion)*, 2007, pp. 459–468.

[5] G. Gousios, "The GHTorrent dataset and tool suite," in *Working Conference on Mining Software Repositories (MSR)*, 2013, pp. 233–236.

[6] G. Gousios, M.-A. Storey, and A. Bacchelli, "Work practices and challenges in pull-based development: The contributor's perspective," in *International Conference on Software Engineering (ICSE)*, 2016, pp. 285–296.

[7] G. Gousios, A. Zaidman, M.-A. Storey, and A. van Deursen, "Work practices and challenges in pull-based development: The integrator's perspective," in *International Conference on Software Engineering (ICSE)*, 2015, pp. 358–368.

[8] T. Mens, "An ecosystemic and socio-technical view on software maintenance and evolution," in *International Conference on Software Maintenance and Evolution (ICSME)*, 2016.

[9] B. Vasilescu, A. Serebrenik, and V. Filkov, "A data set for social diversity studies of GitHub teams," in *Working Conference on Mining Software Repositories (MSR)*, 2015, pp. 514–517.

[10] B. Vasilescu, D. Posnett, B. Ray, M. G. van den Brand, A. Serebrenik, P. Devanbu, and V. Filkov, "Gender and tenure diversity in GitHub teams," in *ACM Conference on Human Factors in Computing Systems (CHI)*, 2015, pp. 3789–3798.

[11] E. Constantinou and T. Mens, "Social and technical evolution of software ecosystems: A case study of Rails," in *European Conference on Software Architecture Workshops (ECSAW)*, 2016, pp. 23:1–23:4.

[12] J. Kabbedijk and S. Jansen, *Steering Insight: An Exploration of the Ruby Software Ecosystem*. Springer, 2011, pp. 44–55.

[13] M. M. M. Syeed, K. M. Hansen, I. Hammouda, and K. Manikas, "Socio-technical congruence in the Ruby ecosystem," in *International Symposium on Open Collaboration (OpenSym)*, 2014, pp. 2:1–2:9.

[14] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "An in-depth study of the promises and perils of mining GitHub," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2035–2071, 2016.

[15] G. Canfora, L. Cerulo, and M. D. Penta, "Identifying changed source code lines from version repositories," in *International Workshop on Mining Software Repositories (MSR)*, 2007, pp. 14–.

[16] C. Cronin and N. S. Nikolov, "Visualization of GitHub's public data," University of Limerick, Tech. Rep., 2014.

[17] J. Tsay, L. Dabbish, and J. Herbsleb, "Influence of social and technical factors for evaluating contribution in GitHub," in *International Conference on Software Engineering (ICSE)*, 2014, pp. 356–366.

[18] S. Stanciulescu, S. Schulze, and A. Wasowski, "Forked and integrated variants in an open-source firmware project," in *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2015, pp. 151–160.

[19] M. Foucault, M. Palyart, X. Blanc, G. C. Murphy, and J.-R. Falleri, "Impact of developer turnover on quality in open-source software," in *Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, 2015, pp. 829–841.

[20] D. Posnett, R. D'Souza, P. Devanbu, and V. Filkov, "Dual ecological measures of focus in software development," in *International Conference on Software Engineering (ICSE)*, 2013, pp. 452–461.

[21] T. Mens, "Evolving software ecosystems a historical and ecological perspective," *Dependable Software Systems Engineering*, vol. 40, p. 170, 2015.

[22] M. Goeminne and T. Mens, "A comparison of identity merge algorithms for software repositories," *Science of Computer Programming*, vol. 78, no. 8, pp. 971–986, 2013.

[23] M. Lungu, M. Lanza, T. Gîrba, and R. Robbes, "The small project observatory: Visualizing software ecosystems," *Science of Computer Programming*, vol. 75, no. 4, pp. 264 – 275, 2010.

[24] J. Pérez, R. Deshayes, M. Goeminne, and T. Mens, "SECONDA: Software ecosystem analysis dashboard," in *European Conference on Software Maintenance and Reengineering (CSMR)*, March 2012, pp. 527–530.

[25] F. W. Santana and C. M. L. Werner, "Towards the analysis of software projects dependencies: An exploratory visual study of software ecosystems." in *International Workshop on Software Ecosystems (IWSECO)*, ser. CEUR Workshop Proceedings, vol. 987, 2013, pp. 7–18.

[26] M. Claes, T. Mens, and P. Grosjean, "maintaineR: A web-based dashboard for maintainers of CRAN packages," in *International Conference on Software Maintenance and Evolution (ICSME)*, 2014, pp. 597–600.

[27] G. Bavota, G. Canfora, M. Di Penta, R. Oliveto, and S. Panichella, "How the Apache community upgrades dependencies: an evolutionary study," *Empirical Software Engineering*, vol. 20, no. 5, pp. 1275–1317, 2015.

[28] J. Businge, A. Serebrenik, and M. van den Brand, "Survival of Eclipse third-party plug-ins," in *International Conference on Software Maintenance (ICSM)*, 2012, pp. 368–377.

[29] D. Dhungana, I. Groher, E. Schludermann, and S. Biffl, "Software ecosystems vs. natural ecosystems: Learning from the ingenious mind of nature," in *European Conference on Software Architecture: Companion Volume (ECSA)*, 2010, pp. 96–102.

[30] F. Thung, T. F. Bissyande, D. Lo, and L. Jiang, "Network structure of social coding in GitHub," in *European Conference on Software Maintenance and Reengineering (CSMR)*, 2013, pp. 323–326.

[31] P. C. Rigby, Y. C. Zhu, S. M. Donadelli, and A. Mockus, "Quantifying and mitigating turnover-induced knowledge loss: Case studies of Chrome and a project at Avaya," in *International Conference on Software Engineering (ICSE)*, 2016, pp. 1006–1016.

[32] E. Wittern, P. Suter, and S. Rajagopalan, "A look at the dynamics of the JavaScript package ecosystem," in *International Conference on Mining Software Repositories (MSR)*, 2016, pp. 351–361.

[33] D. M. German, B. Adams, and A. E. Hassan, "The evolution of the R software ecosystem," in *European Conference on Software Maintenance and Reengineering (CSMR)*, March 2013, pp. 243–252.

[34] A. Decan, T. Mens, M. Claes, and P. Grosjean, "When GitHub meets CRAN: An analysis of inter-repository package dependency problems," in *International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, March 2016, pp. 493–504.

[35] J. Aué, M. Haisma, K. F. Tómasdóttir, and A. Bacchelli, "Social diversity and growth levels of open source software projects on GitHub," in *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2016, pp. 41:1–41:6.

[36] S. Syed and S. Jansen, "On clusters in open source ecosystems," in *International Workshop on Software Ecosystems (IWSECO)*, 2013.

[37] A. MacCormack, C. Baldwin, and J. Rusnak, "Exploring the duality between product and organizational architectures: A test of the "mirroring" hypothesis," *Research Policy*, vol. 41, no. 8, pp. 1309 – 1324, 2012.

[38] L. J. Colfer and C. Y. Baldwin, "The mirroring hypothesis: Theory, evidence and exceptions," Harvard Business School, Tech. Rep. Finance Working Paper No. 16-124, May 2016.

[39] M. Cataldo, J. D. Herbsleb, and K. M. Carley, "Socio-technical congruence: A framework for assessing the impact of technical and work dependencies on software development productivity," in *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2008, pp. 2–11.

[40] D. A. Tamburri, P. Kruchten, P. Lago, and H. van Vliet, "Social debt in software engineering: insights from industry," *Journal on Internet Services and Applications*, vol. 6, no. 1, pp. 1–17, 2015.

[41] A. Decan, T. Mens, and M. Claes, "An empirical comparison of dependency issues in OSS packaging ecosystems," in *International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Feb. 2017.