IOS Press (2015), DOI:10.3233/978-1-61499-495-4-170,
NATO Science for Peace and Security Series -
D: Information and Communication Security.
Volume 40: Dependable Software Systems Engineering,
pp. 170-192, Ed: M. Irlbeck, D. Peled, A. Pretschner

PREPRINT

Evolving Software Ecosystems
A Historical and Ecological Perspective

Tom MENS [a,1]

[a] *Software Engineering Lab, University of Mons, Belgium*

Abstract. Software ecosystems are collections of interacting and communicating software projects developed by the same developer community. Dealing with the complex interaction and dynamics of these ecosystems is an important problem that raises additional challenges compared to the maintenance of individual software systems. In particular, the social aspect becomes crucial, requiring to understand evolving ecosystems as dynamic socio-technical networks. Based on software repository mining research advances, in this lecture I explore the state of the art in the empirical analysis of software ecosystems, with a focus on the evolutionary aspect. I present some of the challenges that need to be overcome during the extraction and cleaning of data obtained from different data sources (such as version control repositories, bug trackers, and mailing lists). I draw an analogy between software ecosystems and natural ecosystems, and illustrate through previously reported empirical case studies how ecological diversity metrics and econometric inequality metrics can be used to increase understanding of evolving software ecosystems from dual viewpoints.

It should be noted that this article has been conceived as a work of reference. It does not provide any novel research results by itself, but rather introduces the important concepts and challenges in the field. It also provides many pointers for further reading for those researchers that wish to start working in this important research domain.

Keywords. software evolution, software ecosystem, ecology, diversity, software repository mining, open source software

1. Introduction

If you are new to the research field of software evolution and software ecosystems, or if you want to get a high-level overview of the state-of-the-art in this important branch of software engineering research, then the current lecture is probably a good starting point. I will highlight, from my personal point of view, some major milestones, providing ample references to allow the reader to dive deeper into more technical material. I will also illustrate the approaches by citing numerous empirical case studies that have been carried out by various researchers in the field.

Software evolution has been a topic of research in software engineering since several decades. In the late 1970's, Manny Lehman — the founding father of this research domain — introduced the term "software evolution" as a replacement of the misleading

[1]Corresponding Author: University of Mons, Place du Parc 20, 7000 Mons, Belgium; E-mail: tom.mens@umons.ac.be

term "software maintenance", and postulated the so-called *laws of software evolution* based on empirical evidence on an industrial case study [1,2].

With the increasing importance of *open source software* (OSS) development and *empirical software engineering*, research interest in software evolution has been growing steadily [3,4]. For a state-of-the-art of the ongoing research in this domain, I refer the interested reader to several books with chapters contributed by experts in the field [5,6,7]. The availability of large, freely accessible data repositories containing historical data of evolving software repositories has even spawned a new research domain, called *software repository mining* [8], with a dedicated annual conference MSR that is being organised since 2004.[2]

Initially, the topic of study was the evolution of individual software systems. Recently, however, software evolution has shifted to a *macro-level*, focusing on the evolution of large collections or distributions of software systems [9,10]. As an instance of this trend, research on *software ecosystems* has become very popular, as can be witnessed from the systematic literature review by Manikas and Hansen [11].

Another recent insight is that software systems should be considered as *socio-technical networks* in order to gain a better understanding of the processes that govern their evolution over time [12,13,14]. This means that one should not only take into account the technical artefacts used and produced while maintaining and evolving a software system, but also the social aspects of the humans involved in this process. Indeed, the most crucial asset in maintaining a software system over time is perhaps not the software itself, but the people involved in developing it [15].

Finally, many researchers have started to study the analogies between biology (more specifically, ecology, biological evolution, and the dynamics of natural ecosystems) and software evolution [16,17]. The hope is that these analogies will lead to new insights, mechanisms, principles, and tools to improve software evolution.

In this lecture, I present some research results of empirical studies on open source software ecosystems considered as socio-technical or ecological networks. Some of the long-term goals are: to help individual developers as well as the community as a whole to improve upon their current practices; to improve the quality and resilience of software ecosystems; and to develop models and tools that predict and improve the chances of survival of individual projects within their ecosystem, as well as survival of the ecosystem as a whole.

The remainder of this article is structured as follows. Section 2 introduces the definitions of software ecosystems. Section 3 presents some research goals that have been pursued in empirical research in software ecosystems, and reports on some example case studies for specific ecosystems (see Section 3.1). It also presents some general-purpose and domain-specific tools that have been developed to facilitate the understanding and evolution of software ecosystems (see Section 3.2). Section 4 presents the challenges researchers are confronted with when carrying out empirical studies on evolving software ecosystems. Section 5 talks about biological diversity and software diversity, and presents some metrics that have been used in both domains to measure diversity. Section 6 explains how so-called bipartite graphs can be used to study the socio-technical aspects of a software ecosystem from dual points of view (the project viewpoint and the contributor viewpoint). Section 7 and Section 8 reconsider diversity metrics and eco-

[2]To be precise, the first four editions of MSR (2004 till 2007) were international workshops, and the series became a working conference since 2008.

nomic inequality metrics from both viewpoints. Section 9 concludes and presents some avenues of future work.

2. Software Ecosystem

The term *ecosystem* was originally coined by Roy Clapham in 1930 in the context of *ecology*, to denote the physical and biological components of an environment considered in relation to each other as a unit [18]. In other words, an ecosystem combines all living organisms (e.g., plants, animals, micro-organisms) and physical components (e.g., light, water, soil, rocks, minerals) that interact with one another. The term *software ecosystem* is inspired by natural ecosystems, but unfortunately there is no common definition of what a software ecosystem is.

Bosch et al. introduced the term *software ecosystem* to refer to how software suppliers, vendors, competitors, users, and third-party developers interact [19]. Typical examples of such ecosystems are mobile app stores. Other well-known examples are software product lines. Jansen et al provide a more precise definition of this business-oriented perspective as:

> "A set of actors functioning as a unit and interacting with a shared market for software and services, together with the relationships among them." [20]

Messerschmitt and Szyperski define a software ecosystem as

> "A collection of software products that have some given degree of symbiotic relationships." [21]

Lungu provides a related definition:

> "A collection of software projects which are developed and evolve together in the same environment." [22]

The latter definition is more general than Messerschmitt's since it does not focus on the software *product* alone, but on the software *project* as a whole, comprising not only the technical software artefacts, but also the processes and the people involved in it. Typical examples of ecosystems adhering to this definition are the different Linux distributions (e.g., *Debian*, *Ubuntu*, *Fedora*) and their desktop environments (e.g., *GNOME* and *KDE*), but also package archive networks (e.g., CRAN for *R* packages) and collections of plugins (e.g., for the *Eclipse* software development environment).

Despite the fact that software product families also fall under this definition of software ecosystem, we will not address them specifically in this lecture, since they have some additional specific requirements that make their maintenance and evolution even more challenging [23,24,25,26].

I would like to stress that the notion of *software ecosystem* should not be confused with the notion of *system of systems* (SoS) [27]. The main distinction is that a system of systems (SoS) is a system whose constituents are themselves operationally independent systems that can be considered in isolation. A system of systems was not designed as a whole, and perhaps could never have been, but nevertheless the whole is more than the sum of its parts and may have emergent properties. In an ecosystem, in contrast, many of its constituents depend on one another and therefore cannot function in isolation.

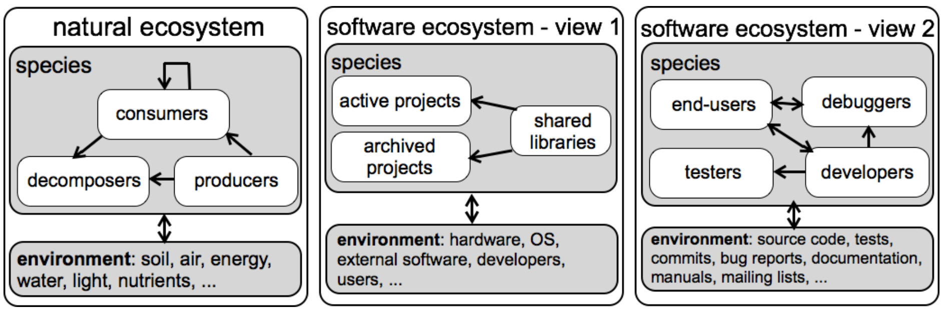

natural ecosystem
species
consumers
decomposers
producers
environment: soil, air, energy, water, light, nutrients, ...
software ecosystem - view 1
species
active projects
shared libraries
archived projects
environment: hardware, OS, external software, developers, users, ...
software ecosystem - view 2
species
end-users
debuggers
testers
developers
environment: source code, tests, commits, bug reports, documentation, manuals, mailing lists, ...

Figure 1. Comparison between a natural ecosystem and two dual views on a software ecosystem.

The notion of software ecosystem partly overlaps with what is commonly known as a *software forge* (e.g., SourceForge, GitHub, and Google Code). The main difference is that the projects contained in software forges are often developed and evolved independently from one another, with little or no interaction between these projects and their developers. This does not exclude, however, that forges sometimes do contain subsets of projects that can be considered as ecosystems by themselves. For example Dabbish et al. observed, through a series of in-depth interviews with central and peripheral GitHub contributors, how much these contributors are aware of, and influence one another [28].

3. Research Goals

Research efforts in empirical studies of software ecosystems focus on a wide variety of different goals. The high-level goal is to gain a better understanding of how very large software systems (multiple millions of lines of code) maintained by large groups of developers (several hundreds or thousands of contributors) evolve over long periods of time (often over a decade). This insight can be used to come up with better tools and techniques to help developer communities in improving their daily evolution practices.

More specific goals could relate to the effort and cost required to evolve such systems, the reduction of defects, the quality, security, and complexity aspects of these systems, or process-related aspects.

3.1. Example Case Studies

This subsection explores the research field by presenting an inevitably incomplete list of empirical studies that have been carried out on specific software ecosystems. Each of these studies had a particular goal in mind, related to the ecosystem under study.

Eclipse is very popular open source integrated development environment (IDE), supported by IBM. The *Eclipse* framework features a plugin-based architecture, allowing third parties to extend its functionalities easily and independently.[3] Its evolution has been studied by various researchers [29,30,31]. In particular, Businge et al. analysed the evolution of the ecosystem of third-party *Eclipse* plugins, with the aim to understand which

[3]Direct competitors of *Eclipse* are the IDEs *Netbeans* and *IntelliJ IDEA*. Like Eclipse, *IntelliJ IDEA* has its own ecosystem of plugins. *Netbeans*, supported by Oracle, has a component-based architecture that can be extended with additional modules developed by third parties.

plugins are less problematic, survive longer, and why [32,33]. They found a clear difference in longevity between plugins that use stable and supported APIs as opposed to plugins that rely on discouraged and unsupported APIs.

CRAN is a package archive, containing several thousands of packages developed and used by the community surrounding the statistical software environment *R*. German et al. have studied its evolution over time [34]. Due to its increasing popularity, the maintainability of the *CRAN* ecosystem is quite challenging. With the goal to improve maintainability of *CRAN* packages, Claes et al. have studied which packages are more likely to cause, upon update, problems in dependent packages [35].

Debian is one of the most popular and most long-lived open source distributions of the Linux operating system. Canneil et al. studied its evolution over time [10]. Given the huge number of available *Debian* packages, a well-known problem is the fact that the installation of a package may cause unexpected and unintended conflicts with other, already installed, packages. Vouillon et al. studied this problem from an evolutionary point of view by analysing whether a given set of installed *Debian* packages can be upgraded safely without "breaking" the installation (due to problems of non co-installability) [36].

GNOME is a popular open source desktop environment for Linux, together with its direct competitor *KDE*. Because of its size, longevity and popularity, the evolution of the *GNOME* ecosystem has been actively studied by various researchers with various goals. Krinke et al. focused on the reuse and cloning of code between different *GNOME* projects [37]. Luijten et al. focused on the process and efficiency of handling issues in *GNOME*'s issue tracker [38]. Koch et al. studied effort, co-operation, and co-ordination in *GNOME* by analysing code line changes and developer communication via mailing lists [39]. They observed significant differences between developers in terms of contributed lines of code. Similarly, Gousios et al. developed an advanced measure of individual developer contribution based on information from the source code repository, the mailing lists and the bug tracking systems, and applied the measure to a number of *GNOME* projects [40]. Jergensen et al. studied six *GNOME* projects in order to understand how developers join, socialize, and develop within *GNOME* [41]. They observed that very experienced developers are *less* involved in the actual coding. Lopez et al. studied relations between the *GNOME* developers by means of social network analysis [42]. Ernst et al. studied perception of software quality requirements in some of the *GNOME* projects [43]. Finally, Vasilescu et al. explored the socio-technical nature of the *GNOME* ecosystem by taking into account the activity types of individual developers and projects [14].

3.2. Example Tools

Several tools have been proposed to analyse, understand, and visualise software ecosystems and their evolution. They can essentially be classified in two categories: those tools that are independent of a particular ecosystem, and those that are ecosystem specific. While the former are more generally applicable, the latter can leverage the specificities of the targeted ecosystem, in order to provide more dedicated support.

Ecosystem-independent tools. Many generic web-based tools are available that provide insight in the evolution of software products by analysing historical data extracted from version control repositories using a combination of metrics, visualisation, and statistics. A well-known example of these is *SonarQube*$^{\text{TM}}$ (www.sonarqube.org) that provides an extensible open source platform for managing code quality. The *GitHub* (`www.github.com`) web-based platform for hosting and managing projects using a git version control repository also includes a variety of views on the version control activity of ongoing projects, including network visualisations and historical visualisations. *Open Hub* (`www.openhub.net`), formerly known as Ohloh, provides similar functionalities for discovering, evaluating, tracking, and comparing open source projects. It is not a forge itself, since it does not host any projects. Instead, it can be considered as an open source directory offering analytical tools, search services, and tools for a wide range of project source code repositories. Neu et al. [44] developed *Complicity*, a web-based application aiming to support software ecosystem analysis through interactive visualisations. Ghezzi and Gall proposed Software Analysis as a Service (SOFAS), a distributed and collaborative software evolution analysis platform based on a service-oriented architecture [45].

Ecosystem-specific tools. Specifically for the *GNOME* ecosystem, Perez et al. [46] developed *SECONDA*, a dashboard for analysing various metrics about the *GNOME* ecosystem. This tool was, however, not integrated into any web application commonly used by the *GNOME* community.

Several *R* packages have been or are being developed to help maintainers of *CRAN* packages. One of these is called *miniCRAN*. It allows to create an internally consistent version of CRAN with selected packages only, and to analyse and visualise which packages are needed to satisfy all dependencies. Another one is called *maintaineR*, a web-based dashboard that allows CRAN package developers to understand and deal with the implications and problems raised by package updates. In addition to visualisation of package dependencies, the dashboard supports analysis of package conflicts and cross-package function clones [47].

In the context of the *Debian* distribution of Linux, the aforementioned problem of package co-installability has been supported by dedicated tools such as *coinst* (`coinst.irill.org`) [48]. The tool extracts and visualises co-installability kernels for GNU/Linux distributions, containing all the information necessary to identify components which are not co-installable.

In the same vein, many other ecosystem-specific tools exist and will continue to be developed.

4. Research Challenges

All of the aforementioned tools help to gain an understanding in how software ecosystems evolve over time. Empirical research in this domain, however, continues to face a number of important challenges, that are scientific, technical, pragmatic or ethical in nature.

4.1. Scientific Challenges

A first important scientific challenge is the *accessibility* of historical data of evolving software systems for analysis purposes. In order to be able to make scientific results *reproducible* by other researchers, free accessibility of data used and produced is a must.

A solution to this problem that is very frequently adopted is to resort to the study of *open source software* (OSS). Such software has become commonplace, and is being embraced even by many companies for commercial purposes. OSS can be found with many different lifetimes (sometimes even several decades), come in a huge range of software and community sizes, and are developed in a variety of different programming languages and for different application domains.

It remains an open question, though, to which extent results found for OSS can be *generalised* to commercial "closed" software systems, that often have a less geographically distributed developer community, a more strictly managed development process, and many other business constraints that are imposed by the commercial nature of the software or the company developing it.

Another limitation to reproducibility is that research results pertaining to the evolution of a particular software (eco)system rely on domain-specific information, such as the use of dedicated tools or guidelines by the development community. These are often only applicable for the system under study, and may influence the results obtained.

4.2. Technical Challenges

An important technical challenge relates to the extraction and combination of data coming from a wide variety of different data sources. To name but a few, historical analysis of software ecosystem relies on data stored in commits in version control repositories (such as *Subversion* and *Git*), communication through developer and user mailing lists, bug reports, and change requests stored in issue tracking systems (such as *Bugzilla* and *JIRA*), information from question and answer websites (e.g., *Stack Overflow*), and so on. Since these data sources store huge amounts of data, all typical problems when doing big data analytics [49] apply. More specifically, a list of typical challenges when mining Git repositories has been reported [50].

Fortunately, open source tools are available to help during the extraction and analysis process. As an example, I mention the *MetricsGrimoire* tool suite that I have used repeatedly in my research (`github.com/MetricsGrimoire`). Among others, it includes: *CVSAnalY*, to extract data from SVN or Git source code repository logs and store it into a relational database; *MailingListStats*, to extract mailing list information; *Bicho*, to extract historical data from issue tracking systems such as Bugzilla and JIRA. Another example are the *Evolizer* and *ChangeDistiller* tools presented in [51].

While these tools provide tremendous help with the extraction process, one will always remain confronted with the inevitable problem of inconsistent of incomplete data. The source of these problems may have been present in the original data that was extracted (for example due to encoding problems, or limitations and bugs in the version control systems and other tools used by the ecosystem community), or may have been introduced during the extraction process (by the tools used for extracting the data). Either way, data cleansing will be required in order to make the extracted data more consistent. As an example, Bruntink [52] explored this specific problem for the *Ohloh* data set, and found lots of problems related to data quality that required improvement.

If we want to focus on the socio-technical aspects of software ecosystems, a specific problem when combining data from different sources is *identity merging*. The same contributor to a software ecosystem may have contributed to different data sources by using a different name, e-mail address or login. In order to match all these different aliases to the same and unique identity, reliable identity merging algorithms are needed, combined with manual processing. Many such algorithms have been proposed (e.g., [53]), often by computing the similarity between pairs of aliases based on the Levenshtein distance. This metric computes the minimal distance between two given strings in terms of single character edits (deletion, addition or replacement) [54]. For example, the distance from Mike to Michael is 4 since a sequence of 4 edits is required:

$$\mathsf{Mike} \rightarrow_{replace(\ \mathsf{3,k,c})} \mathsf{Mice} \rightarrow_{add(\mathsf{3,h})} \mathsf{Miche} \rightarrow_{add(\mathsf{4,a})} \mathsf{Michae} \rightarrow_{add(\mathsf{6,l})} \mathsf{Michael}$$

A comparison between different identity merging algorithms revealed that the simpler ones outperformed (in terms of precision and recall) the more sophisticated ones [55]. Other algorithms, based on latent semantic analysis [56] appear to perform even better. Empirical evidence suggests that they are, on average, as good as the aforementioned algorithms, while providing much better scalability and accuracy for the worst cases.

4.3. Practical and Ethical Challenges

An important practical challenge relates to the need to feed back the results obtained during analysis of evolving software ecosystems to the development communities. As explained in Section 3.2 this requires the development of tools that are integrated into the daily practices of the ecosystem communities.

Another practical challenge is to share the data and results produced with other researchers. This is not always easy since a variety of different formats, tools, and operating systems is used, and since there may be storage problems due to the fact that we are dealing with huge amounts of data. The research community is becoming increasingly aware of this problem, and is putting into place solutions such as the “data track” of the “Mining Software Repositories” conference series that explicitly calls for researchers to share their datasets.

A related challenge is more ethical in nature. Given that we are using and *combining* data related to individual persons (e.g., developers of projects in the software ecosystem), what about their privacy? Making the results freely available to other researchers is necessary for reproducibility purposes, but would go against the need to respect privacy. Anonimising the data partially addresses this problem, but limits the reproducibility of results. Several approaches for guaranteeing anonymity have been proposed in literature, such as k-anonimity and l-diversity [57,58].

5. Ecosystem Diversity

In the context of an interdisciplinary research project, we have been exploring the analogies between natural ecosystems and biological evolution on the one hand, and software ecosystems and software evolution on the other hand. Extensive comparisons between these two research disciplines have been reported in [16,17].

In this lecture I only focus on *biodiversity*, a particular aspect of ecology. The motivation for doing so is similar to the one of the EC-funded FP7 FET project *DIVERSIFY* (2013-2016), which explores diversity as the foundation for a novel software design principle to increase the adaptive capacities in collaborative adaptive systems [59,60]. For a survey on the different aspects and applications of software diversity, see [61].

In order for an ecosystem to survive over longer periods of time, a number of characteristics are desirable [62,63,64,65]. We refer to its *stability* as its capacity to maintain an equilibrium over longer periods of time. Its *resistance* refers to its ability to withstand environmental changes without too much disturbances of the population of species. Its *resilience* refers to its ability to return to an equilibrium (not necessarily the same equilibrium as the original one) after a disturbance. A higher *biodiversity* of the ecosystem favours these characteristics.

Section 5.1 introduces a number of diversity metrics that have been used in ecological research. As an alternative, Section 5.2 presents econometric inequality indices, which also measure diversity in some sense.

5.1. Diversity metrics

In ecology, a series of metrics have been proposed to measure the degree of variation of species within a natural ecosystem. These are summarised in Table 1. These metrics can help to analyse the diversity of software ecosystems and how this diversity relates to their evolutionary characteristics.

Assuming a given ecosystem under study, the definitions of the metrics rely on the following notation. $S = \{s_1, \dots, s_n\}$ denotes the set of all $n = \mid S \mid$ species s_k belonging to the ecosystem. $I = \{i_1, \dots, i_N\}$ denotes the set of all $N = \mid I \mid$ individuals i_j belonging to the ecosystem. $I(s_k) \subseteq I$, abbreviated as I_k, denotes the population of a species s_k, i.e., the set of individuals in I belonging to species s_k. $n_k = \mid I_k \mid$ is the population size of species s_k, i.e., the number of individuals belonging to species s_k. We assume that all I_k form a partition of I, i.e., they are disjoint non-empty subsets such that $I = \cup_{k=1}^{n} I_k$. This implies that none of the species are extinct, i.e., there should be at least one individual in the population of each species. Finally, $p(s_k) = \frac{n_k}{N} \in [0,1]$, abbreviated as p_k, defines the probability of s_k, i.e., the proportion of all individuals in the ecosystem that belong to species s_k.

Table 1. Definitions of some diversity-related metrics

Diversity Index	Definition
species richness	$n = \mid S \mid$
species evenness (Piélou's index)	$E = \frac{H}{log(n)}$
Shannon diversity index (entropy)	$H = -\sum_{k=1}^{n} p_k log(p_k) = -\sum_{k=1}^{n} \frac{n_k}{N} log(\frac{n_k}{N})$
Simpson diversity index	$1 - \lambda = 1 - \sum_{k=1}^{n} \frac{n_k(n_k-1)}{N(N-1)}$
connectance (with cannibalism)	$C = \frac{L}{n^2}$
connectance (without cannibalism)	$C = \frac{L}{n(n-1)}$

The simplest diversity metric, known as *species richness*, measures the number of different species available in the ecosystem. It is defined as $n = \mid S \mid$. The intuition is that bigger is better: if more species are available there is a higher potential for diversity.

A more complex diversity metric is *species evenness*. It quantifies the relative abundance of the population of each species in the ecosystem. If all species are equally abundant (i.e., have the same number of individuals in their population), the evenness is maximal. If some species dominate the others, evenness will be low. One way that has been proposed to measure this evenness is based on Shannon's notion of *information entropy* [66]. Originally, information entropy was used to quantify the information content (or uncertainty) in character strings. The more different letters there are, and the more equal their proportional abundances in the string, the more difficult it is to correctly predict which letter will be the next one in the string. Similarly, when applied to species, entropy quantifies the uncertainty in predicting the species identity of an individual that is taken randomly from the dataset. If we divide Shannon's entropy H (also known in ecology as Shannon's diversity index) by the logarithm of the species richness n (see Table 1), we obtain the species evenness (also known as Piélou's index). For example, if we have $n = 5$ distinct species s_k, each with a population of $n_k = 10$ individuals, totalling $N = 50$ individuals in the ecosystem, we get $H = -\sum_{k=1}^{5} \frac{10}{50} log(\frac{10}{50}) = -log\frac{1}{5} = 1.609$ and maximal evenness $E = \frac{H}{log 5} = 1$.

A third diversity index, called the *Simpson index* (see Table 1), measures the degree of concentration when individuals are classified into species by computing the probability that two individuals taken at random from the dataset belong to *different* species. If all species are equally abundant, the index will be maximal and tends to 1 for large values of n.[4] For example, if we have $n = 5$ species of equal size $n_k = 10$ and hence $N = 50$, we get $1 - \lambda = 1 - \sum_{k=1}^{5} \frac{10\times 9}{50\times 49} = 1 - \frac{9}{49} = 0.816$. For $n = 50$ species of size 10 we get $1 - \lambda = 1 - \sum_{k=1}^{50} \frac{10\times 9}{500\times 499} = 1 - \frac{9}{499} = 0.982$.

According to Peet [67] Shannon's diversity is sensitive to variations in the size of the smallest populations of species, while Simpson's index is sensitive to variations in the size of the most abundant species. Using a combination of different diversity metrics is therefore probably a good idea.

Other diversity-related metrics rely on the representation of the ecosystem as a graph structure in which the nodes of the graph represent the species, and directed edges (called *links*) connect two nodes if there is a relation of consumer-producer (e.g., predator-prey) between both species. In a directed graph with $n = \mid S \mid$ nodes and L links, we define *connectance* $C \in [0,1]$ as the fraction of all possible links that are realised. This value depends on whether or not cannibalism is allowed (i.e., whether individuals of a species can eat individuals of the same species), as shown in Table 1. This metric has been shown to be related to the robustness of an ecosystem against loss of species. It turns out that certain topological properties of the graph structure influence the robustness of the network upon removal of highly connected species [68]. A higher connectance value is beneficial in that it delays the point at which the network displays high sensitivity to removal of most-connected species. Graphs with a low connectance value appear to be very sensitive to removal of most-connected species. Higher connectance values should also increase stability (the likelihood that population fluctuations in one species will affect other species), because such networks can reassemble themselves more readily, and thus may be less vulnerable to disturbance than simple communities.

In social network analysis research, a wide range of other related metrics have been defined (e.g., centrality measures such as betweenness and closeness). These fall outside the scope of the current lecture. For more information about social network measures and their application to open source software development, we refer the interested reader to [42,69,70,71].

[4]For large values of n, the Simpson index can be simplified into the so-called Blau index (also known as Gibs-Martin index or Gini-Simpson index) which is defined as $1 - \sum_{k=1}^{n} \frac{n_k^2}{N^2} = 1 - \sum_{k=1}^{n} p_k^2$.

5.2. Econometrics

Various *inequality indices* have been proposed in economy to study the inequality of statistical distributions (e.g., how is income distributed over the citizens of a country, or how is wealth distributed over different countries in the world). Many inequality indices have been defined in literature: Gini, Theil, Hoover, Kolm, Atkinson, ...

Assume that we have a set of data points $X = \{x_1, \ldots, x_n\}$ and $\overline{x} = \frac{\sum_{k=1}^{n} x_k}{n}$ denotes the mean of X. Table 2 provides the definitions of the inequality indices. These indices are very useful if one wishes to study highly skewed distributions, for which the mean and median values are known to be sensitive to variations.

Table 2. Definitions of inequality indices

Inequality Index	Definition
Gini	$\frac{1}{2n^2\overline{x}} \sum_{i=1}^{n} \sum_{j=1}^{n} \mid x_i - x_j \mid$
Theil	$\frac{1}{n} \sum_{k=1}^{n} (\frac{x_k}{\overline{x}} log \frac{x_k}{\overline{x}})$
Atkinson$_\alpha$	$1 - \frac{1}{\overline{x}} (\frac{1}{n} \sum_{k=1}^{n} x_k^{1-\alpha})^{\frac{1}{1-\alpha}}$
Hoover	$\frac{1}{2n\overline{x}} \sum_{k=1}^{n} \mid x_k - \overline{x} \mid$
Kolm$_\beta$	$\frac{1}{\beta} log (\frac{1}{n} \sum_{k=1}^{n} e^{\beta(\overline{x} - x_k)})$

Note the similarity between the definition of the Theil index and Shannon's notion of information entropy (presented in Table 1). In information theory, the Theil index corresponds to the notion of *information redundancy*. Because the Theil index produces a value ranging between 0 and $\ln n$, often the Theil index is *normalised* by dividing by $\ln n$ to obtain a value between 0 and 1.

Because of their robustness in presence of skewed distributions, inequality indices have been advocated for studying the evolution of software metrics distributions [72,73]. Vasa et al. used the Gini index to aggregate simple source code metrics, and studied how this index evolved over time for the considered metrics [74]. Serebrenik et al. used the Theil index for aggregating software metrics values [75]. Vasilescu et al. studied the correlation between these different econometric aggregation indices when applied to software systems [76]. Except for the Kolm index, all indices were shown to be strongly correlated. This means that, from a correlation point of view, Gini, Theil, Hoover, Atkinson can be used interchangeably. Which aggregation index to choose therefore depends on other factors, related to the definition of each index. The Kolm index, while not being correlated to the other econometric indices, appeared to corellate well to the mean value. Since mean and median values are known to be unreliable in presence of skewed distributions, the Kolm index could therefore present a better alternative than the mean.

Coming back to the diversity metrics presented in Section 5.1, econometric inequality indices may be a good substitute to assess the diversity of an ecosystem, since they can measure the inequality of the population size of the different species belonging to the ecosystem.

6. Duality and Bipartite Graphs

Particular ecological phenomena often exhibit a bidirectional relationship. Examples of this are the interaction between hosts and parasites, between plants and their pollinators (e.g., flowers and bees), between seeds and their dispersers (e.g., birds), or biogeographic networks (that link species to geographical regions where they occur). All of these relationships can be modelled as *bipartite graphs* or networks in which the nodes are divided into two disjoint sets, and edges connect a node of one set to a node of the other [77].

Ecological bipartite graphs are very useful to study the *robustness* to perturbations of the ecosystem [68]. Bipartite graphs can also be used to study *functional redundancy* in terms of the diversity of species. For example, if a flower species disappears, a particular species of bees that relied on it for pollen might be able to find pollen in another flower species. Such functional redundancy increases the chances that the extinction of a particular species will not lead to the extinction of others provided that there is sufficient diversity across layers.

In the remainder of this article we will explain how the same principles can be applied to software ecosystem as well.

6.1. Definitions

Definition 6.1. A *graph* is a tuple $G = (V,E)$ where V is a set of nodes (vertices) and $E \subseteq V \times V$ a set of edges linking nodes to each other. A *weighted graph* is a triple (V,E,w) such that (V,E) is a graph and $w : E \to \mathbb{N}$ is a function assigning a weight (strictly positive integer value) to each edge.

Definition 6.2. A *bipartite graph* is a quadruple $B = (V_1,V_2,E,w)$ with $w : V_1 \cup V_2 \to \mathbb{N}_0$ such that $(V_1 \cup V_2, E, w)$ is a weighted graph with $V_1 \cap V_2 = \emptyset$ and $E \subseteq V_1 \times V_2$.

This notion of bipartite graph can be useful to represent the socio-technical nature of a software ecosystem. We can define a *project-contributor graph* as a bipartite graph in which one layer contains the set of contributors C to the software ecosystem (e.g., the software developers), while the other layer contains the set of projects P of the ecosystem they contribute to.

Definition 6.3. Let P be a set of *projects* and C a set of project *contributors*.
A *project-contributor graph* (P,C,E,w) is a bipartite graph such that
$\forall p \in P, \exists c \in C : (p,c) \in E$ and $\forall c \in C, \exists p \in P : (p,c) \in E$.

The constraints on E in the previous definition are needed to consider only the *active* projects and contributors in the software ecosystem. Intuitively, $(p,c) \in E$ if contributor $c \in C$ has contributed at least once to project $p \in P$.

Given a project-contributor pair $(p,c) \in E$, the edge weight $w(p,c)$ represents the *workload* of contributor c for project p. This workload can be counted in terms of number of commits in the version repository, number of file touches, or any other measure of activity that is useful and relevant for the ecosystem and data source under study.

Definition 6.4. The *projection* $\pi_1(B)$ of a bipartite graph $B = (V_1,V_2,E,w)$ is the graph (V_1,E_1) such that $(v,w) \in E_1$ iff $\exists x \in V_2 : (v,x) \in E$ *and* $(w,x) \in E$. The *projection* $\pi_2(B)$ of a bipartite graph $B = (V_1,V_2,E,w)$ is the graph (V_2,E_2) such that $(v,w) \in E_2$ iff $\exists x \in V_1 : (x,v) \in E$ *and* $(x,w) \in E$.

Given a project-contributor graph we can compute its *project graph* or its *contributor graph* by using these projection functions π_1 and π_2. Intuitively, there is an edge between two projects in the project graph if at least one contributor is active in both projects. There is an edge between two contributors in the contributor graph if they were both collaborating on at least one project. The workload is ignored.

Definition 6.5. Let $B = (P, C, E, w)$ be a project-contributor graph. The *project graph* (P, E_P) of B is defined as $\pi_1(B)$. The *contributor graph* (C, E_C) of B is defined as $\pi_2(B)$.

6.2. Examples

The project-contributor graph can be seen from *dual viewpoints*. From the point of view of individual contributors, there should be a sufficient diversity of projects to work on, so that they can continue to contribute to the ecosystem if some project gets removed. From the point of view of individual projects, there should be a sufficient diversity of contributors, so that projects can continue to evolve if some contributors leave the ecosystem.

Figure 2 illustrates some artificial examples of very small project-contributor graphs (P, C, E, w) containing $\mid P \mid = 6$ projects and $\mid C \mid = 9$ contributors. For each example, the corresponding project graph $\pi_1(B)$ and contributor graph $\pi_2(B)$ is shown. These examples illustrate that the three types of graph are useful, as they reveal a different kind of information. The workload values $w_{ij} = w(p_i, c_j)$ for each edge $(p_i, c_j) \in E$ are not visualised.

In *Example A*, each project has either one or two contributors, and the projects are disconnected from each other since no contributor is working on more than one project. The graphs therefore do not represent a real ecosystem but rather a disconnected set of projects.

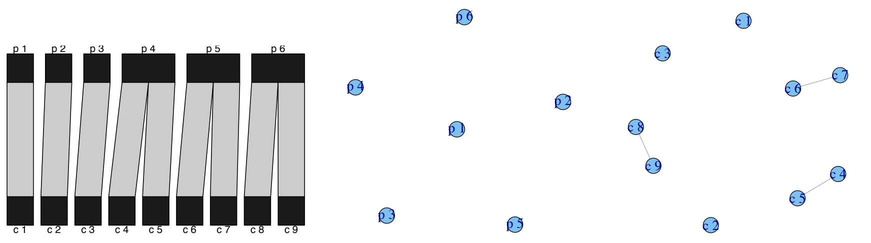
In *Example B*, each project has four contributors. Three contributors c1, c2 and c3 are involved in all projects, we therefore call them *core* contributors. The remaining contributors are involved in only one project each.

In *Example C*, each of the 9 contributors is involved in each of the 6 projects. If we assume the same workload w_{ij} for each edge (p_i, c_j) in the graph, then diversity will be maximal for each project (since all contributors participate in them) and for each contributor (since they participate in each project).

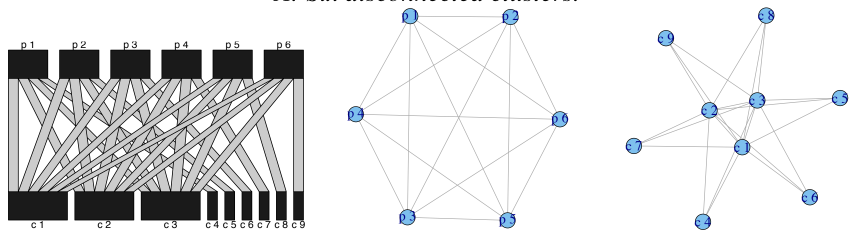
In *Example D*, all 9 contributors are involved in project p1, that we will call *core* project. Core contributor c9 is involved in all 6 projects. Although the project-contributor graph is quite different from the one in example C, its project graph and contributor graph are the same. What differs is the diversity, which is high for project p1 (since all contributors participate in it) but low for all other projects (since only one contributor participates in them). The same can be said for contributor c9 (with high diversity) as opposed to all other contributors (with low diversity). In contrast to Example C, the project-contributor graph for Example D is not very resistant to perturbations. If contributor c9 would disappear, all but one project will become inactive. If project p1 would be removed, all but one contributor would become inactive.

In *Example E* there is one core project p1 in which all contributors are involved. All other projects are maintained by a single contributor. The project graph therefore has a star shape, while the contributor graph forms a complete graph.

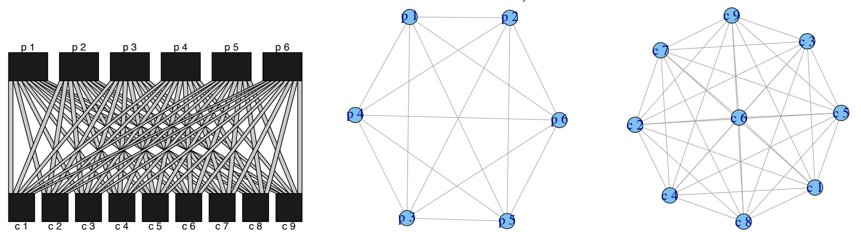
In *Example F* there are 2 core contributors c1 and c2 that are involved in the majority of projects, but not all of them. The number of contributors for each project ranges from 3 to 4.



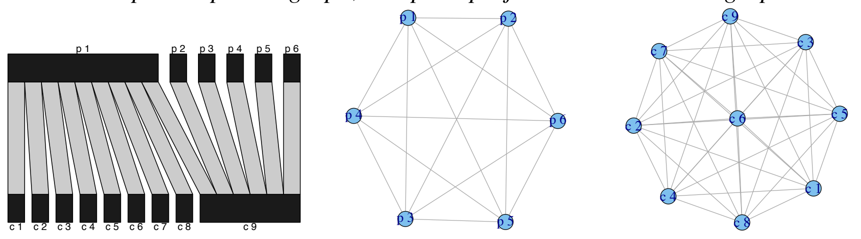
A: Six disconnected clusters.



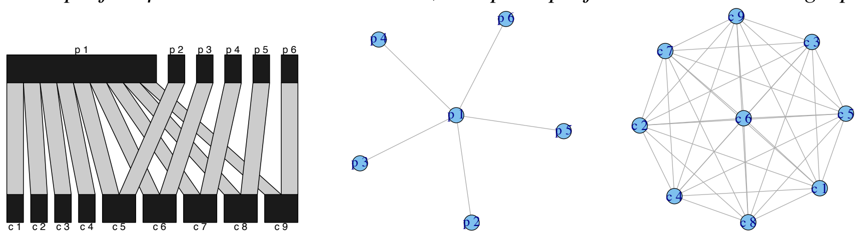
B: Three core *contributors* $c1$*,* $c2$ *and* $c3$*.*



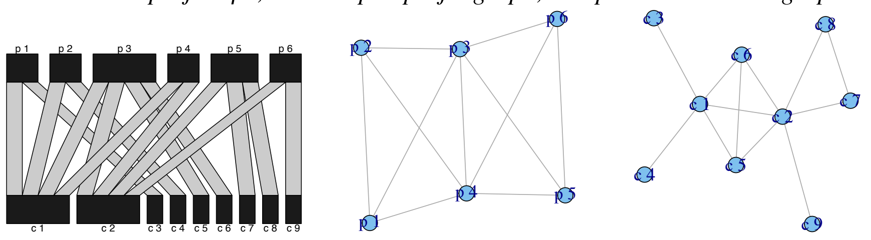
C: Complete bipartite graph; complete project and contributor graphs.



D: Core *project* $p1$ *and* core *contributor* $c9$*; complete project and contributor graphs.*



E: One core *project* $p1$*; star-shaped projet graph; complete contributor graph.*



F: Two core *contributors* $c1$ *and* $c2$*.*

Figure 2. Different configurations of a 6×9 *project-contributor graph* B (left), its corresponding *project graph* $\pi_1(B)$ (middle) and *contributor graph* $\pi_2(B)$ (right).

By construction, these graphs consist of 6 projects and 9 contributors, so the *project richness* is 6 and the *contributor richness* is 9. Table 3 gives the values of the *connectance* (without cannibalism) $C \in [0,1]$ for the project graph and contributor graph of each of the examples. The definition used here is slightly different from the one in Table 1 since we are considering undirected graphs as opposed to directed graphs. With the exception of Example A, the connectance is high for at least one of both graphs, and maximal for examples C and D.

Table 3. Connectance values $C = \frac{2L}{n(n-1)}$ for the examples of Figure 2.

Example	Connectance $C = \frac{L}{15}$ (project graph)	Connectance $C = \frac{L}{36}$ (contributor graph)
A	0	0.083
B	1	0.58
C	1	1
D	1	1
E	0.33	1
F	0.73	0.33

7. Diversity Metrics Revisited

When applied in the context of a bipartite graph, a *diversity index* (e.g., Shannon's or Simpson's diversity) expresses how specialised a given species is in relation to the species in the other level. In order to compute the diversity in a project-contributor graph (P, C, E, w) containing $n = \mid P \mid$ projects p_i and $m = \mid C \mid$ contributors c_j, we define the probability values $p(s_k)$ used in the definition of Shannon diversity in terms of workload. The total workload for each project p_i is defined as $P_i = \sum_{j=1}^{m} w_{ij}$, and the probability that can be attributed to each contributor c_j is defined as $\frac{w_{ij}}{P_i}$ (i.e., its proportion of workload). The total workload for contributor c_j is defined as $C_j = \sum_{i=1}^{n} w_{ij}$ and the probability (proportion of workload) attributed to project p_i is $\frac{w_{ij}}{C_j}$. The total ecosystem workload is defined as $W = \sum_{i=1}^{n} \sum_{j=1}^{m} w_{ij}$.

With these definitions we can compute *project diversity* and *contributor diversity* in terms of Shannon's entropy as defined in Table 1. A project's diversity will be low if it dominates most of the ecosystem's activity (i.e., it is a core project). A contributor's diversity will be low if she dominates most of the ecosystem's activity (i.e., she is a core contributor).

Definition 7.1 (Shannon diversity)**.**
The *diversity of a project p_i* is defined as $H_{p_i} = -\sum_{j=1}^{m} \frac{w_{ij}}{P_i} log \frac{w_{ij}}{P_i}$
The *diversity of a contributor c_j* is defined as $H_{c_j} = -\sum_{i=1}^{n} \frac{w_{ij}}{C_j} log \frac{w_{ij}}{C_j}$
(We assume that all edges (p_i, c_j) have a non-zero weight w_{ij}.)

In a similar way, the *Simpson index* can be computed for project-contributor graphs based on the definition of Table 1.

Definition 7.2 (Simpson diversity)**.**
The *Simpson index of a project* p_i is defined as $1-\lambda_{p_i} \;=\; 1-\sum_{j=1}^{m}\frac{w_{ij}(w_{ij}-1)}{P_i(P_i-1)}$
The *Simpson index of a contributor* c_j is defined as $1-\lambda_{c_j} \;=\; 1-\sum_{i=1}^{n}\frac{w_{ij}(w_{ij}-1)}{C_j(C_j-1)}$

Assuming a bipartite graph of species, we can use the notion of relative entropy (a.k.a. Kullback-Liebler divergence) to define the *specialisation* of a species in one level *relative to* the species in the other level [78]. Specialisation is the opposite of diversity: the more specialised a contributor's behaviour, the less diverse is her participation in the ecosystem's projects. Contributor specialisation is high if the contributor's activity monopolises a project and is not distracted by too many other projects. Project specialisation is high when a project monopolises a contributor's attention and receives little attention from other contributors.

Definition 7.3 (Specialisation)**.**
The *specialisation of project* p_i is defined as $S_{p_i} \;=\; \sum_{j=1}^{m}\frac{w_{ij}}{P_i}(log\frac{w_{ij}}{P_i}-log\frac{C_j}{W})$
The *specialisation of contributor* c_j is defined as $S_{c_j} \;=\; \sum_{i=1}^{n}\frac{w_{ij}}{C_j}(log\frac{w_{ij}}{C_j}-log\frac{P_i}{W})$

In order to obtain values in a [0,1] range, these values need to be normalized by the theoretical maximum and minimum possible values of the measures. See [78] for more details, or simply use the R package bipartite to compute the specialisation.

Table 4. Workload weights and computed diversity values (Shannon H, Simpson $1-\lambda$, specialisation S) for project-contributor graph of Example D.

w_{ij}	c_1	c_2	c_3	c_4	c_5, c_6, c_7 and c_8	c_9	P_i	H_{p_i}	S_{p_i}	$1-\lambda_{p_i}$
p_1	50	10	10	5	same as c_4	5	100	1.706	0.133	0.722
p_2	*0*	*0*	*0*	*0*	same as c_4	5	5	0	1.427	0
p_3	*0*	*0*	*0*	*0*	same as c_4	5	5	0	1.427	0
p_4	*0*	*0*	*0*	*0*	same as c_4	5	5	0	1.427	0
p_5	*0*	*0*	*0*	*0*	same as c_4	5	5	0	1.427	0
p_6	*0*	*0*	*0*	*0*	same as c_4	5	5	0	1.427	0
C_j	50	10	10	5	same as c_4	30	$W=125$			
H_{c_j}	0	0	0	0	same as c_4	1.792				
S_{c_j}	0.223	0.223	0.223	0.223	same as c_4	0.928				
$1-\lambda_{c_j}$	0	0	0	0	same as c_4	0.862				

Table 4 presents the diversity values (Shannon entropy H, Simpson diversity index $1-\lambda$ and specialisation S) for each project and each contributor of the project-contributor graph D of Figure 2. The values for Shannon's entropy H show that diversity is high for project p1 and contributor c9 and minimal (0) for all other projects and contributors. We can observe the same based on Simpson's diversity index. In addition, we find that diversity for c_9 is higher than for p_1. Intuitively, this can be explained by the fact that the workload is more evenly spread for c_9 (over the projects this contributor is involved in) than for p_1 (over the contributors involved in this project).

Posnett et al. [79] referred to these normalised specialisation indices as *developer attention focus* DAF (for contributor specialisation) and *module attention focus* MAF (for project specialisation). They applied them to seven projects maintained by the *Apache* Software Foundation. For these projects, they extracted data from commits in the git source code repository, and from bug reports in the Jira issue and bug tracking system. After merging both data sources, applying the metrics, and analysing them, they reported the following findings: (i) Top contributors tend to exhibit lower attention focus than others; (ii) When developer focus is higher, fewer defects are introduced at both the file and package level; (iii) Increased file activity focus results in a greater number of defects.

Together, these findings suggest that it is important for a developer to focus his efforts, but it is also important for files to receive some general attention. As first sight, finding (iii) may seem contradictory to finding (ii). Several factors could explain this result. Files with high MAF may be more complex and hence more defect prone. Another explanation could be that too focused file attention requires a greater diversity in order for defects are found.

8. Econometrics revisited

The previous section summarised a case study in which diversity and specialisation were defined in terms of entropy in order to study the module-contributor network of *Apache* projects [79]. This section summarises the findings of another case study, in which econometric inequality indices (in particular, the Gini index) have been used to study the specialisation of activity in the *GNOME* project-contributor network [14].

During 16 years of evolution of *GNOME*, more than 1400 projects (i.e., distinct Git repositories) have been maintained by over 5800 distinct contributors (after taking into account identity merging). Many of these contributors have participated to multiple *GNOME* projects, but there is a large disparity in the number of projects each contributor works on, as well as in the workload attached to these contributions. These differences in activity patterns are mainly visible from the contributor viewpoint. Our hypothesis is that the types of activity a contributor is preferentially involved in determines to a large extent her activity pattern.

To verify this hypothesis, we have classified all contributions into 14 different activity types. The most prevalent types are related to coding, localisation (e.g., translation), development documentation, and build files. The type of development activity a project contributor has been involved in can be estimated on the basis of the types of files she touched during each commit in the project's version control repository. The file type is determined based on regular expressions combining file name, file extension, and file path.

The notion of *workload* w_{ij} of a contributor c_j for a project p_i defined in Section 6 can be refined to take into account the activity type t_k as well.

Definition 8.1 (type workload)**.**
The *type workload* $w_{ijk} = w(p_i, c_j, t_k)$ counts the number of files of type t_k touched by contributor c_j for project p_i over its entire history. Since to each file is associated a given type $t_k \in \{t_1, \ldots, t_r\}$, the total workload $w_{ij} = \sum_{k=1}^{r} w_{ijk}$. The *relative type workload* $r_{ij}^k = \frac{w_{ijk}}{w_{ij}}$ defines the workload for a given activity type t_k relative to this total workload. Similarly, the *relative contributor-type workload* $r_j^k = \frac{\sum_{i=1}^{n} w_{ijk}}{\sum_{i=1}^{n} w_{ij}} = \frac{\sum_{i=1}^{n} w_{ijk}}{C_j}$ defines the workload of a contributor for a given activity type t_k relative to her total workload.

Before considering the type workload, let us first analyse the distribution of the workload $C_j = \sum_{i=1}^{n} w_{ij}$ for each *GNOME* contributor c_j. Figure 3 (left) shows a histogram reflecting this distribution. The x-axis is shown on logarithmic scale. We observe

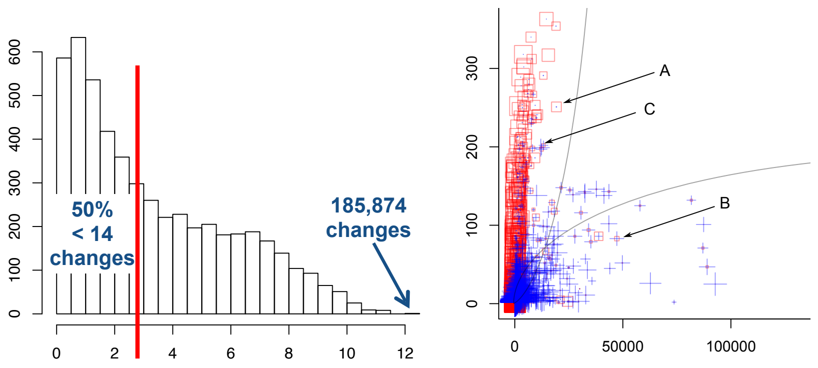


Figure 3. [Left] Histogram of contributor workload C_j (x-axis in log-scale) against number of contributors (y-axis). **[Right]** Scatter plot of contributor workload C_j (x-axis) against the number of projects she is involved in (y-axis). The size of the blue crosses represents the relative coding type workload, and the size of the red squares represents the relative localisation type workload.

a highly skewed distribution of contributor workload: the majority of contributors are involved in very few file touches. More precisely, if we create 2 bins, the first one corresponding to all contributors involved in less than 14 file touches (hereafter called *occasional* contributors), and the second one corresponding to contributors involved in at least 14 file touches (hereafter called *regular* contributors), both bins correspond to about 50% of the total population of contributors.

Let us now consider this contributor workload data from a different point of view, by relating it to the number of projects the contributor is involved in. The scatter plot of Figure 3 (right) clearly distinguishes two subcommunities. Above the exponential curve we find the *occasional* contributors that have a low contributor workload and tend to be involved in many projects. Below the logarithmic curve we find the *regular* contributors that have a higher workload and tend to be involved in less projects. The scatter plot shows that these two subcommunities are preferentially involved in different activity types. For each contributor, the relative type workload for the *coding* activity is shown as blue crosses, and for the *localisation* activity as red squares. The size of the symbol corresponds to the relative workload value. We observe that coders (blue crosses) correspond to regular contributors, while translators correspond to occasional contributors. Labeled arrows A, B and C in the scatter plot present some examples of atypical behaviour. For example, contributor C is involved mainly in coding but has a low workload while being involved in many projects. Contributor B is quite active in both coding and localisation.

The above supports our hypothesis that the activity type determines to a large extent a contributor's activity pattern (in terms of workload and number of projects involved in). But to what extent does a contributor specialise in terms of activity types? Is she evenly involved in many different activity types, or does she tend to specialise in one or very few activity types? To shed light on this question, we used one of the econometric indices introduced in Section 5.2, namely the Gini coefficient. For each contributor c_j, this coefficient aggregates the *relative contributor-type workload* values r_j^k over all activity types. Figure 4 (left) shows boxplots that summarise the Gini values obtained for each contributor. If we consider all contributors, we observe a highly unequal distri-

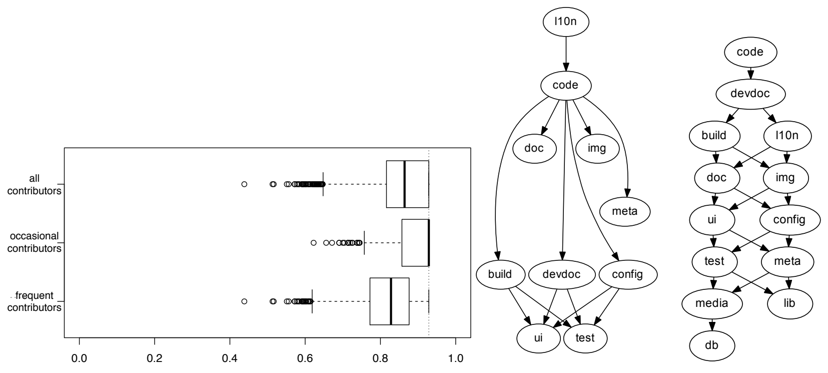


Figure 4. [Left] Inequality of the relative contributor-type workload according to the Gini coefficient. **[Middle]** Pairwise comparison of workload distribution per activity type for *occasional* contributors. **[Right]** Pairwise comparison of workload distribution per activity type for *regular* contributors.

bution. The large majority of contributors have a Gini value of > 0.8.[5] If we consider the subpopulation of *occasional* contributors, the inequality is even more pronounced. In fact, the median value coincides with the theoretical maximal value of the Gini index, implying that half of the occasional contributors are involved in a single activity type. If we consider the subpopulation of *regular* contributors, the inequality is less pronounced, but still very high, since the median value is still well above 0.8. Hence, the majority of regular contributors are involved most of the time in few activity types.

Knowing that occasional and regular contributors are highly specialised, what are the main activity types they tend to focus upon? To answer this question, we statistically performed pairwise comparisons of the distributions of contributor workload per activity type. The results are visually summarised in Figure 4 (middle and right). For occasional contributors (middle figure), we find that the localisation activity type (`l10n`) is the most important activity, and the second most important one is coding. For regular contributors, we find that coding (`code`) and development documentation (`devdoc`) are the most important activity types. This again provides additional evidence that *GNOME* has quite heterogeneous subcommunities of contributors.

9. Conclusion

The goal of empirical research in evolving software ecosystems is to increase understanding of how coherent collections or large software systems maintained by large developer communities evolve over long periods of time, and to come up with better tools and techniques to improve the maintenance of these ecosystems. To achieve this, many research challenge need to be overcome. In addition, in order to develop better models to understand, estimate, and predict the quality, cost, and effort of software maintenance, traditional software metrics need to be complemented by additional measures that capture other relevant properties of the software ecosystem that play an important role in the ecosystem dynamics.

[5]Note that the maximal theoretical value of Gini is $1 - \frac{1}{n} = 0.929$, since $n = 14$ in our case as there are 14 activity types. This is reflected by the dotted vertical line in the figure.

The social interaction (communication and collaboration) in the ecosystem's community of developers is likely to play an important rule in the evolution of the software ecosystem. To capture this aspect, several empirical studies have used the notion of project-contributor graph, focusing on the collaboration between projects and contributors of the ecosystem [79,14]. Such bipartite graphs can be used to reflect the duality between the projects of the software ecosystem and the contributors that are involved in these projects.

For instance, using *Apache* projects as a case study, Posnett et al. applied ecological diversity metrics to measure the attention focus of software contributors and software projects, and linked this to the introduction of software defects [79]. Using the *GNOME* ecosystem as a case study, Vasilescu et al. measured the workload of software contributors and software projects, and related this to specific types of activity [14]. Econometric inequality indices and comparisons of statistical distributions were used to find empirical evidence for the presence of heterogenous subcommunities with different activity patterns in terms of project involvement, workload, and preferential attachment to specific activity types. These and related empirical studies can form the basis of dedicated ecosystem-specific tool support for improving the maintenance activities of (sub)communities involved in the development and evolution of software ecosystems.

Acknowledgments

The research presented in this article was conducted in the context of ARC research project AUWB-12/17-UMONS-3 "Ecological Studies of Open Source Software Ecosystems" financed by the Ministère de la Communauté française - Direction générale de l'Enseignement non obligatoire et de la Recherche scientifique, Belgium. I thank my research collaborators Philippe Grosjean, Maëlick Claes, Alexander Serebrenik and Bogdan Vasilescu for their useful comments. I also express my gratitude to the organisers of the International Marktoberdorf 2014 Summer School on Dependable Software Systems Engineering for having given me the opportunity to present this research, and to the participants and lecturers of the summer school for the very relevant feedback I have received on my lecture.

References

[1] M. M. Lehman, "On understanding laws, evolution and conservation in the large program life cycle," *J. Systems and Software*, vol. 1, no. 3, pp. 213–221, 1980.

[2] M. M. Lehman and L. A. Belady, *Program Evolution: Processes of Software Change*, ser. Apic Studies In Data Processing. Academic Press, 1985.

[3] M. W. Godfrey and Q. Tu, "Evolution in open source software: A case study," in *Int'l Conf. Software Maintenance*. IEEE Computer Society, 2000, pp. 131–142.

[4] J. Fernández-Ramil, A. Lozano, M. Wermelinger, and A. Capiluppi, *Software Evolution*. Springer, 2008, ch. Empirical Studies of Open Source Evolution, pp. 263–288.

[5] N. H. Madhavji, J. F. Ramil, and D. E. Perry, *Software Evolution and Feedback: Theory and Practice*. John Wiley & Sons, 2006.

[6] T. Mens and S. Demeyer, *Software Evolution*. Springer, 2008.

[7] T. Mens, A. Serebrenik, and A. Cleve, *Evolving Software Systems*. Springer, 2014.

[8] H. Kagdi, M. L. Collard, and J. I. Maletic, “A survey and taxonomy of approaches for mining software repositories in the context of software evolution,” *J. Software Maintenance and Evolution: Research and Practice*, vol. 19, no. 2, pp. 77–131, 2007. [Online]. Available: http://dx.doi.org/10.1002/smr.344

[9] J. M. Gonzalez-Barahona, G. Robles, M. Michlmayr, J. J. Amor, and D. M. German, “Macro-level software evolution: a case study of a large software compilation,” *Empirical Software Engineering*, vol. 14, no. 3, pp. 262–285, Mar. 2009.

[10] M. Caneill and S. Zacchiroli, “Debsources: Live and historical views on macro-level software evolution,” in *Int’l Symp. Empirical Software Engineering and Measurement (ESEM)*, 2014.

[11] K. Manikas and K. M. Hansen, “Software ecosystems: A systematic literature review,” *J. Systems and Software*, 2012.

[12] N. Ducheneaut, “Socialization in an open source software community: A socio-technical analysis,” *Computer Supported Cooperative Work (CSCW)*, vol. 14, no. 4, pp. 323–368, Aug. 2005.

[13] 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 (ESEM)*. ACM , 2008, pp. 2–11.

[14] 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, no. 4, pp. 955–1008, 2014.

[15] T. DeMarco and T. Lister, *Peopleware: Productive Projects and Teams*, 2nd ed. Dorset House, February 1999.

[16] D. Dhungana, I. Groher, E. Schludermann, and S. Biffl, “Guiding principles of natural ecosystems and their applicability to software ecosystems,” in *Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry*, S. Jansen, M. Cusumano, and S. Brinkkemper, Eds. Edward Elgar, 2013.

[17] T. Mens, M. Claes, P. Grosjean, and A. Serebrenik, “Studying evolving software ecosystems based on ecological models,” in *Evolving Software Systems*, T. Mens, A. Serebrenik, and A. Cleve, Eds. Springer, 2014.

[18] A. J. Willis, “The ecosystem: an evolving concept viewed historically,” *Functional Ecology*, vol. 11, pp. 268–271, 1997.

[19] J. Bosch and P. Bosch-Sijtsema, “From integration to composition: on the impact of software product lines, global development and ecosystems,” in *Int’l Conf. Software Product Lines*. Springer, 2009.

[20] 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.

[21] D. Messerschmitt and C. Szyperski, *Software ecosystem: Understanding and indispensable technology and industry*. MIT Press, 2003.

[22] M. Lungu, “Towards reverse engineering software ecosystems,” in *Int’l Conf. Software Maintenance*, 2008, pp. 428–431.

[23] J. Bosch, *Design and Use of Software Architectures - Adopting and Evolving a Product Line Approach*. Addison-Wesley, 2000.

[24] M. Cordy, A. Classen, P.-Y. Schobbens, P. Heymans, and A. Legay, “Managing evolution in software product lines: a model-checking perspective,” in *VaMoS*. ACM, 2012, pp. 183–191.

[25] A. Pleuss, G. Botterweck, D. Dhungana, A. Polzer, and S. Kowalewski, “Model-driven support for product line evolution on feature level,” *J. Systems and Software*, vol. 85, no. 10, pp. 2261–2274, October 2012, special Issue on Automated Software Evolution.

[26] G. Botterweck and A. Pleuss, “Evolution of software product lines,” in *Evolving Software Systems*, T. Mens, A. Serebrenik, and A. Cleve, Eds. Springer, 2014.

[27] M. Jamshidi, Ed., *System of Systems Engineering: Innovations for the Twenty-First Century*. John Wiley & Sons, 2008.

[28] L. A. Dabbish, H. C. Stuart, J. Tsay, and J. D. Herbsleb, “Social coding in GitHub: transparency and collaboration in an open software repository,” in *Computer Supported Cooperative Work (CSCW)*. ACM, 2012, pp. 1277–1286.

[29] Z. Xing and E. Stroulia, “Refactoring practice: How it is and how it should be supported - an Eclipse case study,” in *Int’l Conf. Software Maintenance*. IEEE Computer Society, 2006, pp. 458–468.

[30] T. Mens, J. Fernández-Ramil, and S. Degrandsart, “The evolution of Eclipse,” in *Int’l Conf. Software Maintenance*, 2008, pp. 386–395.

[31] E. Shihab, Z. M. Jiang, W. M. Ibrahim, B. Adams, and A. E. Hassan, “Understanding the impact of

code and process metrics on post-release defects: a case study on the Eclipse project," in *Int'l Symp. Empirical Software Engineering and Measurement*, 2010.

[32] J. Businge, A. Serebrenik, and M. G. J. van den Brand, "Survival of Eclipse third-party plug-ins," in *Int'l Conf. Software Maintenance*, 2012, pp. 368–377.

[33] ——, "Analyzing the Eclipse API usage: Putting the developer in the loop," in *European Conf. Software Maintenance and Reengineering*. IEEE Computer Society, 2013, pp. 37–46.

[34] 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.

[35] M. Claes, T. Mens, and P. Grosjean, "On the maintainability of CRAN packages," in *Int'l Conf. on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, 2014, pp. 308–312.

[36] J. Vouillon and R. Di Cosmo, "Broken sets in software repository evolution," in *Int'l Conf. Software Engineering*, 2013, pp. 412–421.

[37] J. Krinke, N. Gold, Y. Jia, and D. Binkley, "Cloning and copying between GNOME projects," in *International Working Conference on Mining Software Repositories*. IEEE, 2010, pp. 98–101.

[38] B. Luijten, J. Visser, and A. Zaidman, "Assessment of issue handling efficiency," in *International Working Conference on Mining Software Repositories*. IEEE, 2010, pp. 94–97.

[39] S. Koch and G. Schneider, "Effort, co-operation and co-ordination in an open source software project: GNOME," *Information Systems Journal*, vol. 12, no. 1, pp. 27–42, 2002.

[40] G. Gousios, E. Kalliamvakou, and D. Spinellis, "Measuring developer contribution from software repository data," in *International Working Conference on Mining Software Repositories*. ACM, 2008, pp. 129–132.

[41] C. Jergensen, A. Sarma, and P. Wagstrom, "The onion patch: migration in open source ecosystems," in *SIGSOFT FSE*, 2011, pp. 70–80.

[42] L. Lopez-Fernandez, G. Robles, J. Gonzalez-Barahona, and I. Herraiz, "Applying social network analysis techniques to community-driven libre software projects," *Int'l J. Information Technology and Web Engineering*, vol. 1, no. 3, pp. 27–48, 2006.

[43] N. Ernst and J. Mylopoulos, "On the perception of software quality requirements during the project lifecycle," in *Requirements Engineering: Foundation for Software Quality*, ser. Lecture Notes in Computer Science. Springer, 2010, vol. 6182, pp. 143–157.

[44] S. Neu, M. Lanza, L. Hattori, and M. D'Ambros, "Telling stories about GNOME with Complicity," in *Working Conf. Software Visualisation (VISSOFT)*. IEEE, 2011, pp. 1–8.

[45] G. Ghezzi and H. C. Gall, "A framework for semi-automated software evolution analysis composition," *Automated Software Engineering*, vol. 20, no. 3, pp. 463–496, 2013.

[46] J. Perez, R. Deshayes, M. Goeminne, and T. Mens, "SECONDA: Software ecosystem analysis dashboard," in *European Conf. Software Maintenance and Reengineering*, T. Mens, A. Cleve, and R. Ferenc, Eds., 2012, pp. 527–530.

[47] M. Claes, T. Mens, and P. Grosjean, "maintaineR: A web-based dashboard for maintainers of CRAN packages," in *Int'l Conf. Software Maintenance and Evolution*, 2014.

[48] J. Vouillon and R. Di Cosmo, "On software component co-installability," in *Joint European Soft. Eng. Conf. and ACM SIGSOFT Int. Symp. on Foundations of Software Engineering*, 2011.

[49] K. Michael and K. Miller, "Big data: New opportunities and new challenges [guest editors' introduction]," *Computer*, vol. 46, no. 6, pp. 22–24, June 2013.

[50] C. Bird, P. C. Rigby, E. T. Barr, D. J. Hamilton, D. M. Germán, and P. T. Devanbu, "The promises and perils of mining Git," in *Int'l Conf. Mining Software Repositories*, 2009, pp. 1–10.

[51] H. Gall, B. Fluri, and M. Pinzger, "Change analysis with evolizer and changedistiller," *IEEE Software*, vol. 26, no. 1, pp. 26–33, 2009. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/MS.2009.6

[52] M. Bruntink, "An initial quality analysis of the ohloh software evolution data," *ECEASST*, vol. 65, 2014.

[53] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan, "Mining email social networks," in *Int'l Conf. Mining Software Repositories*. ACM Press, 2006, pp. 137–143.

[54] V. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet Physics Doklady*, vol. 10, 1966, pp. 707–710.

[55] M. Goeminne and T. Mens, "A comparison of identity merge algorithms for software repositories," *Science of Computer Programming*, 2011.

[56] 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.

[57] L. Sweeney, “k-anonymity: a model for protecting privacy,” *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, vol. 10, no. 5, pp. 557–570, 2002.

[58] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam, “l-diversity: Privacy beyond k-anonymity,” *ACM Trans. Knowl. Discov. Data*, Mar. 2007. [Online]. Available: http://doi.acm.org/10.1145/1217299.1217302

[59] D. Mendez, B. Baudry, and M. Monperrus, “Empirical evidence of large-scale diversity in API usage of object-oriented software,” in *Proc. of the Int. Working Conf. on Source Code Analysis and Manipulation (SCAM)*, 2013. [Online]. Available: http://www.monperrus.net/martin/diversity-api-usage.pdf

[60] B. Baudry, M. Monperrus, C. Mony, F. Chauvel, F. Fleurey, and S. Clarke, “Diversify: Ecology-inspired software evolution for diversity emergence,” in *Proc. of the Int. Conf. on Software Maintenance and Reengineering (CSMR)*, Belgium, 2014, pp. 444–447. [Online]. Available: http://hal.inria.fr/docs/00/91/62/81/PDF/csmr14-diversify.pdf

[61] B. Baudry and M. Monperrus, “The multiple facets of software diversity: A survey,” 2014.

[62] R. H. MacArthur, “Fluctuation of animal populations and a measure of community stability,” *Ecology*, vol. 36, pp. 533–536, 1955.

[63] R. M. May, *Stability and Complexity in Model Ecosystems*. Princeton University Press, 1973.

[64] C. S. Holling, “Resilience and stability of ecological systems,” *Annual Review of Ecology and Systematics*, vol. 4, pp. 1–23, 1973.

[65] K. S. McCann, “The diversity-stability debate,” *Nature*, vol. 405, pp. 228–233, 2000.

[66] C. E. Shannon, “A mathematical theory of communication,” *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 5, no. 1, pp. 3–55, 2001.

[67] R. K. Peet, “The measurement of species diversity,” *Annual Review of Ecology and Systematics*, vol. 5, pp. 285–307, 1974.

[68] J. A. Dunne, R. J. Williams, and N. D. Martinez, “Network structure and biodiversity loss in food webs: robustness increases with connectance,” *Ecology Letters*, vol. 5, no. 4, pp. 558–567, 2002. [Online]. Available: http://dx.doi.org/10.1046/j.1461-0248.2002.00354.x

[69] J. Martinez-Romo, G. Robles, J. M. González-Barahona, and M. Ortuño-Perez, “Using social network analysis techniques to study collaboration between a FLOSS community and a company,” in *Open-Source Software*. Springer, 2008, vol. 275, pp. 171–186.

[70] G. Concas, M. Lisci, S. Pinna, G. Porruvecchio, and S. Uras, “Open source communities as social networks: an analysis of some peculiar characteristics,” in *Australian Conference on Software Engineering*, 2008, pp. 387 – 391.

[71] M. Van Antwerp and G. R. Madey, “The importance of social network structure in the open source software developer community,” in *Hawaii Int'l Conf. System Sciences (HICSS)*. IEEE Computer Society, 2010, pp. 1–10.

[72] K. Mordal, N. Anquetil, J. Laval, A. Serebrenik, B. Vasilescu, and S. Ducasse, “Software quality metrics aggregation in industry,” *Journal of Software: Evolution and Process*, vol. 25, no. 10, pp. 1117–1135, 2013. [Online]. Available: http://dx.doi.org/10.1002/smr.1558

[73] O. Goloshchapova and M. Lumpe, “On the application of inequality indices in comparative software analysis,” in *Australian Conference on Software Engineering (ASWEC)*, 2013, pp. 117–126. [Online]. Available: http://dx.doi.org/10.1109/ASWEC.2013.23

[74] 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.

[75] A. Serebrenik and M. G. J. van den Brand, “Theil index for aggregation of software metrics values,” in *Int'l Conf. Software Maintenance*. IEEE Computer Society, 2010, pp. 1–9.

[76] B. Vasilescu, A. Serebrenik, and M. G. J. van den Brand, “You can't control the unfamiliar: A study on the relations between aggregation techniques for software metrics,” in *Int'l Conf. Software Maintenance*, A. Marcus, J. R. Cordy, and P. Tonella, Eds. IEEE, 2011, pp. 313–322.

[77] N. D. Martinez, R. Williams, and J. A. Dunne, *Diversity, complexity, and persistence in large model ecosystems*. Oxford University Press, 2006, pp. 163–185.

[78] N. Blüthgen and F. M. N. Blüthgen, “Measuring specialization in species interaction networks,” *BMC ecology*, vol. 6, no. 1, 2006.

[79] 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.