An Ecosystemic and Socio-Technical View on Software Maintenance and Evolution

(Invited Paper)

Tom Mens
Software Engineering Lab, COMPLEXYS and INFORTECH Research Institutes
University of Mons, Belgium
Email: tom.mens@umons.ac.be

***Abstract*—In this invited paper I focus on the difficulties of maintaining and evolving software systems that are part of a larger ecosystem. While not every software system falls under this category, software ecosystems are becoming ubiquitous due to the omnipresence of open source software. I present several challenges that arise during maintenance and evolution of software ecosystems, and I argue how some of these challenges should be addressed by adopting a socio-technical view and by relying on a multidisciplinary and mixed methods research approach. My arguments are accompanied by an extensive, though unavoidably incomplete, set of references to the state-of-the-art research in this domain.**

***Keywords*-software ecosystem; socio-technical network; interdisciplinary research; mixed methods research; collaborative software engineering; empirical software engineering**

I. Introduction

This paper presents some insights I have gained through the empirical research that I have been conducting on software ecosystems maintenance in the context of an interdisciplinary project ECOS ("Ecological Studies of Open Source Software Ecosystems", 2012-2017), as well as through some recent results that have been reported by other researchers in the domain of software ecosystem evolution.

In 2015 we conducted a survey on the open challenges in software ecosystem research [1]. Twenty-six respondents (researchers active in the field) answered open questions related to the current research, future trends, lack of tooling and specific challenges that software ecosystem researchers are confronted with. Focusing on software evolution specifically, I will report on some of the identified challenges in more detail.

Many researchers have proposed different definitions of software ecosystems. A software ecosystem is often studied from a business perspective [2]. For example, Jansen et al. [3] define a software ecosystem as "a set of businesses functioning as a unit and interacting with a shared market for software and services, together with the relationships among them." From the point of view of software evolution, however, a more technical perspective seems more adequate. Lungu [4] therefore defined a software ecosystem as "a collection of software projects that are developed and evolve together in the same environment". Manikas [5] combined all these perspectives into a single encompassing definition of a software ecosystem as "the interactions of a set of actors on top of a common technological platform that results in a number of software solutions or services. Each actor is motivated by a set of interests or business models and connected to the rest of the actors and the ecosystem as a whole with symbiotic relationships, while the technological platform is structured in a way that allows the involvement and contribution of the different actors." The latter definition will be adopted in this article.

The remainder of this article is structured as follows. Section II starts by presenting some well-known examples of important problems that can arise during software ecosystem evolution. Section III claims that, in order to understand and cope with these problems, one needs to adopt a socio-technical view of the ecosystem. Section IV justifies the use of an interdisciplinary and mixed methods approach to analyse software ecosystems and to propose solutions that facilitate evolution and increase maintainability. Section V presents what I believe to be the most important challenges in software ecosystems research. Finally, Section VI concludes.

II. When things go wrong

Complicated and changing dependencies are a burden for many developers [6], and are sometimes referred to as the "dependency hell". As will be illustrated by several examples throughout this section, many large and very well-known software ecosystems (e.g., CRAN, npm, Eclipse, Debian, Gentoo and Gnome) have experienced important maintainability problems at some point in their lifetime.

Through interviews conducted with contributors of the CRAN ecosystem, we observed that developers are struggling with backward-incompatible updates of packages they depend on [7]. Bogart et al. explored similar problems related to breaking changes through interviews with contributors of Eclipse, CRAN and npm [8].

Some striking problems have been reported for the JavaScript package ecosystem, and more specifically for the packages of its runtime environment Node.js, managed by the npm package manager [9]. Since its creation in 2009, npm has grown very rapidly, hosting more than 240K packages in August 2016. Some of these packages are required by many other packages, sometimes in an extreme way. For example, the package isarray[1] essentially contains three lines of code:

[1] https://www.npmjs.com/package/isarray

978-1-5090-3806-0/16 $31.00 © 2016 IEEE
DOI 10.1109/ICSME.2016.19

```
var toString = {}.toString;
module.exports = Array.isArray || function (arr) {
  return toString.call(arr) == '[object Array]';
  };
```

Yet, over 150 packages still depend on it (counted in August 2016). Related problems to dependencies have been criticized by David Haney in his blog[2]. An even more severe example is the package leftpad, which essentially contains a few lines of source code but has thousands of dependent projects, including Node and Babel. When its developer decided to unpublish all his modules for npm, this had important consequences, “almost breaking the internet”[3].

The R ecosystem, a very popular and rapidly evolving open source environment for statistical computing [10], relies on a central gatekeeper system called CRAN[4]. It is steadily growing and contains almost 9,000 packages (counted in August 2016). It has been criticized of becoming too large [11] and suffers from problems with its dependency management system [12]. An additional difficulty arises due to the increasing use of GitHub for distributing packages that are not available on CRAN [13], [14]. Empirical studies confirm these maintainability problems [15], and automated solutions are being proposed.

Gentoo, one of the open source Linux distributions, is another example of a popular ecosystem that has witnessed important problems during its evolution history. Zanetti et al. studied the social organisation structure, by analyzing the collaboration structure and dynamics of Gentoo’s bug tracking system over a period of ten years [16]. An increasing centralisation towards a single central contributor, followed by an unexpected departure of this contributor, caused a major disruption in the community’s bug handling performance. This case study reveals that, next to analyzing the technical aspects of an ecosystem (such as its package dependencies), it is equally important to address the social aspects.

The expectations, values, tooling and (versioning) policies suggested or imposed by the community may differ significantly from one ecosystem to another. In addition, ecosystem contributors may not necessarily be fully aware of these practices or do not properly adhere to them. For example, in a study of the Maven Central Repository, Raemakers et al. observed that the recommended semantic versioning policy[5] is not always adhered to, causing many unintended breaking changes [17]. A similar observation was made by Businge et al. for Eclipse third-party plug-ins [18]. It is therefore important to make the “community values and accepted trade-offs explicit and transparent in order to resolve conflicts and negotiate change-related costs” [8].

Yet, the degree to which a community is willing to cope with maintainability issues and breaking changes is highly ecosystem dependent. The Eclipse ecosystem, for example, aims for long-term stability and preservation of backward compatible changes. While imposing a significant burden on its developers, this policy is supported by the Eclipse API Tools component[6] that reduces the chance of accidental breaking changes. npm, on the other hand, does not have a central gatekeeper and aims for rapid evolution. Its developers “are less concerned about breaking changes as long as they are signalled clearly through version numbering” [8]. Tools like Greenkeeper[7] help identify and reduce the impact of breaking changes.

While solutions to analyse and reduce the impact of dependencies and changes are often *technical* in the form of automated tools, *social* solutions may be equally effective. For example, de Souza et al. present several strategies through which software developers manage dependencies in their software projects [19]. One of these strategies consists of giving specific team members the task of communicating and synchronizing with external developers that provide software components to the team.

A challenge of a different nature is how to perform a major upgrade or migration of the software ecosystem itself, while limiting the negative impact of such a major disturbance of the ecosystem. A possible strategy would be to have both versions of the ecosystem co-exist during the migration (as is the case with Python 3.0, the first ever intentionally backwards incompatible Python release that continues to co-exist with Python 2 until 2020 [8]). Sometimes, however, major upgrades may cause disillusions in the ecosystem community. This was the case for the Gnome 3 Linux desktop environment (a replacement of Gnome 2) that caused many Linux users to stop using Gnome and choose Xfce as an alternative[9].

III. A SOCIO-TECHNICAL VIEWPOINT

From the case studies presented in Section II it should be clear that providing proper automated support for the maintainability of software ecosystems is not trivial, and needs to consider both the *technical* and the *social* dimension. To fully understand the dynamics of a software ecosystem, one should therefore consider a software ecosystem as a kind of *socio-technical* network, which is a graph structure featuring two types of nodes: the contributors (people) to the ecosystem, and the technical artefacts (e.g., software packages, documentation, bug reports, patches) produced by these contributors. Such a graph will have dependencies between all types of nodes: contributors will communicate or collaborate with other contributors, technical artefacts may depend on each other, and contributors will produce or modify technical artefacts.

Socio-technical congruence

As early as 1968 already, Melvin Conway [20] hypothesized that “Any organization that designs a system (defined broadly) will produce a design whose structure is a copy

[2] http://www.haneycodes.net/npm-left-pad-have-we-forgotten-how-to-program/

[3] http://uk.businessinsider.com/npm-left-pad-controversy-explained-2016-3

[4] https://cran.r-project.org

[5] See http://semver.org for more details.

[6] http://www.eclipse.org/pde/pde-api-tools/

[7] https://greenkeeper.io

[8] http://python3porting.com/strategies.html

[9] http://www.pcworld.com/article/2691192

of the organization's communication structure."[10] Cataldo et al. introduced the concept of *socio-technical congruence* to reflect the alignment between the technical dependencies and the social coordination in a project [21]. In a company setting, they investigated its impact on developer productivity [21] and failure-proneness of the software system [22]. Kwan et al. [23] analyzed its effect on software build success. McCormack et al. introduced the term *mirroring hypothesis* to reflect the socio-technical alignment, and also found empirical evidence for software developed by individual companies [24]. However, together with [25] they also observed that collaborative open source software projects appear to follow an approach that is more modular and decoupled from the organisational structure. One of the reasons for this seems to be that "digital technologies make possible new modes of coordination that enable groups to deviate from classical mirroring as seen within firms." Further studies remain needed on whether or not socio-technical congruence can be observed in collaborative open source software ecosystems, and whether or not this is beneficial in any way.

Socio-technical software development networks tend to have a *dual nature*: techniques or tools that are defined or used to understand or support the *technical* dependency network often also make sense for the *social* dependency network, and vice versa. Let me illustrate this by means of two concrete examples.

Technical versus social debt

At the technical side, the notion of *technical debt* (coined by Cunningham [26]) is a well-known concept advocated by the agile development community. It can be defined as "a concept in programming that reflects the extra development work that arises when code that is easy to implement in the short run is used instead of applying the best overall solution". Software refactoring techniques can be used to reduce the technical debt. The SQALE quality model and method aims to manage technical debt, and tool support is commonly available.[11] Some ecosystems, like Eclipse, are more subject to technical debt, because of their strong requirement to preserve backward compatibility. This requirement is not always easy to reconcile with the need to accommodate new changes rapidly.

The analogy with *technical debt* would be *social debt*. Tamburri et al. define it as the "unforeseen project cost connected to sub-optimal organisational-social structures" [27]. In a company setting, they identified a set of *community smells*, or socio-technical anti-patterns that may be indicators of social debt. Examples are the lack of communication between teams, egocentric or unresponsive team members, and problems related to differences in culture or experience. Just like refactoring techniques help in reducing the technical debt, reorganisations in the social and organisational structure would help in reducing the social debt. It should, however, be clear that social debt and technical debt cannot be seen in isolation. Due to the intricate socio-technical relationships, and the possible presence of a socio-technical congruence, social debt is likely to have an impact on technical debt and vice versa.

Social debt not only arises in individual software projects, but also in software ecosystems. The types of smells that may be indicators of social debt are likely to be different for such highly collaborative systems. For example, the Gentoo case mentioned in Section II witnessed an increased reliance on a single central contributor, combined with a "disruptive social environment" and "increased bureaucracy" [16].

Social versus technical bus factor

The so-called bus factor (a.k.a. truck factor) is a well-known, but understudied, concept in collaborative software development. It refers to the number of team members that need to be "run over by a bus" before the project gets in trouble. In extreme cases, the loss of a single key person may lead to serious problems, as was illustrated in Section II for the Gentoo case [16]. Cosentino et al. proposed a tool to automatically measure the bus factor of projects stored in Git repositories [28]. Avelino et al. proposed and validated a new way of measuring the bus factor [29]. The risks related to developer turnover can be mitigated by resorting to techniques such as pair programming and shared code ownership. Rigby et al. proposed to identify successors (and involve them as co-owners) to reduce the risk associated to developer turnover [30]. They also presented a method to predict the developer with the most related expertise as a successor. Garcia et al. proposed a machine learning classifier for measuring online contributor motivation to predict contributor turnover [31].

The technical equivalent of the social bus factor phenomenon would refer to serious problems in a project because of the unexpected removal (getting "run over by a bus") of technical artefacts. Several cases of this were already presented in Section II. For example, the removal of the package leftpad from npm caused a massive amount of breaking dependencies. Incidentally, the cause of this removal was another instance of the social bus factor, since the owner of leftpad abandoned npm after having removed all his packages.

IV. Interdisciplinary research

Researchers have embraced the intrinsic socio-technical nature of software ecosystems, and many empirical analyses consider both the technical and social factors. These research results often have an interdisciplinary nature, drawing inspiration or using techniques originating from other domains.

Complex socio-technical networks

Several researchers have analyzed the presence and evolution of *complex network properties* in the socio-technical networks of software projects and ecosystems [32]–[35]. Such properties appear to *emerge* as a side effect of the development

[10] Quote taken from http://www.melconway.com/Home/Conways_Law.html

[11] See www.sqale.org. SQALE is implemented as a plug-in of SonarQube.

process, without any particular software design principle explicitly dictating their presence. Examples of such properties are the presence of *power law behavior*, a *scale-free* network topology[12] and a *small-world* structure[13]. Myers and Concas et al. have proposed models attempting to explain the process leading to complex network properties [32], [34]. It still remains an open challenge to come up with explanatory models that most naturally resemble the collaborative development process that one can observe in socio-technical networks of software ecosystems.

Social network analysis (SNA) techniques [36] have been used frequently to study open source software developer communities [37], [38]. A well-known success story is the use of SNA to improve prediction of software failures. For example, Pinzger et al. [39] empirically investigated the effect of developer contribution fragmentation on the number of post-release failures. They applied so-called *centrality measures* to the socio-technical network, and found that more central modules are more failure-prone. The closeness centrality measure turned out to be the most significant for predicting the number of post-release failures. Bird et al. [40] did something similar, by *combining* technical network properties (which software components have dependencies on others) with social network properties (who has worked on which components and how much). The reasoning behind this is that the social and technical aspects interact to influence the quality of the resulting software. The combined use of technical and social properties was found to predict software failures more accurately than when using each set of properties in isolation.

In a tutorial, Madey proposed to borrow techniques from *complex systems theory* to study large-scale software development [41]. This theory provides a scientific framework to study problems that are difficult to solve and systems that are hard to understand because the causes and effects are not obviously related. In particular, he proposed to consider a software (eco)system as a *complex adaptive system*, i.e., "a system that exhibits behaviors arising from non-linear spatio-temporal interactions among a large number of components and subsystems." Next to SNA, a wide portfolio of techniques and associated tools could be used to study such systems: dynamical systems theory, cellular automata and agent-based simulation.

Ecological, economical and other statistical measures

Ecological modelling techniques have been proposed based on an analogy between natural ecosystems and software ecosystems. In [42] I explored this analogy in more detail. *Ecological measures* have been proposed to study socio-technical networks of software ecosystems. In ecological research, a wide range of *diversity metrics* have been proposed to measure the biodiversity of species belonging to the ecosystem. Examples are Piélou's index (a.k.a. species evenness), Shannon diversity (a.k.a. entropy), and Simpson's diversity index. I have provided an overview of these diversity indices and their application to software ecosystems in [43]. Posnett et al. [44] used the notion of relative entropy (a.k.a. Kullback-Liebler divergence) to define and measure the attention focus (of developers and software modules, respectively) and its effect on defects found in modules. They observed that more focused developers introduce fewer defects, while files that receive narrowly focused activity are more likely to contain defects than other files. *Linguistic diversity* in software projects and the related risk of using a programming language was explored by Vasilescu et al. [45]. The same authors also used diversity measures to measure *gender diversity* and *tenure diversity* in GitHub teams [46]. For gender diversity they used the Blau index (a variant of the Simpson index); for tenure diversity they relied on the coefficient of variation. They observed that increased diversity was associated with greater productivity. Unfortunately, however, women still remain underrepresented in most open source software ecosystems.

Next to the use of diversity measures, *econometric indices* (used to assess the inequalities in a distribution) have been borrowed from research in economy to study socio-technical characteristics of software ecosystems [47], [48]. As an example, Vasilescu et al. defined a series of metrics based on the Gini index to assess the degree of specialisation of projects and contributors in the socio-technical network of the Gnome software ecosystem [49]. Among many other results, they observed the presence of two quite distinct subcommunities of contributors (essentially, coders and translators), with different activity patterns and different needs. This implies that, in order to provide adequate tool support for an ecosystem community, it is important to identify and support the specific needs of its subcommunities. It could even be useful to provide personalized support for individual ecosystem contributors, based on an automatic identification of their specific contributor profiles (in terms of their specific activity, communication and contribution patterns).

An example of a statistical technique that has been borrowed from another scientific discipline is *survival analysis* [50]. It originates from biomedical sciences where it is used to study factors affecting the time to death of patients or laboratory animals. More broadly, it is also widely used in social sciences to analyse the *time to event* for a variety of events (such as child birth, switching employment, marriage or divorce, etc.) Survival analysis uses concepts like *censoring* to deal with observations that suffer from incomplete information (such as dropout of subjects during the study). In the context of software ecosystems, Samoladas et al. applied this technique to assess the expected duration of open source software projects [51]. After partitioning projects by type or domain, they used Kaplan-Meier estimations of the survival functions to compare the survival of projects belonging to different domains. They also observed that survivability increases as projects become larger.

[12] A distribution is considered *scale-free* if it appears the same whatever the scale of the observation.

[13] This means that the average path length between any two nodes is very small, and there is a large amount of clusters.

Natural language processing

Originating from the research domain of natural language processing (NLP), an emerging technique for studying socio-technical aspects of software ecosystems is the use of *sentiment analysis* [52]. Applying sentiment analysis techniques to the domain of software ecosystems requires tuning the tools and lexicons to this domain, in order to avoid producing unreliable results [53]. In the context of Apache's JIRA issue tracking system, Ortu et al. observed that human affectiveness (measured by the presence of positive or negative emotions) has an impact on productivity (measured by the time required to fix issues) [54]. In the context of the Gentoo ecosystem, Garcia et al. analyzed the emotions of contributors to the e-mail archives and bug tracker [31]. They observed that contributors are more likely to become inactive when they express strong positive or negative emotions in the bug tracker, or when they deviate from the expected value of emotions in the mailing list. Based on such observations, a Bayesian classifier was proposed to predict the risk of contributors leaving the project.

To summarize, since engineering of software ecosystems is to a large extent a social activity, a combined focus on human and software engineering aspects is required. This has spawned a thriving new research area called *collaborative software engineering* [55] or *behavioral software engineering* [56].

Analyzing and predicting how software ecosystems evolve over time necessarily requires empirical studies that adopt *mixed methods* research [57]. By involving theories from other disciplines like sociology and psychology, and by combining quantitative methods with qualitative ones (such as user surveys and interviews), a mixed approach increases confidence of the findings, by combining the strengths of the complementary research methods used.

V. Software ecosystem challenges revisited

This section revisits some of the challenges in software ecosystems research previously identified [1]. In line with the previous sections, we will mainly explore them from a socio-technical viewpoint.

The main challenge for any software community resides in *how to design and structure* its ecosystem in such a way that it fosters a lively community, while at the same time maintaining the quality, stability and reliability of the ecosystem. We have observed in Section II that proposed solutions tend to be very *ecosystem-specific*. They will depend on the values and habits of its community members, but they will also depend on the specific tools used by the community. For example, the decision to choose GitHub as the main repository hosting service will inevitably impact the way in which developers collaborate. Indeed, GitHub favours (but does not impose) a pull-based development process that comes with its specific advantages and shortcomings [58], [59]. The analysis of software development data mined from GitHub repositories should also consider the many perils reported by Kalliamvakou et al. [60].

In many cases, the boundaries of a software ecosystem are ill-defined. Some ecosystems naturally *emerge* and grow over time. This phenomenon was observed by Blincoe et al. when studying GitHub-hosted projects [61]. They proposed a method to identify ecosystems within GitHub based on the technical dependencies across projects, and found that most identified ecosystems were clustered around a single project. In addition to this, these ecosystems tend to be interconnected. Automated tool support to *detect and support such "ad hoc" emerging ecosystems* is highly desirable. A better understanding the socio-technical network of such ecosystems is also required in order to understand how ecosystems emerge and become more or less popular and successful over time.

One a software ecosystem has been established (or has naturally emerged), the obvious next challenge resides in *how to evolve and maintain* it. This challenge has been discussed at length in previous sections of this paper. The focus was mainly on managing breaking changes between dependent software components belonging to the same ecosystem, but also on how to migrate an ecosystem to a new major release.

Comparing software ecosystems in order to understand their differences is a challenge that remains wide open. As observed by Vasilescu et al. [49], such differences even arise between different subcommunities of the same ecosystem. Being aware of these differences can be valuable for all contributors of the ecosystem. For researchers, it is important to study how the specificities of each ecosystem affect its maintainability and evolution.

Big data

From a research point of view, undoubtedly the most important is the *big data* challenge [62]. Traditionally, this challenge is broken up into four dimensions, commonly referred to as the 4 V's of big data: Volume, Variety, Veracity and Velocity.[14]

With respect to the first V, software ecosystems contain a huge *volume* of data (often in the range of terabytes) that is worthwhile to be analyzed. Because of this, software ecosystem analysis faces similar issues as big data in many other scientific domains.

Software ecosystem researchers also need to cope with the *variety* or heterogeneity of data sources, that can be either structured (e.g. programs), semi-structured (e.g. e-mails) or unstructured (e.g. unformatted texts). Examples illustrating the wide variety of data sources used include version control repositories (containing the source code and other software artefacts), issue trackers (containing bug reports, feature requests and how they have been addressed), mailing lists (capturing communication between members of the ecosystem community, e.g. developers and users), Q&A websites (answering questions raised by developers or users), Twitter communication, surveys and interviews. Socha et al. even go a step further by proposing wide-field ethnography, combining data from a wide variety of sensors such as video recordings,

[14] www.ibmbigdatahub.com/infographic/four-vs-big-data

screen capture software, audio recordings, photographs and field notes of software developers collaborating in situ [63].

The third big data dimension concerns its *veracity*. The data that needs to be analyzed is often partially incomplete, uncertain or inconsistent. For example, Bruntink reported on a series of problems related to implausible, inconsistent or missing data, which he encountered while analyzing Ohloh, a large-scale on-line index and analytics platform for open source projects [64]. Similar issues can be observed in other data sources.

Velocity, the final V, is also an issue in some cases, with new data appearing faster than can be processed. As an example, new commits are made to GitHub several times every second. This may be less of an issue for empirical studies, in which the data is typically analyzed off-line. For automated tools that support the activities of a software ecosystem community (e.g. web-based dashboards), however, it may be important to rely in the most recent data in order to make informed decisions.

Big data can also benefit from techniques such as *deep learning* algorithms to open up new opportunities for improved understanding of, and better support for, evolving software ecosystems. A first step in the direction of deep learning software repositories was proposed by White et al. [65]. Wang et al. applied deep learning for improved software defect prediction [66], and Corley et al. used it for improving developer effectiveness in feature location [67].

Privacy versus Reproducibility

Another very important issue is *preserving privacy*. Software ecosystem data typically contains information about contributors involved in the development and maintenance of components of the ecosystem. Researchers are interested in such data, for example to understand how the expertise and work patterns of individual contributors may affect the software ecosystem as a whole. Techniques such as identity merging have been proposed to facilitate this analysis [68], [69]. It is, however, imperative to preserve privacy of those individuals. For example, in May 2016, a new EU directive for the protection of personal data has been published, giving citizens more control over of the use of their personal data. Persons or organisations collecting and managing personal information must protect it from misuse and must respect certain rights of the data owners which are guaranteed by EU law. Satisfying these laws can be quite challenging, since it is technically possible to infer personal information (such as gender, personality traits, and even beliefs and perceptions) from data gathered from software ecosystems (such as e-mail correspondence). Therefore, appropriate techniques need to be developed and put into place to guarantee anonymity. Fung et al. presented a survey of research results and future directions in privacy-preserving data publishing [70]. Malik et al. provided an overview of privacy-preserving data mining tools and techniques, and proposed future research directions [71].

It remains a challenge to reconcile the need to preserve privacy with the needs of researchers to make their data and results openly accessible to other researchers for *reproducibility* purposes [72], [73]. A directly related practical challenge is the difficulty of *sharing this data*, since a variety of different formats, tools, and operating systems are used, and because of the storage problems due to the fact that we are dealing with huge amounts of data. Partial solutions to these problems are being proposed by the research community, such as the use of Docker containers [74] or dedicated experimental software engineering research workbenches such as TraceLab [75].

VI. Conclusion

In this paper, I explored the active research domain of software ecosystems, and in particular the challenges of their maintenance and evolution. Typical problems relate to breaking software components due to backward incompatible updates in their dependencies, and difficulties to perform major upgrades of the ecosystem as a whole because of all these interdependencies. Every software ecosystem has its own specificities, requiring customized tools to address these problems.

Nevertheless, empirical research reveals that, in order to provide such solutions, it is important to consider not only the technical aspects of the ecosystem dynamics but also the social or human aspects of its community of contributors. This socio-technical viewpoint opens up a new domain of collaborative software engineering research, combining ideas from traditional software engineering research with those from computer-supported collaborative work.

In addition to this, empirical research on the socio-technical dynamics of software ecosystems is becoming increasingly interdisciplinary, borrowing techniques and ideas from many other scientific disciplines. Examples include the use of social network analysis techniques from social sciences, the use of diversity metrics borrowed from ecology, the adoption of econometric indices used in economy, the use of survival analysis techniques coming from medial sciences, and many more. The socio-technical nature of software ecosystems also requires a mixed methods research approach, combining quantitative empirical methods with more qualitative ones such as those used in sociology and psychology.

While socio-technical research on software ecosystem evolution is very active, as can be witnessed by the many recent references cited in this paper, I have identified many remaining open challenges. It is my hope that researchers in this field will be inspired by this article to take up some of these challenges, and as such further advance this important field of research.

Acknowledgment

This research was carried out in the context of ARC research project AUWB-12/17-UMONS- 3. I would like to thank my research collaborators Bogdan Vasilescu, Alexander Serebrenik, Alexandre Decan, Maelick Claes and Mathieu Goeminne for very useful feedback on an earlier version of this article.

References

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

[2] S. Jansen, M. Cusumano, and S. Brinkkemper, Eds., *Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry*. Edward Elgar, 2013.

[3] S. Jansen, A. Finkelstein, and S. Brinkkemper, "A sense of community: A research agenda for software ecosystems," in *Int'l Conf. Software Engineering*, May 2009, pp. 187–190.

[4] M. Lungu, "Towards reverse engineering software ecosystems," in *Int'l Conf. Software Maintenance*, 2008, pp. 428–431.

[5] K. Manikas and K. M. Hansen, "Software ecosystems: A systematic literature review," *J. Systems and Software*, vol. 86, no. 5, pp. 1294–1306, May 2013.

[6] C. Artho, K. Suzaki, R. D. Cosmo, R. Treinen, and S. Zacchiroli, "Why do software packages conflict?" in *Int'l Conf. Mining Software Repositories*, Jun. 2012, pp. 141–150.

[7] T. Mens, "Anonymized e-mail interviews with R package maintainers active on CRAN and GitHub," University of Mons, Tech. Rep., 2015. [Online]. Available: http://arxiv.org/abs/1606.05431

[8] C. Bogart, C. Kästner, J. Herbsleb, and F. Thung, "How to break an API: Cost negotiation and community values in three software ecosystems," in *Int'l Symp. Foundations of Software Engineering*, 2016.

[9] E. Wittern, P. Suter, and S. Rajagopalan, "A look at the dynamics of the JavaScript package ecosystem," in *Int'l Conf. Mining Software Repositories*. ACM, 2016, pp. 351–361.

[10] D. M. Germán, B. Adams, and A. E. Hassan, "The evolution of the R software ecosystem," in *European Conf. Software Maintenance and Reengineering*, 2013, pp. 243–252.

[11] K. Hornik, "Are there too many R packages?" *Austrian Journal of Statistics*, vol. 41, no. 1, pp. 59–66, 2012.

[12] J. Ooms, "Possible directions for improving dependency versioning in R," *R Journal*, vol. 5, no. 1, pp. 197–206, Jun. 2013.

[13] A. Decan, T. Mens, M. Claes, and P. Grosjean, "On the development and distribution of R packages: An empirical analysis of the R ecosystem," in *European Conference on Software Architecture Workshops*, 2015, pp. 41:1–41:6.

[14] ——, "When GitHub meets CRAN: An analysis of inter-repository package dependency problems," in *Int'l Conf. Software Analysis, Evolution, and Reengineering*. IEEE, Mar. 2016, pp. 493–504.

[15] M. Claes, T. Mens, and P. Grosjean, "On the maintainability of CRAN packages," in *Int'l Conf. Software Maintenance, Reengineering, and Reverse Engineering*. IEEE, 2014, pp. 308–312.

[16] M. S. Zanetti, I. Scholtes, C. J. Tessone, and F. Schweitzer, "The rise and fall of a central contributor: Dynamics of social organization and performance in the Gentoo community," in *Int'l Workshop on Cooperative and Human Aspects of Software Engineering*, May 2013, pp. 49–56.

[17] S. Raemaekers, A. van Deursen, and J. Visser, "Semantic versioning versus breaking changes: A study of the Maven repository," in *Working Conf. Source Code Analysis and Manipulation*, Sept 2014, pp. 215–224.

[18] J. Businge, A. Serebrenik, and M. G. J. van den Brand, "Eclipse API usage: the good and the bad," *Software Quality*, vol. 23, no. 1, pp. 107–141, 2015.

[19] C. R. B. de Souza and D. F. Redmiles, "An empirical study of software developers' management of dependencies and changes," in *Int'l Conf. Software Engineering*. ACM, 2008, pp. 241–250.

[20] M. Conway, "How do committees invent?" *Datamation Journal*, pp. 28–31, April 1968.

[21] 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 *Int'l Symp. Empirical Software Engineering and Measurement*. ACM , 2008, pp. 2–11.

[22] M. Cataldo, A. Mockus, J. A. Roberts, and J. D. Herbsleb, "Software dependencies, work dependencies, and their impact on failures," *IEEE Transactions on Software Engineering*, vol. 35, no. 6, pp. 864–878, Nov 2009.

[23] I. Kwan, A. Schroter, and D. Damian, "Does socio-technical congruence have an effect on software build success? A study of coordination in a software project," *IEEE Trans. Soft. Eng.*, vol. 37, no. 3, pp. 307–324, May 2011.

[24] 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.

[25] 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.

[26] W. Cunningham, "The WyCash portfolio management system – experience report," in *OOPSLA '92*, Mar. 1992.

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

[28] V. Cosentino, J. L. C. Izquierdo, and J. Cabot, "Assessing the bus factor of Git repositories," in *Int'l Conf. Software Evolution, Analysis and Reengineering*. IEEE, Mar. 2015, pp. 499–503.

[29] G. Avelino, L. Passos, A. Hora, and M. T. Valente, "A novel approach for estimating truck factors," in *Int'l Conf. Program Comprehension*, 2016.

[30] 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 *Int'l Conf. Software Engineering*. ACM , 2016, pp. 1006–1016.

[31] D. Garcia, M. S. Zanetti, and F. Schweitzer, "The role of emotions in contributors activity: A case study on the Gentoo community," in *Int'l Conf. Cloud and Green Computing*, Sept 2013, pp. 410–417.

[32] C. R. Myers, "Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs," *Physical Review E*, vol. 68, no. 4, 2003.

[33] A. Potanin, J. Noble, M. Frean, and R. Biddle, "Scale-free geometry in oo programs," *Commun. ACM*, vol. 48, no. 5, pp. 99–103, May 2005.

[34] G. Concas, M. Marchesi, S. Pinna, and N. Serra, "Power-laws in a large object-oriented software system," *IEEE Trans. Soft. Eng.*, vol. 33, no. 10, pp. 687–708, 2007.

[35] P. Louridas, D. Spinellis, and V. Vlachos, "Power laws in software," *ACM Trans. Software Engineering and Methodology*, vol. 18, no. 1, pp. 1–26, Oct. 2008.

[36] J. Scott, *Social Network Analysis*. SAGE, 2012.

[37] G. Madey, V. Freeh, and R. Tynan, "The open source software development phenomenon: An analysis based on social network theory," in *Americas Conf. Information Systems*, 2002.

[38] J. Xu, Y. Gao, S. Christley, and G. Madey, "A topological analysis of the open source software development community," in *Hawaii Int'l Conf. System Sciences*, 2005.

[39] M. Pinzger, N. Nagappan, and B. Murphy, "Can developer-module networks predict failures?" in *Int'l Symp. Foundations of Software Engineering*. ACM, 2008, pp. 2–12.

[40] C. Bird, N. Nagappan, H. Gall, B. Murphy, and P. Devanbu, "Putting it all together: Using socio-technical networks to predict failures," in *Int'l Symp. Software Reliability Engineering*. IEEE Computer Society, 2009, pp. 109–119.

[41] G. Madey and S. Kaisler, "[Tutorial:] Complex Adaptive Systems: Emergence, self-organization, tools, analysis and case studies," in *Hawaii Int'l Conf. on System Sciences*, 2009.

[42] T. Mens, M. Claes, P. Grosjean, and A. Serebrenik, "Studying evolving software ecosystems based on ecological models," in *Evolving Software Systems*. Springer, 2014, pp. 297–326.

[43] T. Mens, "Evolving software ecosystems A historical and ecological perspective," in *Dependable Software Systems Engineering*, ser. NATO Science for Peace and Security Series, D: Information and Communication Security. IOS Press, 2015, vol. 40, pp. 170–192.

[44] D. Posnett, R. D'Souza, P. Devanbu, and V. Filkov, "Dual ecological measures of focus in software development," in *Int'l Conf. Software Engineering*. IEEE, 2013, pp. 452–461.

[45] B. Vasilescu, A. Serebrenik, and M. G. J. van den Brand, "The Babel of software development: Linguistic diversity in open source," in *Int'l Conf. Social Informatics*. Springer, 2013, pp. 391–404.

[46] 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 *Int'l Conf. Human Factors in Computing Systems*. ACM, 2015, pp. 3789–3798.

[47] R. Vasa, M. Lumpe, P. Branch, and O. Nierstrasz, "Comparative analysis of evolving software systems using the Gini coefficient," in *Int'l Conf. Software Maintenance*, 2009, pp. 179–188.

[48] E. Giger, M. Pinzger, and H. Gall, "Using the Gini coefficient for bug prediction in Eclipse," in *Int'l Workshop on Principles of Software Evolution*. ACM , 2011, pp. 51–55.

[49] B. Vasilescu, A. Serebrenik, M. Goeminne, and T. Mens, "On the variation and specialisation of workload: A case study of the Gnome ecosystem community," *J. Empirical Software Engineering*, vol. 19, pp. 955–1008, Aug. 2014.

[50] D. G. Kleinbaum and M. Klein, *Survival Analysis: A Self-Learning Text*, 3rd ed. Springer, 2012.

[51] I. Samoladas, L. Angelis, and I. Stamelos, "Survival analysis on the duration of open source projects," *Information & Software Technology*, vol. 52, no. 9, pp. 902–922, 2010.

[52] B. Liu, "Sentiment analysis and opinion mining," *Synthesis Lectures on Human Language Technologies*, vol. 5, no. 1, pp. 1–167, 2012.

[53] N. Novielli, F. Calefato, and F. Lanubile, "The challenges of sentiment detection in the social programmer ecosystem," in *Int'l Workshop on Social Software Engineering*. ACM , 2015, pp. 33–40.

[54] M. Ortu, B. Adams, G. Destefanis, P. Tourani, M. Marchesi, and R. Tonelli, "Are bullies more productive? Empirical study of affectiveness vs. issue fixing time," in *Int'l Conf. Mining Software Repositories*. IEEE, 2015, pp. 303–313.

[55] I. Mistrík, J. Grundy, A. Hoek, and J. Whitehead, Eds., *Collaborative Software Engineering*. Springer, 2010.

[56] P. Lenberg, R. Feldt, and L. G. Wallgren, "Behavioral software engineering: A definition and systematic literature review," *J. Systems and Software*, vol. 107, pp. 15 – 37, 2015.

[57] R. B. Johnson and A. J. Onwuegbuzie, "Mixed methods research: A research paradigm whose time has come," *Educational Researcher*, vol. 33, no. 7, pp. 14–26, Oct. 2004.

[58] G. Gousios, A. Zaidman, M.-A. Storey, and A. van Deursen, "Work practices and challenges in pull-based development: The integrator's perspective," in *Int'l Conf. Software Engineering*. IEEE Press, 2015, pp. 358–368.

[59] G. Gousios, M.-A. Storey, and A. Bacchelli, "Work practices and challenges in pull-based development: The contributor's perspective," in *Int'l Conf. Software Engineering*. ACM, 2016, pp. 285–296.

[60] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining GitHub," in *Int'l Conf. Mining Software Repositories*. ACM, 2014, pp. 92–101.

[61] K. Blincoe, F. Harrison, and D. Damian, "Ecosystems in GitHub and a method for ecosystem identification using reference coupling," in *Int'l Conf. Mining Software Repositories*, 2015.

[62] Y. Demchenko, P. Grosso, C. De Laat, and P. Membrey, "Addressing big data issues in scientific data infrastructure," in *Collaboration Technologies and Systems*, May 2013, pp. 48–55.

[63] D. Socha, R. Adams, K. Franznick, W.-M. Roth, K. Sullivan, J. Tenenberg, and S. Walter, "Wide-field ethnography: Studying software engineering in 2025 and beyond," in *Int'l Conf. Software Engineering*. ACM , 2016, pp. 797–802.

[64] M. Bruntink, "An initial quality analysis of the Ohloh software evolution data," *ECEASST*, vol. 65, 2014. [Online]. Available: http://journal.ub.tu-berlin.de/eceasst/article/view/906

[65] M. White, C. Vendome, M. Linares-Vásquez, and D. Poshyvanyk, "Toward deep learning software repositories," in *Int'l Conf. Mining Software Repositories*. IEEE Press, 2015, pp. 334–345.

[66] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," in *Int'l Conf. Software Engineering*. ACM, 2016, pp. 297–308.

[67] C. S. Corley, K. Damevski, and N. A. Kraft, "Exploring the use of deep learning for feature location," in *Int'l Conf. Software Maintenance and Evolution*, Sep. 2015, pp. 556–560.

[68] 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, Aug. 2013.

[69] E. Kouters, B. Vasilescu, A. Serebrenik, and M. G. J. van den Brand, "Who's who in Gnome: using LSA to merge software repository identities," in *Int'l Conf. Software Maintenance*. IEEE, 2012, pp. 592–595.

[70] B. C. M. Fung, K. Wang, R. Chen, and P. S. Yu, "Privacy-preserving data publishing: A survey of recent developments," *ACM Comput. Surv.*, vol. 42, no. 4, pp. 14:1–14:53, Jun. 2010.

[71] M. B. Malik, M. A. Ghazi, and R. Ali, "Privacy preserving data mining techniques: Current scenario and future prospects," in *Int'l Conf. Computer and Communication Technology*, Nov. 2012, pp. 26–32.

[72] F. J. Shull, J. C. Carver, S. Vegas, and N. Juristo, "The role of replications in empirical software engineering," *J. Empirical Software Engineering*, vol. 13, no. 2, pp. 211–218, 2008.

[73] J. M. González-Barahona and G. Robles, "On the reproducibility of empirical software engineering studies based on data retrieved from development repositories," *J. Empirical Software Engineering*, vol. 17, no. 1-2, pp. 75–89, 2012.

[74] J. Cito, V. Ferme, and H. C. Gall, "Using Docker containers to improve reproducibility in software and web engineering research," in *Int'l Conf. Web Engineering*. Springer, 2016, pp. 609–612.

[75] B. Dit, E. Moritz, M. Linares-Vásquez, D. Poshyvanyk, and J. Cleland-Huang, "Supporting and accelerating reproducible empirical research in software evolution and maintenance using TraceLab Component Library," *J. Empirical Software Engineering*, vol. 20, no. 5, pp. 1198–1236, 2015.