Evolution of the R software ecosystem: metrics, relationships, and their impact on qualities

Konstantinos Plakidas[1], Daniel Schall[2], Uwe Zdun[1]

Abstract

Software ecosystems are an important new concept for collaborative software development, and empirical studies on their development are important towards understanding the underlying dynamics and modelling their behaviour. We conducted an explorative analysis of the R ecosystem as an exemplar on high-level, ecosystem-wide assessment. Based principally on the documentation metadata of the R packages, we generated a variety of metrics that allow the quantification of the R ecosystem. We also categorized the ecosystem participants, both in the software marketplace and in the developer community, by characteristics that measure their activity and impact. By viewing our metrics across the ecosystem's lifecycle for the various participant categories, we discovered interrelationships between them and determined the contribution of each category to the ecosystem as a whole.

Keywords: R, software ecosystems, evolution, quantitative analysis, empirical study

1. Introduction

The move from monolithic, vertically integrated product development to more open, modular, and collaborative models in recent decades is a well-documented trend in software engineering and the business practice of software companies [44]. Software ecosystems represent the most recent step

Email addresses: konstantinos.plakidas@univie.ac.at (Konstantinos Plakidas), daniel.schall@siemens.com (Daniel Schall), uwe.zdun@univie.ac.at (Uwe Zdun)

[1]Software Architecture Research Group, University of Vienna, Währinger Straße 29, 1190 Vienna, Austria

[2]Siemens Corporate Technology, Siemensstraße 90, 1210 Vienna, Austria

in this process [19, 35], where a community of developers collaborates asynchronously, and often without central direction, over a common platform or software market [36, 44, 49]. As summed up by Hanssen [35], the ultimate objective for investing in and working towards an ecosystem is that all members will gain more benefits from being a part of it, as compared to a more traditional software product development approach with segregated roles, a low level of collaboration, and closed processes.

The background and motivation of the members, the variety of the resulting software components of the ecosystem, and the complexity of interactions between them, present a challenge in modelling a software ecosystem. The need for such modelling, of interest for both the software development and business management fields, has been recognized since the early days of the emergence of software ecosystems as a separate domain [45]. In this paper we build upon our previous work on the high-level quantitative assessment of the R ecosystem [55] as an exemplary study on the understanding and modelling of software ecosystems. We provide an in-depth examination of the evolution of the R ecosystem's dependency network and the contribution and collaboration patterns of the developer community. Finally we use the metrics we have extracted to examine qualitative aspects of the R ecosystem that pertain to its health. Thus, for instance, we examine the developer contribution and collaboration patterns to determine their influence on the R ecosystem's development and prospects. Finally, we discuss the applicability of our method and our findings to software ecosystems in general.

The remainder of this paper is structured as follows. Section 2 summarizes the current state of research on the field of categorizing and modelling software ecosystems. Next, Section 3 offers an introduction to the R ecosystem and presents the work that has been done on it by other studies. Then Section 4 presents our objectives in studying the R ecosystem. Our analysis and data-gathering approach is explained in Section 5. The study of the R ecosystem itself is divided in two main sections: Section 6 presents the composition, evolution, and dependency network characteristics of the R software marketplace, where the individual software package is the main unit of analysis, and Section 7 presents the characteristics of the R developer community as extracted from the package author information. Section 8 summarizes the metrics we have gathered and presented in the previous sections, and uses them to perform a qualitative analysis of the R ecosystem from the viewpoint of "ecosystem health". The discussion in Section 9 addresses the main insights of our research on the R ecosystem as well as on

software ecosystems in general, while examining the problems, limitations and potential improvements of our approach. Finally, Section 10 presents our intentions for possible future works.

2. Related Work

The concept of a software ecosystems (SECO) emerged in the 2000s, building upon the namesake business concept, and ultimately drawing upon the well-established concept in ecology [40, 50]. SECOs have been conceived as the natural next step in an ongoing process of moving from monolithic, company-internal development processes to open, collaborative ones [19]. Since its inception, the field has been the object of several studies examining ecosystems from the perspective of architecture, business and management issues, and the social relationships evidenced in their communities (for an overview, cf. [49, 17]).

Many of the research challenges in the SECO domain were defined by Jansen et al. [45], and several attempts have been made to construct a framework for a comprehensive SECO taxonomy. Thus, Bosch [19] proposed a basic categorization between ecosystems centred on operating systems, applications, or end-user programming, with a parallel differentiation based on the platform (desktop, web, mobile) the ecosystem is deployed on. Jansen et al. [45] take a software vendor view and consider the ecosystem as composed of three levels: the ecosystem itself, the software supply network, and the software vendors. In another work, Jansen et al. [43] proposed three different scopes for analyzing ecosystems as organizations: external, internal, and organization-centric. Campbell and Ahmed [22], in contrast, stress the business, architectural, and social dimensions of an ecosystem and the interactions between them. As business entities, SECOs are further commonly categorized according to their *accessibility*, depending on the existence of any barriers or vetting process before participating in the ecosystem; their *ownership* by a managing entity, which determines its business model, ranging between free and community-owned ecosystems to proprietary, company-owned ones; and the existence, number, and nature of the ecosystem's *software markets* [44, 49].

Indeed, business and management issues are particularly important for any company deciding to move towards the SECO model [19], as well as for the maintenance of an existing ecosystem. A business-aspect analysis of an ecosystem inevitably leads to attempts to measure the "health" of an

ecosystem, i.e., determining what its characteristics say about its current state and prospects, detect trouble-spots, and highlight the positive areas. This is linked to the biological (and business) concept of a system lifecycle involving birth, expansion, leadership, and either self-renewal or eventual death [51, 40]. This concept has been expanded and elaborated upon by subsequent studies such as den Hartigh et al. [30], Manikas and Hansen [48], and particularly Jansen [42], who links specific metrics to each of these health aspects on two levels: the "network level" encompassing the entire ecosystem, and the "project level" concerning the individual ecosystem constituents. Franco-Bedoya et al. [31] have also linked specific metrics to quality attributes of a software ecosystem, distinguishing the health of the community from the health of the ecosystem's software component network.

Analyses of specific open source ecosystems have already been undertaken before. For example, Kabbedijk and Jansen [46] analyzed Ruby as a network of software components and developers; Syeed et al. [59] compared the overlap of relationships between developers and the project dependency structure of the Ruby ecosystem; Spauwen and Jansen [56] studied the roles and motives of open-source developers active in browser extensions; Hoving et al. [39] categorized and quantified the software and community members of the Python ecosystem; Alami et al. [16] examine three open source e-commerce platforms and assess the health of the respective ecosystems.

To the best of our knowledge, however, no single study has ever attempted to combine the quantitative analysis of a diverse, major ecosystem with thousands of members in its actual state with an analysis of the same metrics across its existence, as opposed to small samples or individual software companies over smaller periods of time. Our approach therefore builds upon, tries to validate, and extends the state of the art in the field to produce a more comprehensive method for examining the evolution of software ecosystems.

3. R and its Ecosystem

As with other similar empirical studies, we chose a free, open-source (FOSS) ecosystem as it allows us to examine it in a depth unavailable for closed or commercial, proprietary ecosystems [49]. The R ecosystem was selected as it has grown from a niche tool for statisticians in academia to a popular solution in the broader data mining community [21]. Benefiting from the growing importance of the latter field, R has moved to become one of the most popular programming languages in the world: in July 2015, *IEEE*

Spectrum ranked it on place six, moving up from the ninth place in 2014, behind Java, C, C++, Python, and C# [13].

R is a programming language and free open-source (GNU-licensed) environment derived from the earlier S language developed at Bell Labs. The R core began its existence in 1997, and is maintained since by the R Foundation and the R Core Team [38]. Its main application fields are statistical computing and graphics. In addition to native R code, it makes extensive use of C, C++, and Fortran code, and it is available for all three major OS platforms: Windows, MacOS, and UNIX-derived platforms like Linux [38]. The main repository for software products developed by the R community, known as packages, is the *Comprehensive R Archive Network*, or CRAN [14]. In addition there is a number of related projects [11], chiefly the Bioconductor repository, which focuses on computational biology and bioinformatics packages [33], and RForge [8]. A large number of packages also exists independently on code repositories like GitHub.

R can be used either via the command line, or via GUIs and IDEs, of which RStudio [9] is one of the main representatives, developed under the supervision of Hadley Wickham, one of the main contributors to R.

A general outline of the R ecosystem is represented in simplified form in Figure 1. The ecosystem consists of three main areas, i.e., the R platform, the software marketplace(s), and the community, along with their interactions: dashed arrows represent the community impact on the software offerings, while solid arrows are a representation of dependencies between software products. The R platform consists of the R base packages contributed by the R core team, as well as a small set of "recommended" community-contributed packages, which are installed by default as part of the core, but which are also independently hosted in CRAN. Various third-party vendors like Microsoft offer broader collections as a basic "platform". In this study we have generally limited ourselves to the groups representing CRAN and Bioconductor, represented with striped fill in Figure 1.

The R ecosystem has already been the subject of some empirical studies. In its current state in March 2011 it has been analyzed by German, Adams and Hassan [34]. Based on a sample of 52 users and the packages they used, the authors examined the code characteristics and dependency relationships between the R platform and community-contributed packages, discovering that the latter were on average less well documented, had less versions and were more recent, less downloaded, and less re-used than platform packages. From an analysis of R mailing lists until the end of 2010, they also showed

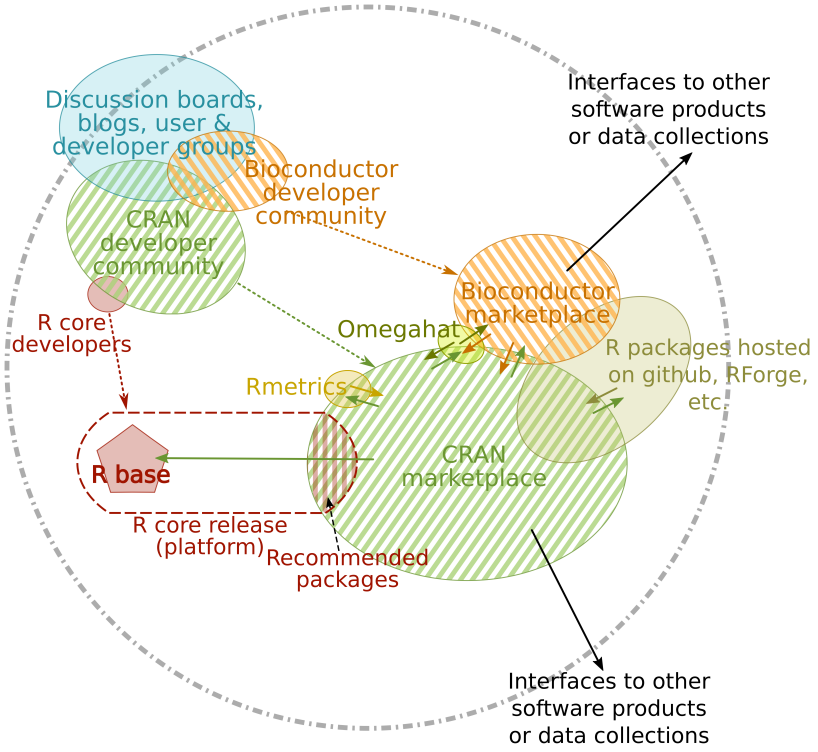

Figure 1: Overview of the R ecosystem. The striped groups represent CRAN and Bioconductor repositories, which were examined in the present study. The displayed sizes are not representative.

that the end-user community has grown super-linearly, whereas the developer community has remained stable.

Another team, composed of Claes, Mens, Claes, Grosjean, and Decan, from the University of Mons, have explored the CRAN repository from the perspective of package maintainability resulting from code errors and dependencies between packages. In [24], Claes et al. analyzed the occurrence of errors in CRAN packages over a period of three months, and found that errors resulted mostly from updates in the packages' dependencies, but that they were quickly corrected (usually within a week for package versions intended for Windows and Linux, but over twice that for MacOS). For this purpose, the team suggested a dashboard that visualizes package dependencies in [23] to aid in package maintenance. In another study [25], Claes et al. identified different types of code cloning behaviour within R packages.

The authors also pointed out the rise of GitHub both for active development as well as for hosting packages, but reaffirmed the importance of CRAN as the main repository for R packages, both numerically and, more importantly, dependency-wise. Decan et al. [29] also examined the dependency links between the main repositories of the R ecosystem, i.e., CRAN, Bioconductor, GitHub, and R-Forge, and found that CRAN is largely self-contained and forms the basis upon which the other repositories build. In another study, Decan et al. [28] returned to consider the maintainability problems from breaks in backwards compatibility between packages hosted in CRAN and GitHub, which is attracting an increasing share of R packages not hosted elsewhere.

Our own first study [55], examined the broad outlines of the R ecosystem's characteristics, particularly from the aspect of its division into two principal repositories, CRAN and Bioconductor. Unlike German et al. [34], we examined the entire content of the ecosystem, both past and present, but we did not consider GitHub or other minor repositories since, just as [29], we found that CRAN and Bioconductor were self-contained repositories. Our study contained a quantitative assessment of the R ecosystem's platform, marketplaces, and communities, established the main package and contributor metrics, and measured R's growth in these metrics throughout its existence. We discovered that a core group of users, linked to the development of the R platform, have played the most crucial role in extending the R ecosystem's functionality by contributing to the most important community-developed packages as well. We also made an attempt to combine the metrics we had gathered in a predictive model for package behaviour, but found that no meaningful results were possible with the data available.

This study builds upon and significantly extends this work to examine the evolution of the R ecosystem in greater depth, particularly from the aspect of interaction between the community and the marketplaces. We examine in much more detail the community and marketplace networks, identify the main categories of ecosystem components, the forces that have shaped it and what role they have played, thereby extending the quantitative analysis to examine qualitative aspects of the R ecosystem.

4. Objectives

Inspired by similar studies in the literature presented in the previous sections, we used the data we gathered on the R ecosystem's past and present

state to formulate metrics that can be used to gain insights on the ecosystem's evolution and detect any underlying trends, both on the level of the entire ecosystem, as well as on the level of individual ecosystem members. The metrics we used are those defined in proposed frameworks [42, 31], or commonly used in network analysis, and which can be generated easily, and as far as possible automatically, from openly accessible information, rather than requiring intensive human input for generation.

A platform-oriented SECO like R comprises three main components: the *software platform*, the *community of users*, and the *marketplace(s)* where additional software products are offered. In the present study, we focus on a high-level examination of the latter two components, whose existence and synergy present the main extension offered by SECOs as software development paradigms.

RQ1 *How is the R marketplace built up?* A software marketplace is the realization of the software ecosystem as a concept. By studying its dependency structure in terms of composition, width, depth, and interconnectedness, we extract metrics that can reveal the trends that have shaped its evolution.

RQ2 *How do community members contribute to the R ecosystem?* The developer community is the force that drives a software ecosystem's growth. We examine the patterns of behaviour of its members, both in terms of their contributions in terms of software packages, and in terms of their collaboration with each other. We also attempt a basic categorization of the community members, and study the impact of each category on the R ecosystem.

RQ3 *What is the current state of the R ecosystem?* Research Questions 1 to 3 are to a large extent explorative and were designed to help clarify the underlying dynamics and composition of the R ecosystem. That is, they aim at resolving the question what is "important" in the R ecosystem and how the R ecosystem "behaves". We combine the insights gained to review the usefulness of our metrics and assemble a qualitative assessment of the R ecosystem's health.

The R ecosystem is a special case in a wide spectrum of software ecosystems. At the end of our study we discuss of the results and insights gained are specific to the R ecosystem, which can be generalized to other software ecosystems, and why.

5. Data Collection and Analysis

At the start of our study, we defined our target data sources. From our own survey, we discovered that numerous packages hosted on RForge or GitHub are also listed in CRAN or Bioconductor. Consequently, as our focus was on examining the dynamics of marketplaces and their attendant communities in the R ecosystem, we limited our study to CRAN and Bioconductor. GitHub for instance forms an important software repository, but not a community; rather it is an agglomeration of individual developers and their contributions, and not one exclusive to R. Smaller collections such as Omegahat [6] or the Rmetrics Computational Finance group [12] were also disregarded as too small and niche-oriented, especially as many Omegahat packages are also hosted in CRAN, and Rmetrics has switched to using CRAN for its packages. In so far as these packages were present in CRAN, they were included in our analysis as part of the latter repository.

Our decision to limit our study to CRAN and Bioconductor was further strengthened due to the self-contained nature of both: as will be seen in Section 6.2.3, CRAN and Bioconductor both reference only a very small number (about 0.2% of documented dependencies) of software products external to themselves, i.e., either to packages hosted in smaller collections, personal repositories, code repositories like GitHub, or referred to non-R software altogether. Consequently, while many R packages can be also hosted in, and retrieved directly from, RForge, GitHub, or other such repositories, our data support the view that even if CRAN and Bioconductor do not contain the *entirety* of the R ecosystem, they do contain the vast *majority* of it and represent coherent and self-contained entities for analysis purposes. CRAN and Bioconductor membership overlaps to some degree, but this is mostly the case with packages that have migrated from CRAN to Bioconductor; where relevant in the subsequent sections, we note that overlap and how it was addressed.

As described in [55], we parsed the content of the R ecosystem at its state of 31 December 2015. This was performed using a Java program that parsed the online documentation pages and downloaded both the present and the archived versions for each package. In the latter, the documentation file was retrieved and parsed, and the code files were analysed using the *cloc* [4] tool. The information gathered was stored in JSON format due to the non-homogeneous nature of the data. Two data sets were extracted, concerning marketplace packages and the community. These data sets were

cleaned up in tandem across several rounds, with the aid of OpenRefine [7] and RStudio. OpenRefine's ability to cluster similar names based on different algorithms (Nearest Neighbor, Key-Collision, etc.) was particularly useful in cleaning up the many variants of developer names. In cases where there was ambiguity (e.g., is "Mike Smith" one of "Mike K. Smith" or "Mike L. Smith", or is one of the latter simply a typographic error?) we tried to match developers to the package in question by online search, but this was not always a successful endeavour. In such cases, we decided to retain all the variants as authors. We also improved or filled in missing or inaccurate release date documentation—particularly in Bioconductor, where the release cycles skewed the package version release dates by replacing them with the Bioconductor version release date—through the use of the far more reliable download data. About a third of all Bioconductor packages were affected in this way.

By combining our author data with package data, we calculated metrics on the static characteristics of the R ecosystem, as well as metrics on its evolution over time. Partly in order to counteract the skewing of our Bioconductor date data resulting from the effects of its semi-annual release cycle, and partly to make the resulting charts more easy to grasp, we generally examined both the packages and the community by yearly cohorts. For network analysis, we used the open source graph visualization and analysis platform *GEPHI* [5], as well as R. Both offered a number of statistical methods; in particular we used:

- *Kendall rank correlation* (or *tau test*) [57], included in the *base* package of R. The Kendall rank correlation is a non-parametric statistical test used to measure the correlation of two ordinal variables, or different subsets of the same variable.

- *Cliff's* δ [26], provided by the *orddom* package in R. Cliff's δ is a robust statistical method for calculating the difference between the probability that a random value in one distribution is larger than a random value in a second distribution, and the reverse[47]. For easier comprehension, we have presented it in the form of the *probability of superiority* $\hat{P} = \frac{(\delta+1)}{2}$ (also known as *area under the receiver curve*, AUC) [47].

- *Betweenness centrality*, provided by *GEPHI*, measures a node's centrality in a network by calculating the possible paths from all other nodes that pass through it.

- *Closeness centrality*, provided by *GEPHI*, measures a node's centrality in a network by calculating its proximity to all other nodes.

- *PageRank* [54] is an algorithm developed for use by the Google search engine and provided as the homonymous plugin for *GEPHI*. It is an implementation of *eigenvector centrality*, which measures a node's importance in a network by examining its connections to other nodes. It considers not only the number of incoming connections to a node, but also whether these are from important nodes.

6. Marketplace and Package Characteristics

6.1. Marketplace Size and Evolution

RQ1 concerns the quantitative and qualitative assessment of the R marketplace. This involves on the one hand its study in its present state, so as to establish the main categories of marketplace members, examine the roles they play, and measure the interactions between them. Subsequently, the development of the marketplace is traced back through time, so as to establish the driving forces and trends that have shaped it.

Table 1 gives an overview of the CRAN and Bioconductor marketplaces on 31 December 2015, the date of our data collection. "Current" packages were those included in the current release of each repository, i.e., they were up-to-date with the latest R release version (at the date of our data collection, this was Version 3.2.3). Packages which no longer satisfied these criteria were relegated to "archived" lists, which means that they did not appear in the main package list of the CRAN repository, and no longer had a package "home page" there. Older versions of "current" packages are likewise relegated to the "archived" lists.

45 packages were encountered in both CRAN and Bioconductor, either being released in parallel in both repositories—officially a discouraged practice [1]—or having migrated from CRAN to Bioconductor in later versions (some as late as 2015), usually due to their proximity to Bioinformatics. 41 of them were in the latter category, two—*hexbin* and *pamr*—appear to have migrated from Bioconductor to CRAN, while the situation of the remaining two packages—*bim* and *GeneTS*)—was unclear. In this study, the migrated packages have been treated as part of the repository they have been migrated to, while the two anomalous cases have been regarded as part of CRAN as it is broader in scope. In the subsequent analysis, where repository-specific

data is used, only the data (e.g., downloads) drawn from the repositories they have been assigned to have been considered.

Bioconductor further distinguishes its content into three broad categories, "function" packages which provide additional functionality, "annotation data" packages (e.g., genome data on various organisms), and "experiment data" packages. As the main expansions to the basic Bioconductor core, the "function" packages were of particular interest, and displayed markedly different characteristics from the other two categories. We have therefore often considered them as a special group in this study. Of the subset of packages included in the latest Bioconductor release (Version 3.2), 1,104 (49%) were "function" packages.

	CRAN	**Bioconductor**
Packages total	8,942	3,891
of which in "current" release	7,690 (86%)	2,255 (58%)
of which "function" packages	–	1,104 (49%)
Package versions total	54,490	27,024

Table 1: Overview of marketplaces as of 31 December 2015. The 45 packages published in both repositories are included in both.

It should further be noted that out of the Bioconductor versions, about 36% feature no change in their code content (or rather, no change in the number of code files, lines of code, or comment lines) between consecutive versions. In other words, a significant proportion of "new" Bioconductor version releases are identical to an older version, but as a result of the Bioconductor version numbering system [2], the version number is automatically increased with each Bioconductor release. The equivalent figure in CRAN remains under 1.5%. This gives an idea of the skewing introduced by Bioconductor's versioning system (and its version naming policy) in some categories of raw metrics, which consequently had to be handled with care.

As can be seen in Figure 2, CRAN and Bioconductor marketplaces have been growing at markedly different rates. Bioconductor was established in 2004 [33]. The first of what would later become Bioconductor packages were extant already in 2002, and by 2005, 162 packages were available, but a full third of the Bioconductor marketplace, 1,364 packages, was published in 2006, and the repository has been only slowly growing since. As pointed out in [55], the annotation data packages experience a high deprecation rate,

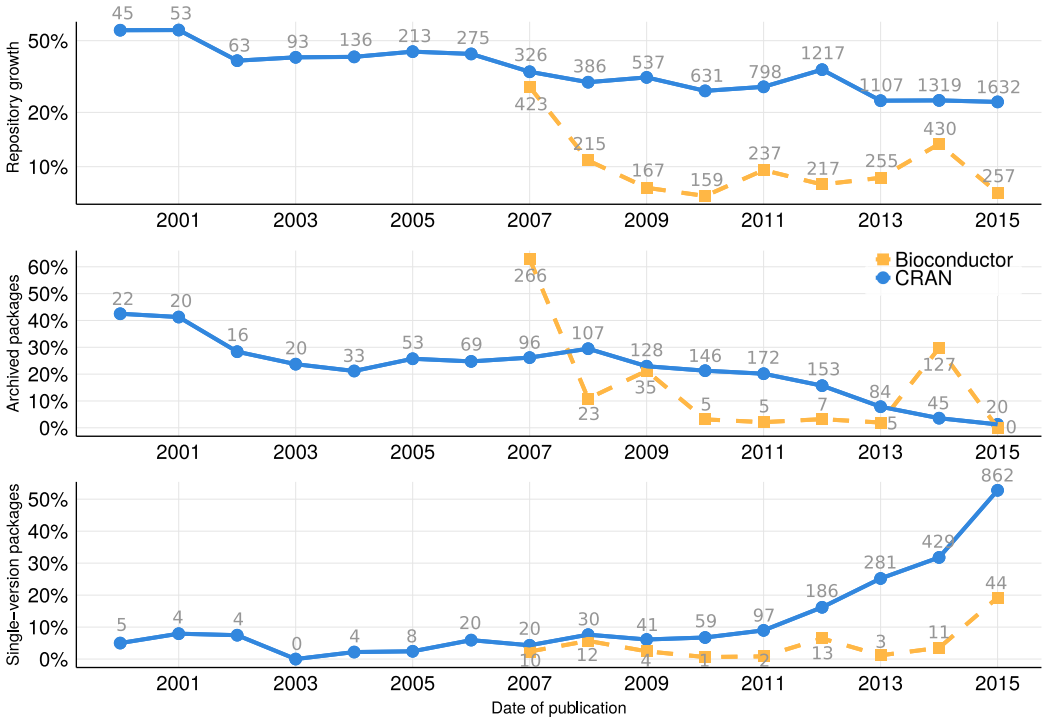

Figure 2: Growth rates (top), proportion of packages in a given year now marked as "archived" (middle), and proportion of single-version packages (bottom), in the CRAN (blue solid line) and Bioconductor (yellow dashed line) repositories. Initial years have been removed as outliers. Each node is labelled with the corresponding absolute value.

which is expressed in the over 80% of packages published in 2006 being currently "archived" in Figure 2. As will be seen in the subsequent sections, this affects many of the metrics we have used in this study. CRAN, as the older repository, exhibits a quite steady evolution, with an average annual growth of 28% since 2007, and a deprecation rate of between 20% and 30% for the bulk of the older packages (published before 2012). Newer packages of course are disproportionately represented in the *active* repository.

6.2. Dependency Structure

Dependency structure is of particular importance for any analysis of a composite software system, since the interdependency of the various elements increases complexity and development overhead, and reduces the efficiency and the composability of the resulting software [20], while at the same time

increasing modularity and reusability of the individual components. In addition, dependency relationships can reveal much information about the reuse of functionality, the propagation of changes such as updates, errors, or bug fixes ("ripple effects") through the ecosystem, and the underlying structure of the ecosystem in the form of functional clusters.

In this section we employ the data we gathered from CRAN and Bioconductor to examine **RQ1**. We examine its structure and define groups of packages according to their position and role in the dependency network, with emphasis on examining the co-dependence between CRAN and Bioconductor.

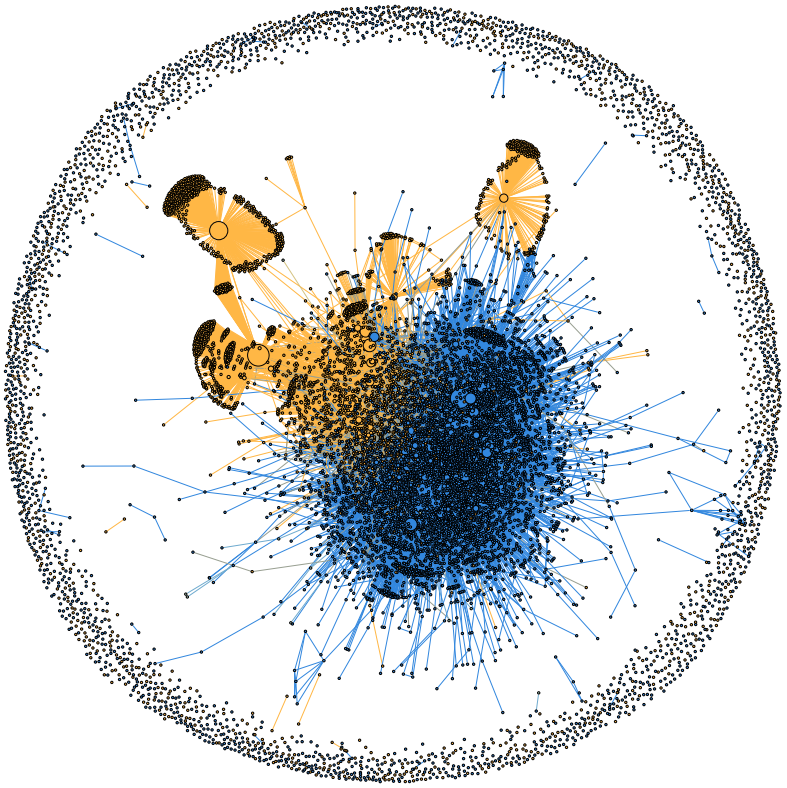

Figure 3: Overview of the dependency relationships (edges) of the packages (nodes) in the CRAN (blue) and Bioconductor (yellow) repositories. The "halo" surrounding the central component and smaller groupings are packages without explicit dependencies to or from other packages outside the R platform.

The dependencies manifest the reuse of offerings in the ecosystem; to borrow from biology, they represent the links in a food chain, connecting

the lower and more elementary offerings to the highly specialized top-level consumers. As such, a package's position in that chain, and its connections to other packages, determines the role it plays in the ecosystem. A package situated low on the food chain, whose offerings are reused further along the line by many other packages, plays a fundamental role in the ecosystem, whereas a package that is located "higher", i.e., at the middle or higher rungs of the food chain, and has perhaps one or two packages that "feed off" of it, is a branch or leaf in the wider ecosystem. Apart from its role and importance to the ecosystem's functionality, a package's position in a dependency network is also an indicator of its own susceptibility to change: the more dependencies, direct or indirect, a package has, the more likely it is to need active maintenance due to interface changes lower in the food chain—indeed, as we demonstrate further on, the most reused (depended-upon) packages are largely found in the lower rungs of the chain.

6.2.1. Dependency Metrics

The data collected for our study already provided information on the *direct dependencies* (*Deps*) of each package, which was included in the relevant documentation in the fields "depends" and "imports". We did not consider "soft" dependencies like "suggests", which include packages that might be required for additional features, but are not strictly necessary for a package to function. About 790 packages also explicitly documented the *direct reverse dependencies* (*RevDeps*) upon each package, but this information was otherwise missing and had to be reconstructed through the direct dependencies. Many packages contained a direct reference to either the R core as a whole or to individual base or "recommended" packages included within the core. For the purposes of reconstructing the marketplace as a dependency network radiating outwards from the R core, we considered the core (but not the "recommended" packages) as a unitary platform.

In order to perform a dependency-based categorization of each package in the ecosystem network, we built upon the direct and reverse dependency data to define a few additional metrics. We were inspired by similar approaches in software analysis[3], but largely the metrics arose from simple consideration of the dependency network (example in Figure 4).

- The *maximal depth of the dependency chain* (*MaxDepthDep*) for each

[3]See e.g. http://www.jarchitect.com/Metrics

package, measured from the R core platform via intervening packages, where "0" represents packages without any dependency outside the R platform. This serves as a measure of the package's maximum distance from the platform.

- The *maximal depth of the reverse dependency chain* (*MaxDepthRevDep*) for each package, measured from the package itself to the last "leaves" of the network, via the intervening packages, with the given package at place 0. This is an indicator of how deep the dependency tree that a package supports is.

- The *number of transitive dependencies* (*TrDeps*) of each package, i.e., the number of all packages that find themselves in the dependency path(s) of a package's dependencies. In similar manner, we also calculated the *number of transitive reverse dependencies* (*TrRevDeps*), i.e., the transitive closure of *TrDeps*.

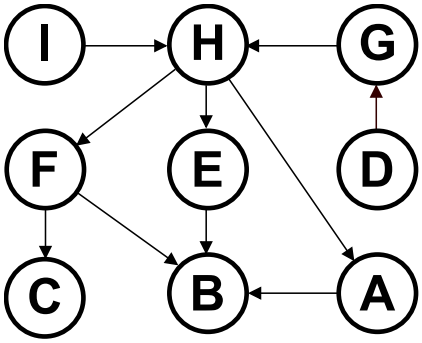

Figure 4: Example of the dependency metrics: package H has 3 direct and 5 transitive dependencies (outgoing arrows) and a max. dependency depth of 2, as well as 2 direct and three transitive reverse dependencies (incoming arrows) and a max. rev. dependency depth of 2.

The metrics we defined are useful for exploring the dependency network of a package, but are not well suited for examining the exact nature of the dependency relations. Thus it is not possible to state with certainty, without analyzing the source code so as to examine specific function calls, whether all nodes and paths of the dependency chains forming on either side of the package actually represent a package's actual technical dependencies. For instance, in Figure 4, we cannot know if package H actually reuses functionality

from C; only that it transitively requires C to be installed. Equally it is impossible to examine the scale and nature of these dependencies. This means that, without a more detailed examination of the source code, the presence of transitive dependencies alone is not sufficient for examining something like the ripple effect of an update in package C, or quantifying the relative importance of a package's dependencies.

6.2.2. Dependency Network Survey

In [55] we reported on the basic quantitative parameters for the CRAN and Bioconductor repositories. In that study, only direct dependencies through the "depends" field were considered, and resulted in rather low numbers of packages with dependencies on other R packages: some 54% for CRAN, and 66% for Bioconductor. By adding the "imports" field into the consideration, the figure rises to ca. 70.3% of packages with dependencies for both CRAN and Bioconductor, which are figures comparable to similar open-source ecosystems like Android [18].

The revised figures concerning the basic dependency data are summarized in Table 2. In addition, Bioconductor function packages (denoted as *Bioc-f*) have been examined separately, since they display markedly different behaviour than the annotation and experiment data packages. The characteristics have been calculated on the basis of the combined dataset of both repositories, thereby including dependencies from Bioconductor to CRAN which would otherwise not be accounted for.

Out of the 12,788 packages examined in total, only 9,829 (7,005 in CRAN, 2,824 in Bioconductor) are involved in a dependency relationship. Out of these, 95 packages form smaller dependency clusters (mostly package–and–dependency pairs) independent from the wider marketplace, while the rest form a central dependency network of 9,751 interconnected packages (6,923 CRAN, 2,812 Bioconductor) and 31,908 dependency links. The remaining 1,896 packages in CRAN and 1,063 packages in Bioconductor have no dependencies other than to the R core.

At first glance, Table 2 shows a similar behaviour for CRAN and the Bioconductor marketplace. The two exceptions are Bioconductor's markedly higher *MaxDepthDep* values, and the higher values of *RevDeps* among CRAN packages. The reason for this is that Bioconductor builds upon CRAN offerings, as will be seen later when the dependency links between the two repositories are examined. Table 2 also shows that the bulk of the packages have a small number of dependencies, and that dependency chains are mostly

	CRAN	**Bioconductor**	**Bioc-f**
Packages total	8,901	3,887	1,170
Deps >0	6,252 (70.2%)	2,752 (70.8%)	1,088 (92.7%)
Deps >1	4,451 (50%)	1,600 (41.2%)	959 (81.7%)
Deps >5	1,092 (12.3%)	671 (17.3%)	479 (40.8%)
Deps quartiles	0–2–3	0–1–3	2–4–8
MaxDepthDep quartiles	0–2–4	0–6–6	3–6–9
RevDeps >0	2,285 (25.7%)	566 (14.6%)	391 (33.4%)
RevDeps >1	1,304 (14.6%)	268 (6.9%)	219 (18.7%)
RevDeps >5	552 (6.2%)	110 (2.8%)	96 (8.2%)
RevDeps >10	337 (3.8%)	73 (1.9%)	65 (5.5%)
RevDeps quartiles[a]	1–2–5	1–1–4	1–2–5
MaxDepthRevDep quartiles[a]	1–1–3	1–1–3	1–1–3

[a] for packages with at least one reverse dependency

Table 2: Overview of basic package dependency characteristics by repository

shallow—this is particularly true in Bioconductor, where most packages have *MaxDepthDep* of either 0, indicating that they have no dependencies, or 5–6, indicating that they directly reuse a Bioconductor package that has a high *MaxDepthDep*, such as *AnnotationDbi* (*MaxDepthDep*=4, *RevDeps*=697) or *matchprobes* (*MaxDepthDep*=5, *RevDeps*=553). In CRAN, in contrast, *MaxDepthDep* is much more evenly spread out due to the greater variety of packages that comprise it. Thus while the CRAN network had an average clustering coefficient[61] of 0.079, indicating a mostly amorphous and sparsely connected network, Bioconductor had an average clustering coefficient of 0.179, precisely due to the clustering of a large number of packages around packages like *AnnotationDbi* or *matchprobes*. Otherwise the two repositories display similar characteristics. *Bioc-f* packages show consistently higher dependency density, both in *Deps* as well as in *MaxDepthDep*. This is as we expect, since functional packages naturally make much greater re-use of the ecosystem offerings than annotation or data packages. It is possible that if a similar filtering of CRAN packages were undertaken to remove the

obscuring influence of various dataset and similar packages, the results would be similar.

6.2.3. Dependency Flows

Next we examined the flows of dependencies between various package groups. We decided to split up the central dependency network into smaller functional communities, and performed a modularity-based community detection [52] to that effect. At each iteration, however, we obtained different clusterings around the major packages, with greatly varying compositions. Consequently we were not able to define specific communities in the dependency network. However, we observed that the algorithm was consistently able to discern between CRAN and Bioconductor packages with great accuracy, and detect the main Bioconductor clusters. For instance, even with settings that resulted in a very small number (3 to 5) of communities, the large *AnnotationDbi–matchprobes* and *MeSHDbi* clusters of Bioconductor, which are so prominent in Figure 3, were always present as individual communities.

As a result we moved to examining the dependency links within and between the two repositories. The results, presented in Table 3, verify the largely self-contained nature of CRAN, which remains largely independent of either Bioconductor or of R packages published independently or in smaller repositories. The reverse is true for Bioconductor, which demonstrates a continuing, significant technical dependency on CRAN, with about a third of all dependencies in Bioconductor being provided by CRAN, a figure that rises to over 40% for function packages. This also accounts for the differences between CRAN and Bioconductor in Table 2: CRAN packages provide functionality for Bioconductor, resulting in a greater dependency depth of Bioconductor packages, since it includes the dependency depths of many CRAN packages as well.

This relationship between CRAN and Bioconductor is illustrated by the simple graph representation, using *GEPHI*, of the two repositories in Figure 3. This representation is typical of the visualizations we obtained for the R ecosystem, regardless of the tool or the clustering algorithm employed. We can see that Bioconductor packages form distinct clusters that build upon, and stand out from, the mass of CRAN packages. These clusters are mostly composed of annotation and data packages, however, where the re-use factor is very low.

Furthermore, both repositories refer to a very small number of "external"

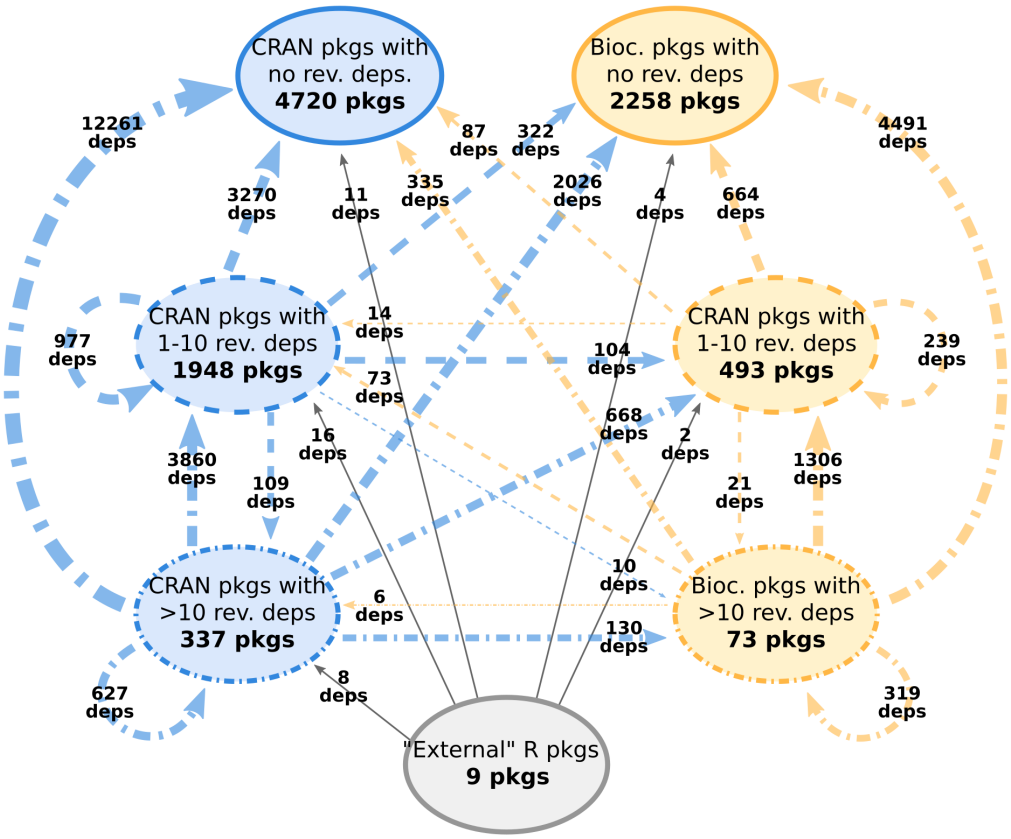

Figure 5: Dependency flows between the two repositories (CRAN at left and Bioconductor at right) and package groups (no rev. dependencies–low rev. dependencies–high rev. dependencies).

sources as dependencies. Of the 17 "external" dependencies, three (*rcom*, *RGtk*, *fEcofin*) are packages formerly hosted in CRAN, but now removed entirely rather than archived; one (*Rlibstree*) is a former Bioconductor package; eight refer to different software products altogether (e.g., *ODBC* or *mSQL*) that are not strictly dependencies; and five to truly external R packages like *Rniftilib*, hosted by R-Forge.

A striking feature in Table 2 is the narrow base of dependency packages for both repositories: only a small fraction of packages is re-used more than five times. To better examine this phenomenon, we separated packages into three groups, based on Table 2: packages with high reuse (*RevDeps* >10),

	CRAN	**Bioconductor**	**Bioc-f**
Deps referenced in group	21,663	10,308	6,905
Corresp. indiv. packages	2,346	1,076	1,002
Deps to same group	21,104 (97.4%)	7,040 (68.3%)	3,731 (54%)
Corresp. indiv. packages	2,201 (93.8%)	542 (50.4%)	346 (34.5%)
Cross-repository *Deps*	515 (2.4%)	3,260 (31.6%)	2,854 (41.3%)
Corresp. indiv. packages	130 (5.9%)	530 (49.1%)	528 (52.6%)
Packages used only in other repository	84	24	22
Non-CRAN or Bioconductor *Deps*	44	8	9
Corresp. indiv. packages	15	4	5

Table 3: Overview of basic dependency network characteristics by repository

packages with low reuse (1–10 *RevDeps*), and packages with no reuse (0 *RevDeps*). The resulting dependency provider-consumer flow is illustrated in Figure 5.

It emerges that a relatively small group of packages plays a fundamental role in the R ecosystem's dependency network. This group is in reality even smaller than can be seen in Figure 5: even if we take the subset of packages with *RevDeps* >100, leaving us with only 67 packages, or 0.5% of all R packages examined, they still provide approximately 51% of all dependencies in the CRAN and Bioconductor repositories combined. This confirms the observation in [55] that the technical dependency of the Bioconductor repository from CRAN is mainly based upon a few "big" packages (*RSQLite*, *ggplot2*, *MASS*, *RColorBrewer*, *XML*, *Rcpp*, etc.), which unsurprisingly are also among the most common dependencies referenced within CRAN, while the rest of the CRAN packages are used relatively sparsely.

6.2.4. Assessing Package Importance

While *RevDeps* is an adequate measure of a package's degree centrality, or its immediate impact, in the ecosystem, it does not account for the transitive influence it may have. Thus a package that is reused only a few times, but by packages with a high number of reverse dependencies, may have a greater impact on the ecosystem than is immediately apparent. Here the other metrics defined in Section 6.2.1 come into play: the number of transitive reverse dependencies *TrRevDeps* and the maximal depth of the reverse dependency chain *MaxDepthRevDep*. As a normalized measure of a package's *transitive impact* (*TrImpact*), we divided *TrRevDeps* by *MaxDepthRevDep* for each package. The results are summarized in Table 4.

	CRAN	**Bioconductor**
Packages with *RevDeps* >0	2,285	566
TrImpact=1	902 (39.5%)	280 (49.5%)
1<*TrImpact*≤10	1,045 (45.8%)	214 (37.8%)
TrImpact>10	337 (11.8%)	72 (12.7%)
TrImpact quartiles	1–2–5	1–1.33–3.67

Table 4: Overview of package *TrImpact* by repository

An illustration of the correlation of *TrImpact* with its constituent variables is shown in Figure 6. In both repositories, *TrRevDeps* plays the more significant role, especially for the lower values, i.e., the overwhelming majority of packages involved, where it shows an almost linear relationship with the *TrImpact*. This is confirmed through performing a Kendall rank correlation. Across our dataset, the Kendall τ coefficient for *TrRevDeps* was calculated as 0.885, 0.832 for *RevDeps*, and 0.645 for *MaxDepthRevDep*. For packages with a higher impact value (arbitrarily set at greater than 10), *TrRevDeps* has a τ coefficient of 0.822, *MaxDepthRevDep* 0.529, and *RevDeps* 0.347.

Examining these "high transitive impact" packages, we discovered that they were mostly on the lower levels of the dependency chain: 40.3% had *MaxDepthDep*=0, and a further 17.6% *MaxDepthDep*=1. Furthermore, they generally had a large number of direct reverse dependencies, with 75.2% having *RevDeps*>10. Table 5 compares the metrics of this group compared with the similar-sized group of packages with *RevDeps* >10. The strong similarities are not coincidental, as 310 packages are common to both groups.

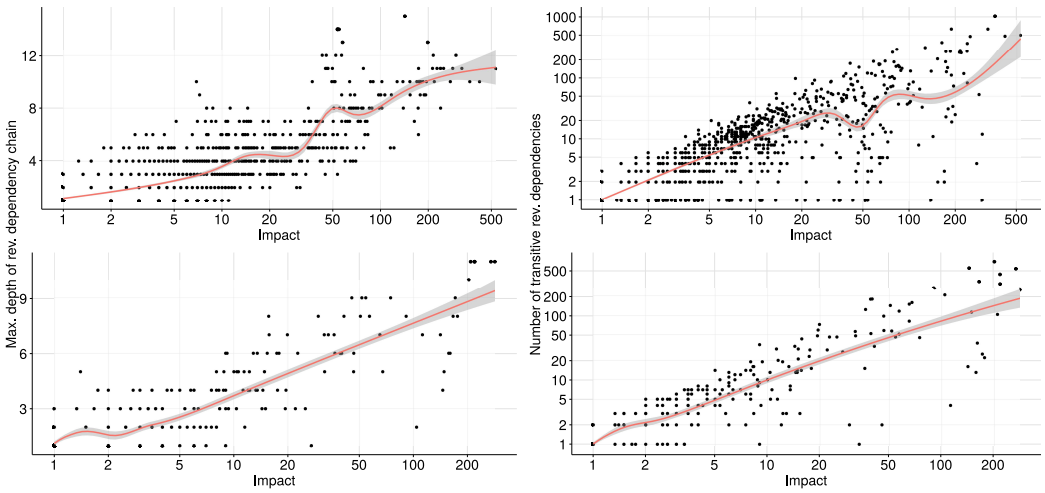

Figure 6: Correlation of *TrImpact* with *MaxDepthRevDep* and *TrRevDeps*for CRAN (top) and Bioconductor (bottom).

This shows that on the one hand, package impact can be adequately assessed by direct reuse of a package, and on the other, that the high-impact packages are found in the lower levels of the dependency chain and are generally older.

The differences between the two groups are few, but precisely therefore of considerable interest. This is found in the code content: compared with an average R code content of 435 lines of code (loc) in the two repositories, both subsets are considerably larger, but the "high reuse" subset has a clear advantage over the "high transitive impact" one. Since R code is generally analogous to feature richness, it is an indication that "richer" packages are more likely to be reused as dependencies.

Another interesting discovery was that among R packages used as dependencies, the *RevDeps* and *TrRevDeps* metrics are strongly correlated, with a τ coefficient of 0.797, indicating a strong, positive, almost linear relationship between them, as shown in Figure 7. Along with the similar values in *RevDeps* shown by the two subsets in Table 5, this corroborates the empirically observed fact that packages tend to refer directly to their functional dependencies even if they are part of a dependency cluster or chain, rather than implicitly consume what they require through intermediaries. In general, the picture that emerges is that the average R package has few dependency trees building transitively upon a dependency, and the trees that do form are neither deep nor wide. The exception here is again a small group

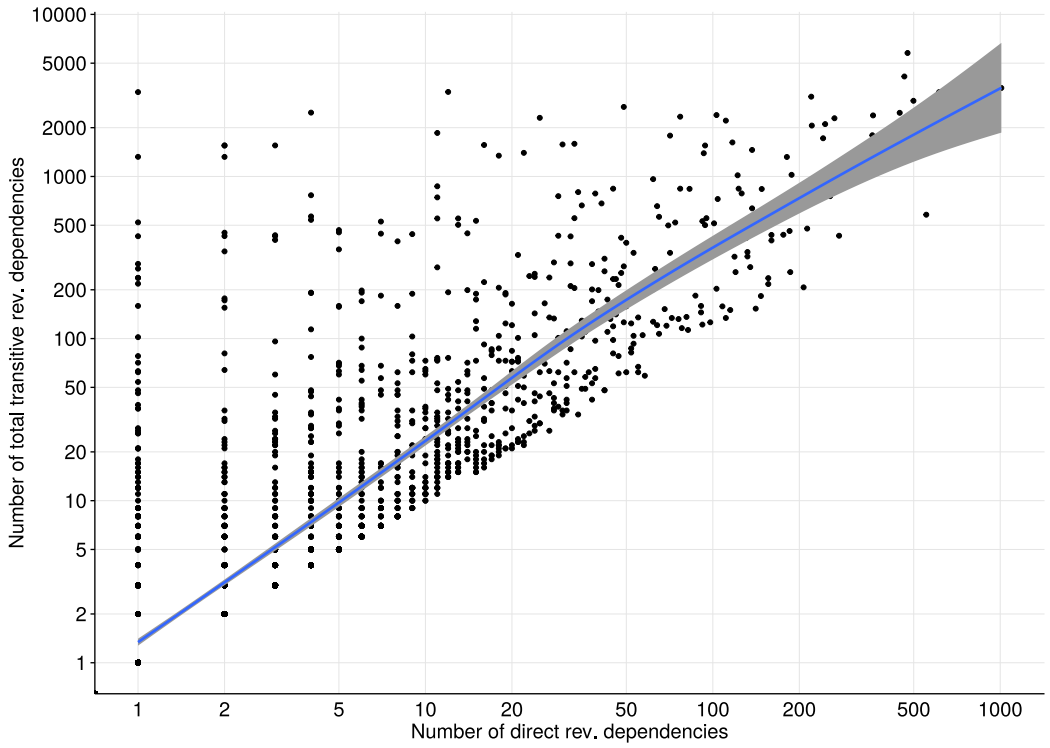

Figure 7: Plot of the number of transitive rev. dependencies to the number of direct rev. dependencies.

of "high-impact" packages, which largely coincide with the "fundamental" packages previously identified based on *RevDeps* alone. Even here, however, the chains of reuse, although often deep (5 levels and above), do not "branch out".

Nevertheless, while *RevDeps* and *TrRevDeps* are highly correlated—largely because they build upon the same subset of packages—they are not identical. Thus bugs and breaks in compatibility in the lower rungs of the dependency chain likely affect packages situated higher along the chain. This is a major maintenance problem for the developers of R packages, and one which is bound to grow the more the R ecosystem grows: changes in the "fundamental" packages have an increasingly wide-ranging impact, while the newer and more "distant" packages, which are also more likely to have a larger number of dependencies (as will be seen in Section 6.3), are more susceptible to these changes.

	High transit. impact	**High reuse**
Number of packages	409	410
CRAN/Bioconductor	337/72	337/73
Deps quartiles	0–1–3	0–1–4
RevDeps quartiles	10–25–56	15–25–55.75
TrRevDeps quartiles	63–160–511	32.25–89–350.8
MaxDepthDep quartiles	0–1–3	0–1–3
MaxDepthRevDep quartiles	4–5–8	3–5–7
Release date quartiles	2004-02 – 2007-09 – 2010-10	2004-03 – 2006-11 – 2010-03
Max. R code content (*loc*)	536–1,539–4,727	693.8–2,020–6,070

Table 5: Comparison of the two "fundamental" package groups: "high transitive impact" vs. "high reuse" packages

To complement the basic dependency-based metrics, we also performed a ranking analysis of the central dependency network with the PageRank algorithm. About 71% of all packages in the network were disregarded because they had no *RevDeps*. Only 691 had a value $\geq 1 \times 10^{-4}$, and 93 $\geq 1 \times 10^{-3}$. In Table 6 the 10 packages with a value $\geq 1 \times 10^{-2}$ are listed, along with the corresponding metrics examined so far.

6.3. Evolution of the Dependency Structure

In the previous sections, we have examined the state of the dependency network as it was at the time of our sampling. In this section, we move towards examining the dynamics that have shaped it over time.

6.3.1. Dependency Metrics by Package Age

The evolution of the main dependency metrics over time is illustrated in Figure 8. For ease of presentation, we have divided the packages into cohorts based on their age, i.e., their date of first release: 1997–2000, 2001–2005, 2006–2010, 2011–2015. Although the dates were somewhat arbitrarily chosen, they nevertheless can be said to represent specific eras in the R ecosystem's evolution: the first two cohorts represent its infancy and early years, whereas the latter two represent its maturity.

Package	Repository	PageRanks	RevDeps	TrImpact
BiocGenerics	Bioconductor	6.22×10^{-2}	220	284.27
lattice	CRAN	3.021×10^{-2}	476	529
AnnotationDbi	Bioconductor	2.233×10^{-2}	669	203.3
MASS	CRAN	1.976×10^{-2}	1,009	363
matchprobes	Bioconductor	1.754×10^{-2}	553	145.5
Biobase	Bioconductor	1.646×10^{-2}	499	270.27
Rcpp	CRAN	1.637×10^{-2}	613	323.63
Matrix	CRAN	1.286×10^{-2}	464	418.2
DBI	CRAN	1.114×10^{-2}	103	236.9
S4Vectors	Bioconductor	1.047×10^{-2}	311	219.27

Table 6: Top packages by PageRanks, along with two of the main metrics presented earlier: *RevDeps* and *TrImpact*

Examining the behaviour of the two repositories separately, they show a marked difference in the evolution of *Deps* per package: in CRAN, Figure 8 shows a clear increase over time whereas in Bioconductor, the packages of the period 2006–2010 show a decrease. This can probably be attributed to the nature of the packages published at the time: as already mentioned, a large number were annotation and experiment data packages, which largely depended on a single central package (chiefly in the *AnnotationDbi–matchprobes* and *MeSHDbi* clusters), which became obsolete quickly, and which were mostly not re-used in other packages. Nevertheless, the data overall appears to show growth in both the average *Deps* and the average *MaxDepthDep* in packages that are more recently released, as well as a corresponding increase for the *RevDeps* and MaxDepthRevDep metrics from the older packages over time.

All four metrics are indicative of an increase in the complexity of the ecosystem's dependency structure around the core provided by the older packages. This is in accord with Table 5, where both the "high reuse" and "high impact" packages were mostly found on the lower rungs of the dependency chains, and were, to over 75%, published up until 2010. The same holds true even in the smaller core of 67 packages with over 100 reverse dependencies.

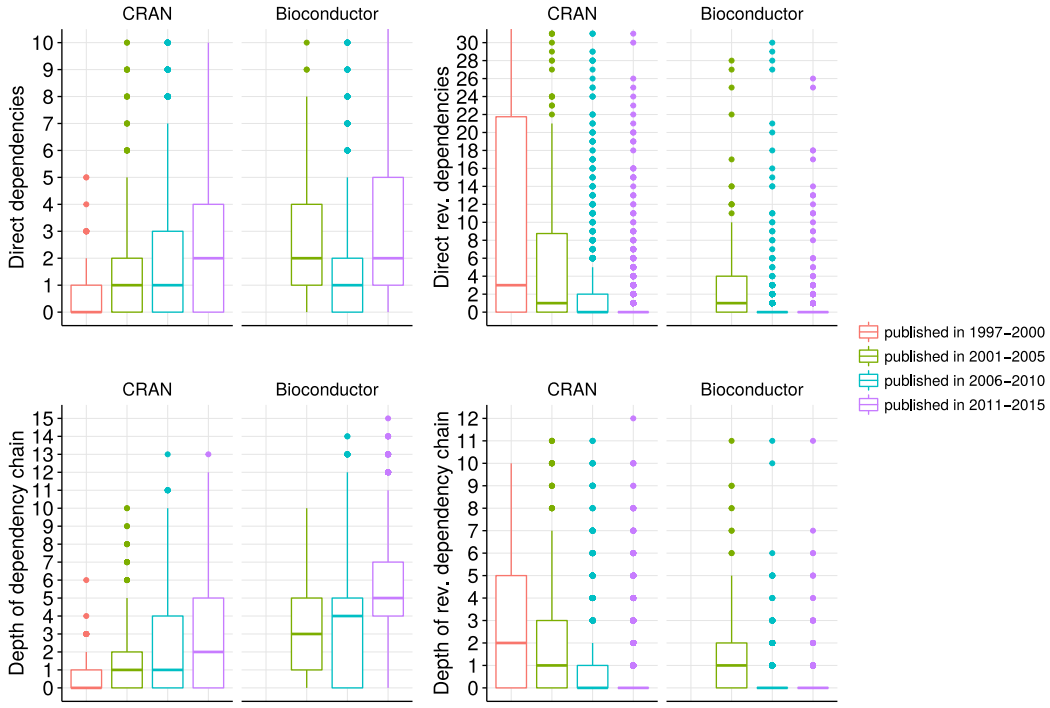

Figure 8: Comparison of the number of direct dependencies and reverse dependencies, and the max. depth of the dependency and reverse dependency chains per package in Bioconductor (left) and CRAN (right) by package age

We also observe that for *MaxDepthDep*, Bioconductor displays a more marked increase than CRAN. This is not unexpected: CRAN has a more varied nature, which tends to obscure such tendencies, while Bioconductor relies on a smaller number of fundamental packages. Given that the depth of the CRAN packages upon which Bioconductor builds is low—about 63% are at levels 0 or 1, i.e., adjacent to the core, and adjusted to take into account the frequency of package reuse, the overall average is at 1.3—this indicates that whatever growth there is in dependency depth derives largely from within the Bioconductor repository itself.

To verify this, we performed a comparative statistical analysis of the dependency metrics across the individual yearly cohorts. As the data groups had neither the same distribution nor a similar group size, we used *Cliff's* δ, from which we calculated the *probability of superiority* $\hat{P} = \frac{(\delta+1)}{2}$, also known as *area under the receiver curve* (AUC). The results of the analysis are shown in Figure 9 in the upper triangles, with the corresponding p-values

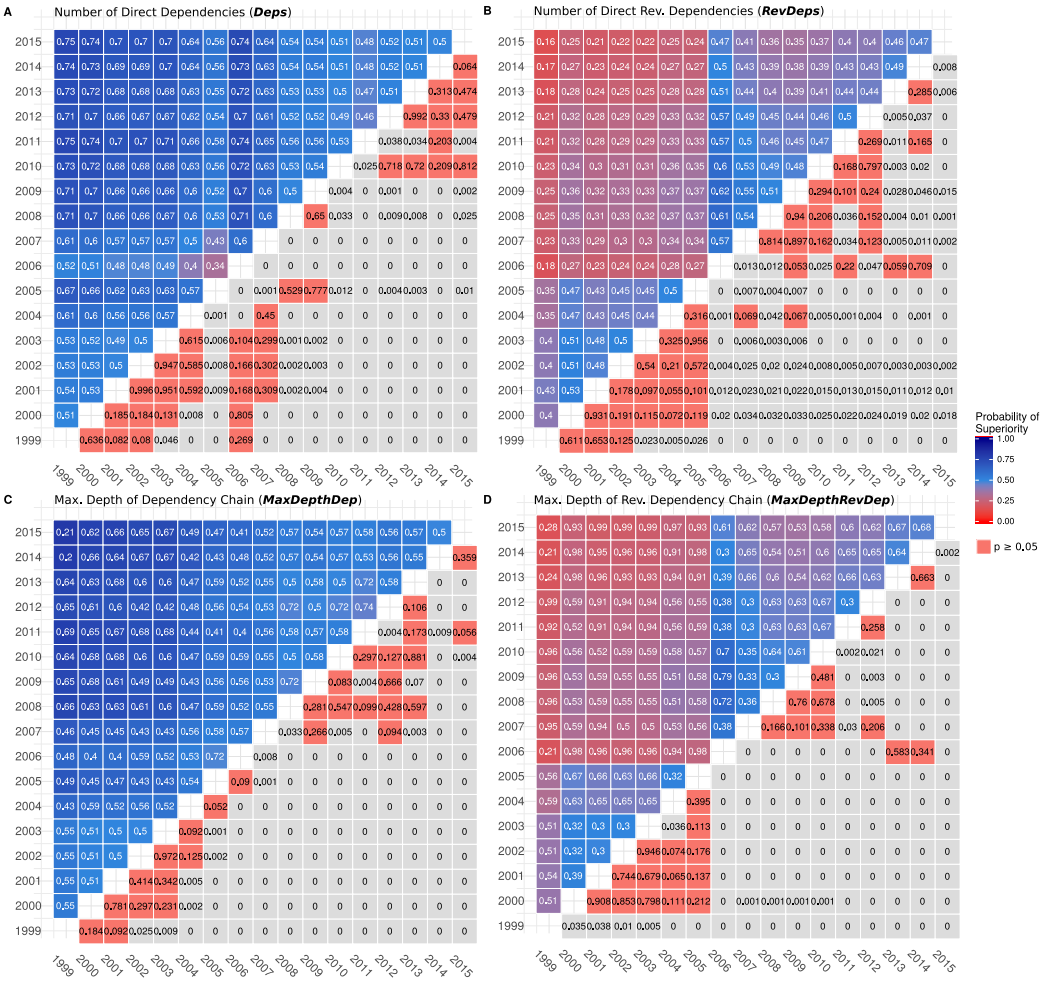

Figure 9: *Probability of superiority* $\hat{P}(y \geq x)$ (in the upper triangles, with the corresponding p-values in the lower triangles) for the number of dependencies (A), number of direct reverse dependencies (B), max. depth of dependency chain (C), and max. depth of the reverse dependency chain (D) per package, by year group.

in the lower triangles.

The results were of varying quality. Due to the extremely small size of the respective groups, we omitted the years 1997 and 1998 entirely. As can be seen in the figure, for all the groups up to 2003 (100 packages) and 2004 (174 packages), and to an extent 2005 (337 packages) as well, the tests we ran had low statistical significance, due to the small number of packages and their generally similar behaviour. Comparison with the packages of 2006

proved contrary to the trend prevailing across the marketplace due to the sudden influx of a large number of Bioconductor packages. As explained in Section 6.1, about a third of all Bioconductor packages was published in that time, of which most were gathered in a few large clusters dependent on a small numbers of central packages. Consequently the data for 2006 were severely distorted, as is evident in Figure 9: the Bioconductor packages had few dependencies (mostly to the one central package, hence lower *Deps* compared to both older and newer packages), and were quickly deprecated and saw virtually no reuse (hence lower *RevDeps* and *MaxDepthRevDep* compared to both older and newer packages). Another observation from Figure 9 is that a few of the recent cohort groups were virtually undistinguishable: the 2012–2015 cohorts in terms of *Deps*; the 2008–2012 cohorts in terms of *RevDeps*; the 2009–2013 in terms of *MaxDepthDep*; and the 2007–2010 in terms of *MaxDepthRevDep*.

For greater differences in age, however, we obtained results with a very high degree of confidence that confirm the original supposition, i.e., that more recent packages have more direct dependencies on other packages, are at the end of a longer dependency chain, and conversely provide fewer reverse dependencies and in turn support shallower chains of reuse. It can be observed that 2008 appears to form a cut-off point for the direct dependency metrics, after which they remain relatively stable. This is not the case for the reverse dependency metrics, where the downward trend continues. It is also apparent that the values for both reverse dependency-based metrics are virtually identical.

From Figure 9, three distinct periods in dependency behaviour can be seen: the "early period" until 2005, a "transitional period" in 2006–2008 marked by the sudden growth of Bioconductor, and the "current period", beginning in 2008/09 up to 2015. The number of packages in the "deeper" positions has increased over the packages published before 2008, but that that growth has stalled since, and the figures for the last five years are more or less stable both for the forward as for the reverse dependencies. Given the explosion in packages published in the same period, this is remarkable, and possibly indicates that the "food chains" in R have reached their maximum level. It is open to question why this is so. This level could represent a maximum sustainable complexity level for the ecosystem in terms of a balance of forces, i.e., reuse of other packages versus the effort needed to maintain them on account of their complex dependency structure. Alternatively, given the relatively narrow focus of the R ecosystem, it is may simply represent a

maximum necessary level.

Part of the answer certainly lies in another observation: the most frequently re-used packages, i.e., those with high *RevDeps*, are mainly located very near the core: of the 410 packages with more than 10 *RevDeps* across the ecosystem, 35.8% have *MaxDepthDep* of level 0, 18.3% of level 1, and 13.4% of level 2.

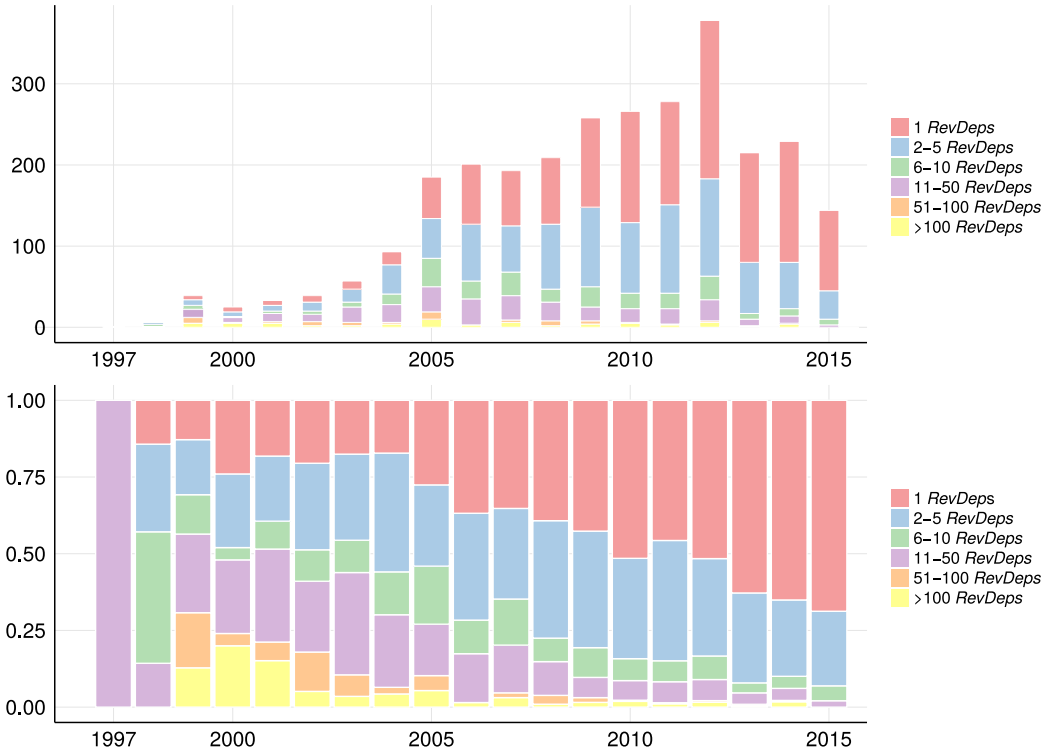

Figure 10: Distribution of packages published each year, grouped by the number of the reverse dependencies (*RevDeps*) upon them, in absolute numbers (top) and as proportion within each year (below)

6.3.2. Emergence of the Dependency Network

To determine how the present dependency network was built up, we examined the packages with *RevDeps* >0 according to their age, as shown in Figure 10, on the basis that packages with many reverse dependencies are those that by definition have supported the expansion of the ecosystem. It is apparent that the rate of new packages with reverse dependencies has

dropped significantly in the 2013–2015 period across all categories. This is possibly due to the fact that more recent packages have not yet become established and well-known so that they are more frequently reused, a supposition supported by plots B and D of Figure 9, where (the probability of) reuse appears to rise with package age.

The lower chart of Figure 10 furthermore indicates that reuse has been declining in relative terms as well for an even longer time, especially among the high-reuse categories. While it is clear that the present dependency network is a gradual evolution, with "important" packages continuing to be added over time to the present day, given that the CRAN repository more than doubled in size in the 2013–3015 period, the drop in reuse of packages published during the same period is significant.

It appears therefore that the essentials of the R dependency network (the packages with more than 10 *RevDeps*), as it was at the end of 2015, had been in place already at the end of 2012. 2012 itself represents a significant outlier in this regard: as can be seen in Figure 2, there was an abrupt spike in the number of new entries into the ecosystem, and Figure 10 makes clear that these new arrivals also included much new functionality, that quickly became reused.

6.3.3. Dependency-driven Package Obsolescence

In this regard we also examined whether possible changes in the dependency network played a role in the deprecation of the "archived" packages. We discovered that no "archived" package depended on another deprecated package, and that the dependencies used by the now "archived" packages throughout are essentially the same as those used by the currently active ones. Even for the four packages—*fEcofin*, *rcom*, *RGtk*, and *Rlibstree*—removed altogether rather than archived, the situation is mixed: of the 18 packages dependent upon them, half remained marked as active, including one case (*R2PPT*) where the sole dependency was to a removed package (*rcom*). The only common feature of the archived packages in this group was the fact that they had last been updated no later than 2010, whereas the packages marked as still active had their last release version in 2012 or later.

This eliminates the mere "archiving" of dependencies as a reason for the deprecation of packages, and implies that the reason for package deprecation is lack of maintenance on behalf of their developers in order to keep their contributions up-to-date with the packages they depend on, or, more importantly, the latest R releases. It would therefore appear that the main force

for package deprecation in the R ecosystem comes from the R platform itself, rather than changes in the marketplace. Overall, the "archived" packages did not differ in any way in their dependency metrics from their currently active siblings, except in so far as mirroring the trends shown in Figure 8 and having lower average values.

Summing up, we have demonstrated the dominant role of a relative handful of packages, that were published early in the ecosystem's lifecycle and have persisted since. We have also shown that the R marketplace has a mostly shallow dependency structure, whose growth has levelled out, but is capable of supporting a large number of add-on packages. A high turnover rate exists among packages, but this does not affect the dependency network, whose members are rarely removed. It also suggests that breaks in compatibility with the R platform, but not dependency obsolescence, are responsible for package deprecation.

7. Community Characteristics

7.1. Community Composition and Evolution

The first step towards addressing **RQ2** is assessing the size and structure of the R ecosystem's community. After cleaning up our author data, we determined the presence of 10,972 authors in the CRAN community and 1,700 in the Bioconductor community. 133 authors were involved in the 45 packages published in both repositories. Since the versions published in each repository had documentation of varying completeness, 64 of them appeared only in CRAN and 123 only in Bioconductor; only 54 appeared in both repositories. In addition, a further 342 authors contributed packages to both repositories. These 475 authors are considered as a separate group in the remainder of this section, under the label "Both". Likewise, the "CRAN" and "Bioconductor" groups will contain the authors who have been active *only* in the respective repository.

The next step involved determining the active status of community members. As we did not consider data from sources like Github, we relied on indirect evidence: authors who have participated only in packages that are *not* in the current release versions of the two repositories, that have been deprecated due to lack of maintenance, are marked as "inactive", whereas those who are mentioned in up-to-date packages are regarded as "active". This obviously is less than ideal, given that out of a group of five authors in a package only one may be indeed active, but given the low numbers of

authors per package, the fact that most authors have participated in only one package, and the tendency of less active authors to work closely in relatively self-contained groups (see below), we feel that this still represents a reasonable indicator of the general community trends. For the same reason, we consider as the "date of departure" from the community the date of the final contribution of an author marked as being "inactive".

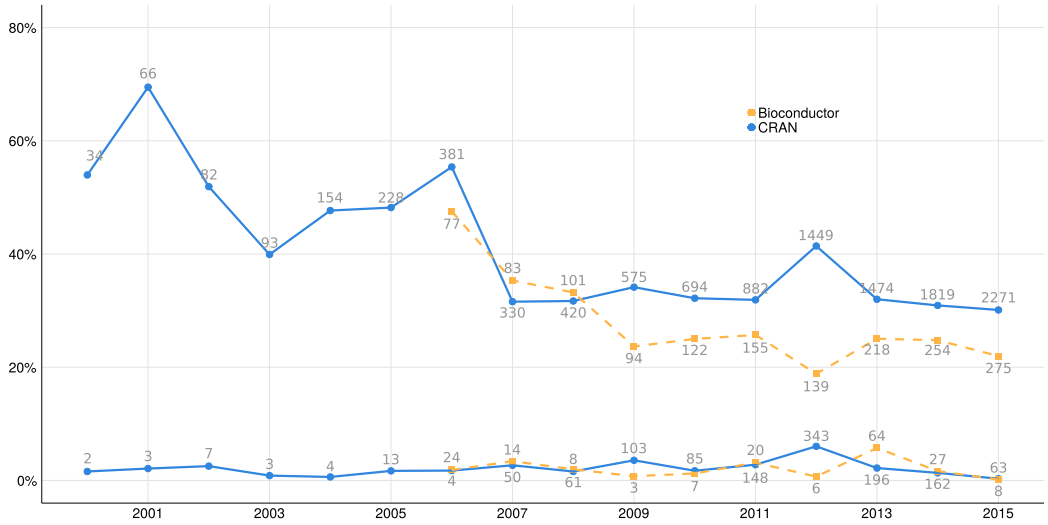

Figure 11: Acquisition (top) and turnover (below) rates for of new contributors by year in CRAN (blue solid line) and Bioconductor (yellow dashed line). The initial years of each repository are not included as outliers. Each node is labelled with the number of new members (acquisition diagram) and departures (turnover diagram).

Accordingly, 88.3% of authors in both CRAN and Bioconductor communities were marked as "active". From this, we calculated acquisition and turnover rates for the communities of the two repositories, as can be seen in Figure 11. For CRAN, the annual acquisition rate in the community has remained stable since 2006 at ca. 31%, with an annual turnover rate of ca. 2%. Bioconductor also appears to have achieved a relatively stable acquisition rate at ca. 25% with a turnover rate that is fluctuating but generally is of the same level as for CRAN. What is interesting here is that both repository communities took some time to settle into these rates, and the obvious conclusion is that in terms of community growth, both have entered a "mature" period since ca. 2007. 2012 represents an outlier here just as in the dependency metrics.

Regarding the turnover rate, due to the way we calculated "active" authors it reflects more on package management in R, where a package is retained as "active" regardless of other factors as long as it is considered compatible with the latest R release, rather than on actual level of participation in the ecosystem. However, given that a majority of CRAN as well as Bioconductor packages are developed as single-author packages, as demonstrated in the next section, the low turnover rate remains indicative of a strong commitment by the community to the ecosystem.

A further criterion we used to categorize authors is the type of their contributions, e.g., their role in creating or maintaining a package, the frequency and size of their commits, etc. As the available documentation in this regard varies considerably in content, style, form, and level of detail across the various packages, or even within the same version of a package, we were only able to assign a simple role distinction between "contributor" and "maintainer", meaning someone mentioned explicitly in the "maintainers" field, to which was added a qualification of "author" when a contributor was present in the very first release version of a package. This classification is neither exhaustive nor perfect, but it allowed a basic categorization of author behaviour. Other than that, the degree of contribution and the actual role of each contributor, e.g., whether an author was the original creator of a package written in S, ported by a second author into R with the possible assistance of further contributors, were not examined further. Such distinctions are interesting, but would have required far more effort at parsing the available documentation, and would exceed the scope of this study.

Based on this rough classification, 44.8% of CRAN members, 49% of Bioconductor members, and 68.8% of members active in both repositories were classed as both "authors" and "maintainers" of packages, while 40.8% in CRAN, 42.7% in Bioconductor, and 25.7% in the "Both" category were present as simple "authors". The first group represents the true mainstay of the community, indicating members with a very close relationship, to the point of ownership, of a package. The "authors" group is also important, but includes an undetermined number of people who were only casual or secondary contributors to a package, or whose work was reused, and more effort has to be expended to determine the true extent of their contributions to the ecosystem. Among the remainder, contributors who came along later in a package's lifecycle, an interesting subgroup are the 1.5–2% of the overall community who eventually became "maintainers".

7.2. Author Activity Metrics

Next we proceeded to examine *author activity*, i.e., how authors interact with the ecosystem. In this context the ecosystem is represented by two networks: the *author collaboration* network between authors, and the bipartite *author contributions* network between authors and packages.

In the author collaboration network, we assess the individual developer's collaboration with other developers. The primary metrics extracted from our data were:

- *Collaboration instances* (*ClbInsts*), i.e., an occasion of two authors collaborating in a specific package.
- *Collaboration cases* (*ClbCases*), i.e., a package with at least two authors collaborating.
- *Collaboration pairs* (*ClbPairs*) represents a unique pairing of authors, which can have many collaboration instances across several packages.

Of interest here are particularly two types of collaboration patterns, namely *ClbPairs* that are present in different packages over time, indicating a work team, and developers who have many different *ClbPairs* across different packages. The latter represent a distinct group of community members: *networkers*, members who bind the community together.

To assess each author's position in the community, a number of metrics are used. Thus a high number of *ClbInsts* or *ClbPairs* is indicative of a high *degree centrality*, indicating a developer with strong social ties in his or her immediate community. Furthermore, *closeness centrality* and *betweenness centrality* can be used to quantify an author's position in the overall community.

In the author contribution network, we assess the individual developer's participation in and impact on the marketplace. The primary metric here are the number of packages contributed to (*Ctbs*). It is a metric that illustrates the connections between the developer community and the marketplace, and hence of considerable importance for the evolution of the ecosystem. From the ranking of packages by reverse dependencies and downloads that we undertook in Section 6, we can also broadly measure each author's impact on the ecosystem. In addition, it also serves as an indicator of developer involvement with the ecosystem, because developers who contribute to many packages over time have a higher stake in the ecosystem's success.

In order to test whether the involvement suggested by frequent collaborations ("networking") and numerous contributions ("productivity") is truly indicative of a commitment to the ecosystem, we correlate the above metrics with author age and author active status. We expect authors who have been active in the ecosystem longer, and remain active, to also display correspondingly higher productivity and networking activity.

7.3. Collaboration Network

The overall author collaboration data for the community are summarized in Table 7.

	CRAN	**Bioconductor**	**Both**
Authors	10,566	1,235	475
Authors with *ClbInsts*	9,135 (86.5%)	1,090 (88.3%)	462 (97%)
Total *ClbInsts*	51,704	8,184	11,881
Total *ClbPairs*	47,650	2,679	6,118
Total *ClbCases*	4,776	2,022	2,364
ClbInsts per author quartiles	1–3–6	1–3–4	4–8–18
ClbPairs per author quartiles	1–3–5	1–2–4	3–6–14

Table 7: Overview of author collaboration metrics, by author community

	CRAN	**Bioconductor**	**Both**
Authors	2,330	306	369
Total *ClbInsts*	15,365	6,439	9,041
Total *ClbPairs*	11,311	934	3,278
Total *ClbCases*	2,593	1,542	2,121
ClbInsts per author quartiles	2–5–10	3–5–9	3–8–16
ClbPairs per author quartiles	2–3–6	2–3–5	2–5–10

Table 8: Overview of author collaboration metrics for the reduced collaboration network, by author community

In all communities, a very high proportion of authors had at least some form of collaboration with another author, or, at least, reused and referenced someone else's work in their own package. The "average" author in CRAN and Bioconductor has collaborated at least three times (*ClbInsts*), with three other authors (*ClbPairs*). Inevitably, there is some overlap in the data: there are 9,073 *ClbInsts*, 5,074 *ClbPairs*, and 2,114 *ClbCases* where one author is active exclusively in CRAN or Bioconductor and the other is active in both repositories. The data is further skewed due to the presence of a few big outliers in CRAN, such as *spatstat* with its 167 listed authors, many of whom only appear in this one package, and which alone accounts for 13,875 *ClbInsts*. *spatstat* is a "comprehensive open-source toolbox" with over 2,000 functions, and its authors include statisticians whose work is used, as well as some R developers whose packages are imported. Obviously the majority of these "authors" are not in reality R developers. Furthermore, 7,681 authors, or 62.5% of the entire R community, only collaborate with other authors in a single package.

To detect the core of R's collaborative network and avoid such outliers, we examined a *reduced collaboration network* of authors who have collaborated with others in more than one package. The reduced network comprises 3,005 authors active in 9,832 packages. The metrics for this are presented in Table 8.

Over 80% of the authors in the reduced network had some collaboration outside it, but only 139 authors had collaborations exclusively with people outside the network (in 2–4 different packages). While *ClbPairs* dropped precipitously, the *ClbCases* of each author remained relatively stable: 2,060 authors remained unaffected, 627 authors lost one package, 247 lost two, and 71 lost between three and seven. This shows that the technique is effective in isolating the *core collaborative network* of the R ecosystem.

A ranking of importance within this network can be undertaken by any of the metrics introduced above. However, *ClbInsts* per author is imprecise, as it measures volume of collaboration but not variety. The same applies to *ClbCases* per author, as can be seen in the following example. The top global contributor in the R ecosystem is Bioconductor's Manhong Dai with 1,207 packages, followed by the "Bioconductor Core Team" (1,161 packages). All 461 *ClbCases* of Manhong Dai in the Bioconductor repository were with the "Bioconductor Core Team" alone. Furthermore, although we know the members of the "Bioconductor Core Team", in many Bioconductor packages this was the only author information included, and we were forced to con-

sider the "Bioconductor Core Team" as a separate author. Consequently, packages where the "Bioconductor Core Team" alone or the "Bioconductor Core Team" and one or more other authors are listed as authors, it is impossible to know, without examining their commit history, how many authors collaborated in reality.

Author	**Joined**	**Pkgs**	**ClbInsts**	**ClbPairs**	**ClbCases**	**Repo**
M. Mächler	1999	62	572	418	56	Both
A. Zeileis	2000	48	486	380	46	CRAN
B. Ripley	1998	44	454	363	41	CRAN
K. Hornik	1998	71	412	308	63	CRAN
R. Bivand	2000	23	351	307	20	CRAN
A. Baddeley	2002	6	294	281	6	CRAN
B. Rowlingson	2000	18	321	279	17	Both
M. Kuhn	2006	14	297	278	12	CRAN
M. Stevenson	2006	3	270	265	3	CRAN
R. Turner	2002	9	281	263	5	CRAN

Table 9: Overview of the top *networkers* in the R ecosystem, ranked by *ClbPairs*

Due to the unreliability of our dataset, and since our purpose is to find the authors who "bind the community together", the main collaboration metric we use is *ClbPairs* per author, in other words, the number of individual community members with whom an author has collaborated. Table 9 shows the top ten *networkers* of the R ecosystem, ranked by *ClbPairs*, with the relevant author activity metrics.

Focusing on the *ClbPairs* metric, we have determined 51,249 individual author pairings in our dataset. Almost 94% are active in a single package, leaving 3,166 pairs, involving 2,156 individual authors, as *work groups* who collaborate frequently across several packages. A more detailed examination is given at Table 10.

Once more it is apparent that a small core of highly active authors, disproportionately drawn from older members of the community, form the most active element of the collaboration network. This is all the more so since the majority of authors present in the most active pairs are also present in the less active ones as well.

As shown in Table 7, a number of community members had no collaborations with the rest of the community, throughout the period examined.

	Pairs	Individual Authors	Author Join Date	Age Difference
in 1 package	48,083	2,156	2010-02 – 2012-11 – 2014-09	178–920–2,168
in 2–5 packages	2,932	2,129	2007-05 – 2010-12 – 2013-02	22–626–1,617
in over 5 packages	234	155	2003-05 – 2006-05 – 2009-06	77–397–1,079

Table 10: Overview of the author pairs in the R ecosystem

About 13% fall in this category, representing the proverbial "lone wolves", 90% of whom were active in CRAN, and 80% of whom joined the ecosystem after 2010. The remainder of the community comprised a giant component ("central community") that included 47.5% of authors, and a remaining 39.5% of authors who formed 1,413 smaller clusters. These clusters were mostly small, with an average of 2.65 *ClbPairs* per author, and quartiles at 2–3–4 authors per cluster. The two largest clusters had 21 authors. As a result, this part of the community has a very high clustering coefficient of 0.991.

The central community comprised some 84% CRAN authors and 10% Bioconductor authors, with the remainder falling under the "Both" category. It had a surprisingly high clustering coefficient at 0.836, with an average of 15.35 *ClbPairs* per author. This is largely attributable to the presence of packages such as *spatstat* (167 authors), *DescTools* (92 authors), and other packages with a large number of authors—19 packages had 20 or more authors, and 134 had 10 or more—many of whom only appear there, and form tightly knit clusters. With the aid of *GEPHI* we performed a betweenness centrality, closeness centrality, and eigenvector centrality (PageRank) analysis on the central community. Table 11 presents the quartiles calculated for each method. While the CRAN and Bioconductor communities are again comparable, the "Both" group deviates markedly. Its members have a far higher betweenness centrality, which is partly due to the fact that the members of this group link Bioconductor to CRAN, but also because of the considerably higher contribution volume of this group, as will be seen in the next section, and which leads to more opportunities for collaboration. They are also a more interconnected group, as shown by their higher eigenvector centrality

values.

	Total	**CRAN**	**Bioconductor**	**Both**
Betweenness centrality quartiles	0–0–1,736	0–0–26.7	0–0–0.5	1,006–11,670–48,430
Closeness centrality quartiles	0.1668–0.1957–0.2241	0.1667–0.1961–0.225	0.1633–0.1872–0.2078	0.1700–0.2056–0.2334
PageRank quartiles	8.866×10^{-5}–1.327×10^{-4}–1.897×10^{-4}	8.783×10^{-5}–1.307×10^{-4}–1.851×10^{-4}	8.025×10^{-5}–1.155×10^{-4}–1.538×10^{-4}	1.417×10^{-4}–2.144×10^{-4}–3.326×10^{-4}

Table 11: Overview of centrality metrics for the central CRAN–Bioconductor author community

In total, 1,718 members had a betweenness centrality value over 0 in the central community. The top 10 are presented in Table 12, and are mostly names familiar from Table 9.

We consider these 1,718 authors as the *networkers* of the R community. It is notable that Bioconductor authors are under-represented in this group in relation to their numbers in the core community, with only 148 authors, and most of them found in the lower ranks. Conversely, the "Both" group are considerably over-represented with 278 authors, including, as can be seen in the table, among the top members.

7.4. Contribution Network

The contribution network is a bipartite network, with the authors on the one side and the packages they contributed to on the other. Table 13 summarizes basic author contribution metrics in terms of contributions (*Ctbs*) and number of authors per package, by community, at the date of our data collection. The CRAN and Bioconductor communities display a very similar behaviour overall, while the authors present in both repositories are, as is to be expected, more productive. The pattern that emerges in Table 10 for the contributions of author pairs is repeated here: the large majority of ecosystem members is active in one package—indeed, a significant portion of packages have only a single author—and only a small percentage of authors is active in many packages.

Author	Joined	ClbInsts	ClbPairs	Betw. Centr.	PageRank	Repo
M. Mächler	1999	572	418	2,095,265	2.82×10^{-3}	Both
B. Ripley	1998	454	363	1,224,752	2.06×10^{-3}	CRAN
R Core Team	2005	171	147	1,127,202.3	1.74×10^{-3}	CRAN
K. Hornik	1998	413	309	885,019.4	1.85×10^{-3}	CRAN
W. Huber	2004	212	91	868,497.4	1.49×10^{-3}	Both
A. Zeileis	2000	486	380	864,524.9	2.26×10^{-3}	CRAN
D. Eddelbuettel	2002	245	202	840,108.2	1.71×10^{-3}	CRAN
R. Gentleman	2001	268	111	839,046.7	1.52×10^{-3}	Both
B. Bolker	2001	338	228	727,639.4	1.6×10^{-3}	CRAN
H. Bengtsson	2005	132	114	686,013.4	1.12×10^{-3}	Both

Table 12: Overview of the top authors, ranked by betweenness centrality, in the CRAN–Bioconductor core community

The *Ctbs* metric gives a good idea of an author's direct involvement in the ecosystem, but not of his or her impact. For instance, an author may have only contributed in a single package, but this may be a heavily reused package, requiring constant maintenance in turn. Consequently we tried to quantify the individual author's impact on the marketplace by using the metrics of the packages he or she has contributed to: the *average daily downloads* (*AvgDls*) of a package, and *reverse dependencies* (*RevDeps*) on a package. The distribution of the tallied figures is given in the lower part of Table 13. This shows that the vast majority of authors have a negligible impact on the ecosystem, especially when one considers that the actual numbers in many cases would have to be "shared" among the many authors of a package.

The disparity is even more pronounced when examining impact of each of the main collaboration network groups in Table 14. The data show a very strong correlation between membership in the core community, higher contribution, and, most importantly, usefulness to the ecosystem in terms of frequently reused and downloaded packages. This does not mean that there are no such contributions in the other groups—for instance, the "lone wolves" contributions include package *snowfall* with 45 *RevDeps*, and the "independent clusters" the *MeSHDbi* package with 208 *RevDeps*—but they

	CRAN	Bioconductor	Both
Authors	10,566	1,235	475
with 1 *Ctbs*	7,513 (71%)	855 (69.2%)	45 (9.5%)
with 2 *Ctbs*	1,580 (14.9%)	193 (15.6%)	121 (25.5%)
with >5 *Ctbs*	380 (3.6%)	68 (5.5%)	125 (26.3%)
Packages contributed to by resp. community	8,510	3,196	3,136
with 1 author	3,801 (44.7%)	1,357 (42.5%)	772 (24.6%)
with 2 authors	2,237 (26.3%)	861 (26.9%)	1065 (34%)
with >5 authors	405 (4.8%)	207 (6.5%)	303 (9.7%)
RevDeps per author quartiles	0–0–1	0–0–2	0–3–24
AvgDls per author quartiles	7.38–9.89–23.2	8.31–13.36–30.78	17.89–35.77–127.8

Table 13: Overview of basic author productivity metrics, by author community

are the exception rather than the rule.

Finally, another factor affecting author contribution metrics is the tendency of authors to create clusters of related packages. We found that in the R ecosystem, in 17.6% of all dependency pairings, these were between packages that had *at least* one author in common.

7.5. Author Activity Trends

The development of the basic author activity metrics over time is shown in Figure 12, where the collaboration metrics have been condensed to whether a new author is engaged in collaboration or not. From the chart emerges a twin trend of fewer authors per package and declining levels of collaboration over time.

To test the hypothesis that older community members are also more active, we divided the authors dataset in cohorts by year of joining the R community. Figure 13 shows the distributions of author productivity and networking metrics by repository. To compare the different communities on an equal footing, we divided each author's productivity metric with the number of packages contributed to by the respective community from Table 13, and each author's networking metric with the respective number of

	Central community	**Lone wolves**	**Independent clusters**
Authors	5,837	1,589	4,850
with 1 *Ctbs*	3,443	1,276	3,694
with >5 *Ctbs*	509	11	56
Packages contributed	7,486 (58.5%)	2,083 (16.3%)	3,149 (24.6%)
corresp. *RevDeps*	29,947 (93.8%)	557 (1.7%)	1,410 (4.4%)
RevDeps per author quartiles	0–1–6	0–0–0	0–0–0
corresp. *AvgDls*	214,887.6 (81%)	19,375.33 (7.3%)	30,512.07 (11.5%)
AvgDls per author quartiles	8.39–16.23–53.33	6.64–7.98–10.8	7.14–8.56–13.52

Table 14: Overview of author impact metrics by network groups

collaboration pairs generated by the respective community in Table7.

Disregarding the outliers represented by the early data and some packages like *spatstat* (visible as a line of outliers across the year cohorts in the CRAN column in Figure 12), all three communities share the trends of reduced collaborations and contributions among the newer contributors, particularly when compared with the older community members. This broad tendency of the R community is confirmed by a *probability of superiority* analysis by year of the authors' joining the community, as shown in Figure 14.

In part this is to be expected, as contributors who joined early have naturally been around longer and have had time to become more involved and contribute more, but at least in terms of productivity, the increasing predominance of one-package authors indicates a shift in composition of the community: instead of first-generation "ecosystem-builders", newer users tend to "live off" the already established ecosystem, publishing one or two packages for their own needs. This is demonstrated not only in the increasing presence of "lone wolves", as demonstrated in Section 7.3, but also in the dependency flows between packages examined in Section 6.2.

Furthermore, Figure 15 shows that in the more recent years, the number of collaborations from within the same age cohort has steadily increased both in absolute terms and as a proportion of the total number of collaborations.

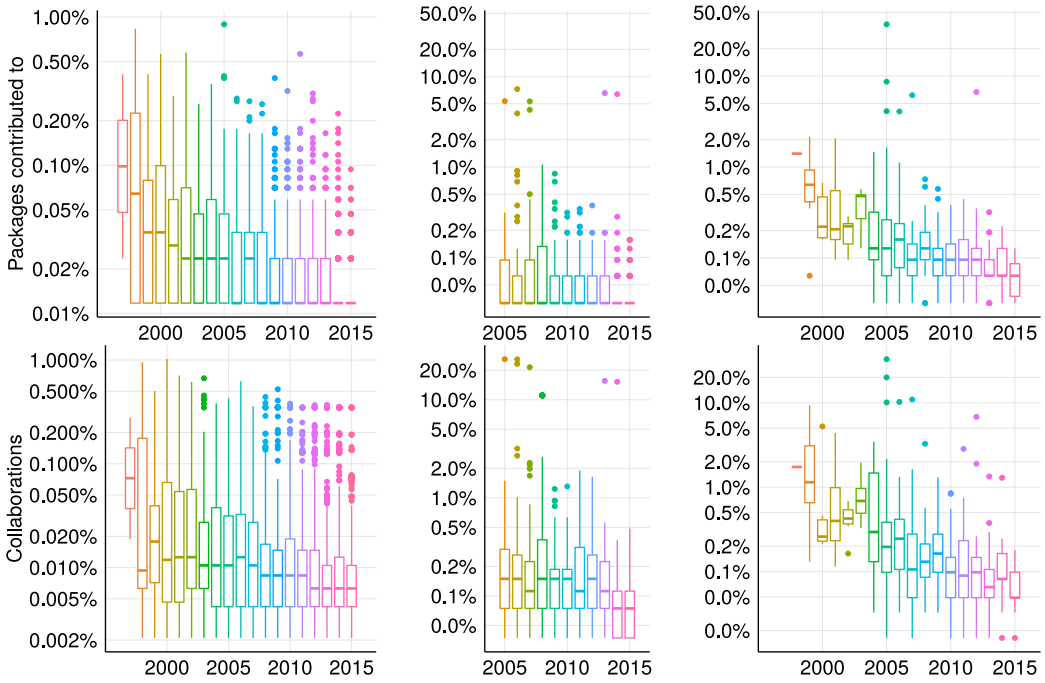

Figure 12: Author contributions (as percentage of total *Ctbs*) and collaborations (as percentage of total *ClbPairs*) for authors active in CRAN (left), Bioconductor (centre), and both repositories (right), grouped by year of entering the R ecosystem.

On the one hand, this can mean that the number of collaborative packages is increasing, or that perhaps the average size of the author group has been growing. Our data, as shown in Figure 13, disproves this supposition; quite the contrary, single-author packages are on the rise. Of course, given the growing number of packages published in the last few years, it is natural that the absolute number of collaborations in the same cohort rises, even if the latter's overall proportion is reduced. The rapidly growing "relative" presence of same-year collaborations however, seems to corroborate the more "self-centred" nature of the newer members' contribution, who limit their involvement in the ecosystem to their own few contributed packages and their co-authors.

We also found that of the *ClbPairs* in CRAN, 8,150 (17%) are between authors who "joined" on the same day, in the same package(s). Excluding these values, the median distance between the dates where two collaborating authors joined is 1,256 days (3.4 years). The precise result is skewed by the

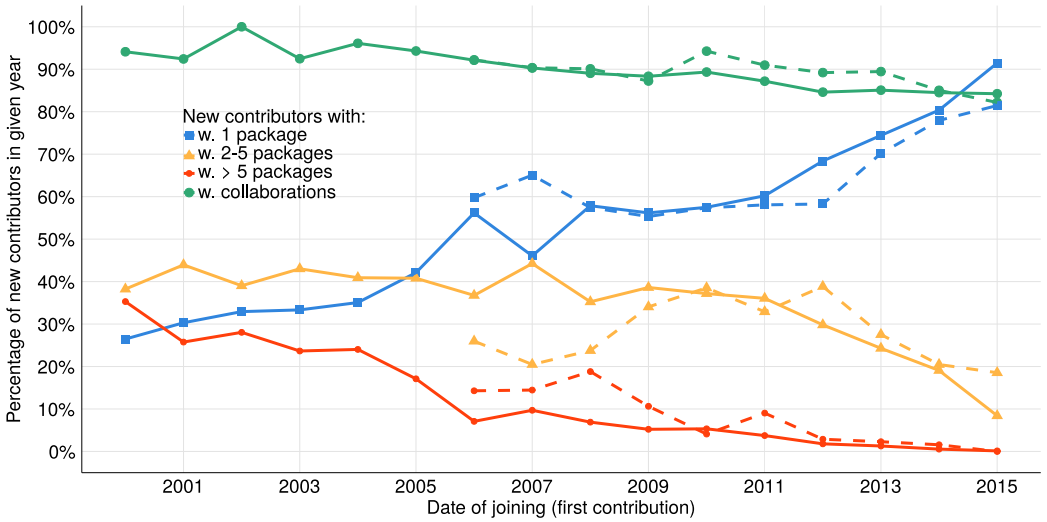

Figure 13: Evolution of contributor statistics by year cohort in CRAN (solid line) and Bioconductor (dashed line).

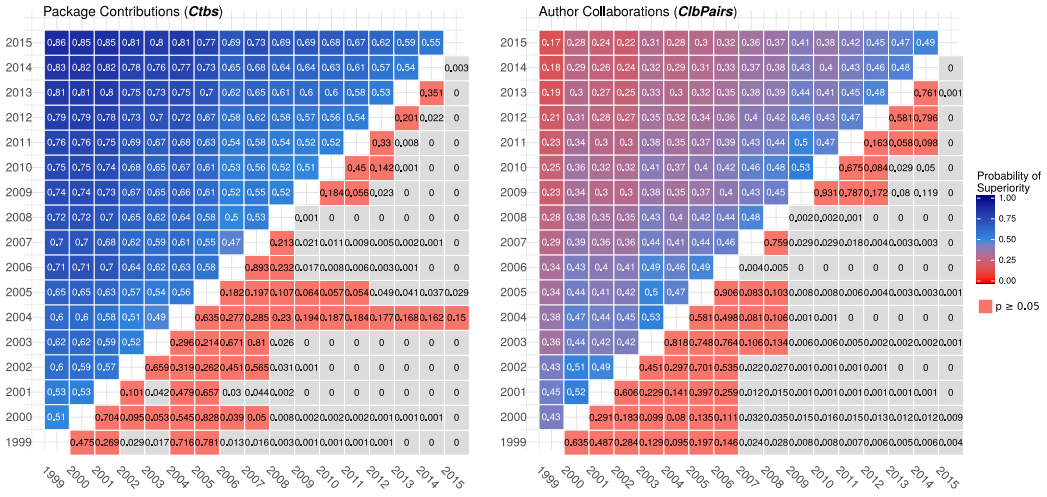

Figure 14: *Probability of superiority* $\hat{P}(x \geq y)$ (in the upper triangles, with the corresponding p-values in the lower triangles) for the number of packages contributed (left) and the authors collaborated with (right) by year groups.

predominance of recent authors in the data set, but even when examining the community by yearly author cohorts, as shown in Figure 15, it is evident that even older authors, from the 1997–2000 period, collaborate frequently with newer ones of the 2012–2015 period. In our view, this is a positive

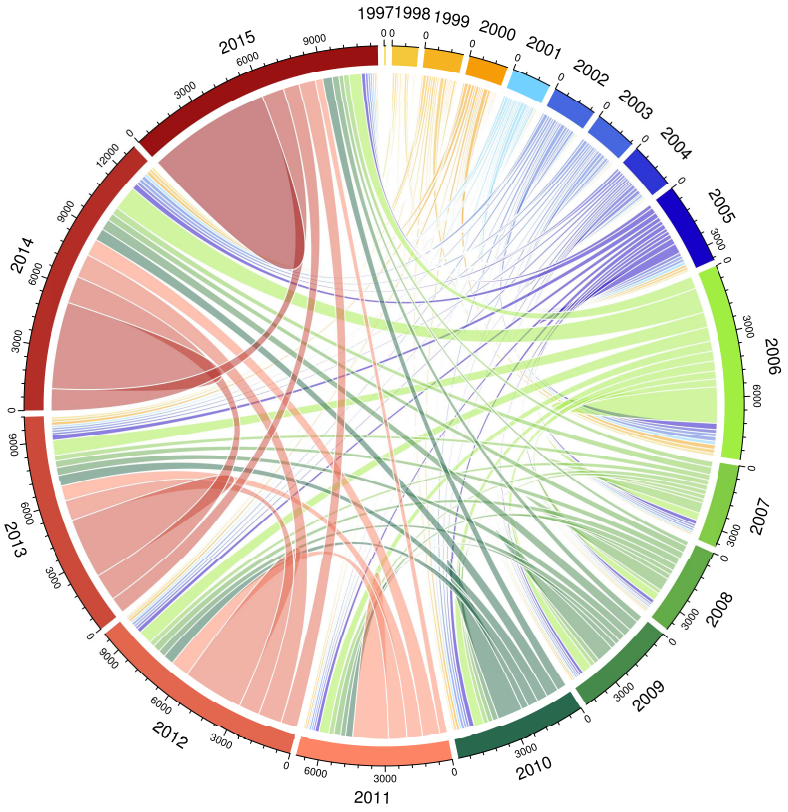

Figure 15: Collaboration between author cohorts by year in CRAN

indicator for a community where the older members are not only active, but also willing to welcome newer ones; conversely, the newer members of the community show willingness to collaborate with the older members and, more more importantly, to get involved in developing and maintaining already extant packages, as is demonstrated in Figure 16. Even the 61 authors who joined before 2000 in CRAN had about 25% of their work done with members who joined in 2012 or later, although in this case this is mostly due to a small set of authors, namely the members of the "R Core Community". These are the members of the "R Foundation", who have played a central role in the growth and maintenance of the R ecosystem, being responsible not only for the R core packages, but also for many of the most reused packages in CRAN as well [55]. An analogous role has been played in Bioconductor by the members of the "Bioconductor Core Team".

By way of summing up, in Figure 17 we have compared the basic author

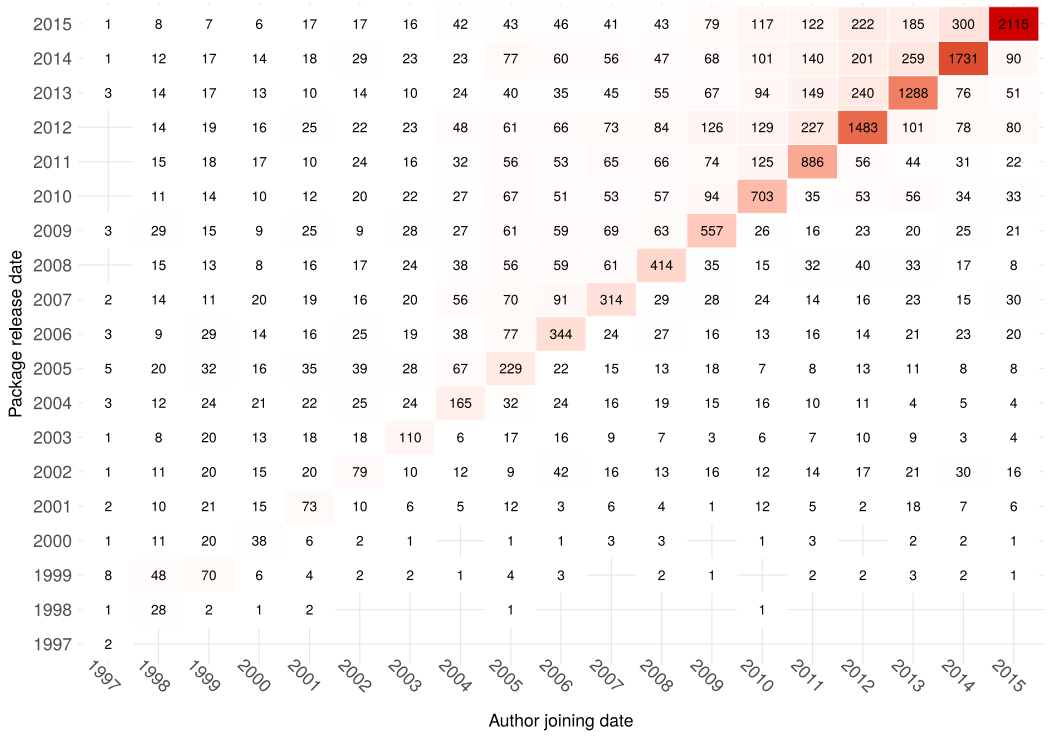

Figure 16: Instances of authors contributing to packages by year of author joining (x-axis) and year of package release (y-axis) in CRAN

activity metrics of the developer community from CRAN and Bioconductor with the most important groups we have identified so far: "networkers", "lone wolves", and the "core teams", the members of the "R Core Community" and the "Bioconductor Core Team".

8. Linking Metrics with Ecosystem Condition

As we have pointed out in Section 4, our ultimate aim is to extract qualitative insights and laws of behaviour that will allow us to model ecosystem dynamics and that will be applicable for all ecosystems. To begin with, to examine **RQ3** we will study the state and condition of the R ecosystem as it emerges from the metrics we have gathered.

8.1. Metrics Summarized

Understanding and modelling a software ecosystem is done in two levels: one internal, concerning individual ecosystem members, software packages

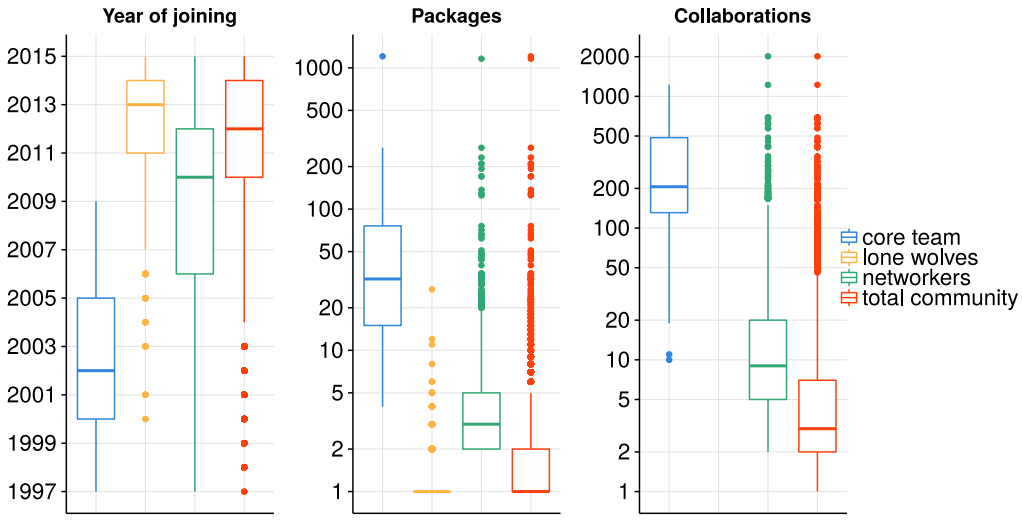

Figure 17: Author activity metrics (year of joining, packages contributed, collaborations with other authors) by author group: members of the CRAN and Bioconductor core teams (blue, outer left), "lone wolves" (yellow, centre left), "networkers" (green, centre right), and the total community (red, outer right) for comparison.

and community members, and one external, the ecosystem regarded in its entirety. Both approaches entail two types of metrics: static metrics describe the characteristics of the ecosystem and its components at a given time; dynamic metrics describe the evolution of the ecosystem's characteristics over a period of time. In our case, the static metrics (cf. Figure 18) were derived from a package documentation and content, and their aggregation over time by the timestamps of the individual versions gave us their dynamic dimension. The metrics in Figure 18 represent the main attributes of a software ecosystem's members (software components and community members) and their activities and interactions. Similar metrics can be found in the frameworks proposed in literature [42, 31].

The basic quantitative metrics we gathered or calculated for the ecosystem are shown in Table 15. It is obvious that the marketplace and community members can be analysed in pretty much the same way, but the data available did not allow us to do this, e.g., we did not know when archived packages were removed from the "active" list. The quantitative metrics for individual packages are shown in Table 16, and for the community members in Table 17. Community members are further categorized by role, as discussed in

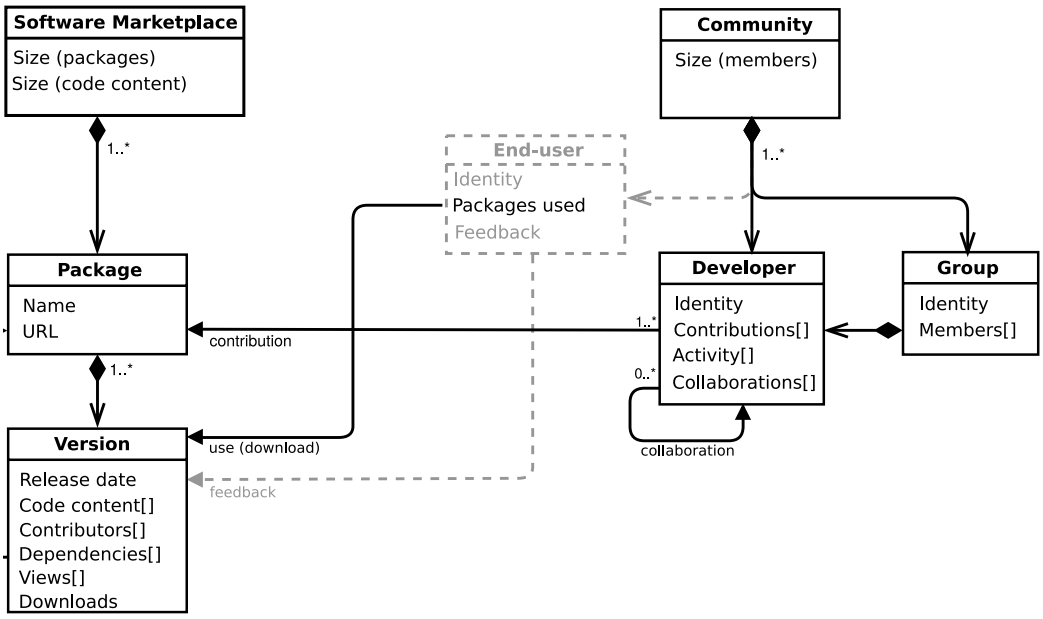

Figure 18: Diagram of the primary metrics and the relationships between them, for the study of a software ecosystem. In this study, end-user impact is only examined indirectly through package downloads.

Metric	Definition
Overall marketplace size	number of packages
Active marketplace size	number of packages
Archived marketplace size	number of packages
Marketplace acquisition rate (*Growth in size*)	$\frac{\text{new packages over } \Delta T{=}T_b{-}T_a}{\text{extant packages at } T_a}$
Developer community size	contributors
Community acquisition rate (*Community growth*)	$\frac{\text{new contributors over } \Delta T{=}T_b{-}T_a}{\text{extant contributors at } T_a}$
Community turnover rate (*Author departures*)	$\frac{\text{inactive contributors over } \Delta T{=}T_b{-}T_a}{\text{extant contributors at } T_a}$

Table 15: Ecosystem-level quantitative metrics

Section 7.

Metric	Range	Quartiles	SD
Number of authors	0 – 167	1–2–3	2.668
Number of versions	1 – 182	2–3–7	8.805
Date of version release	1997-10-08 – 2015-12-31	2007-12-01– 2012-01-16– 2014-04-01	–
Avg. time (days) between versions (*Update frequency*)	0 – 1,539	2–100–273	154.926
Number of versions w. source code changes	1 – 182	1–3–6	8.600
Avg. time (days) between versions w. source code changes	0 – 2,064	0–80–183	194.525
Time (days) between first release and date of collection (*Lifespan*)	0 – 6,658	638–1,774– 2,952	–
Avg. download rate per day (*AvgDls*)	0 – 2,501	2.153– 7.405–9.647	106.336
Number of dependencies (*Deps*)	0 – 47	0–1–3	3.932
Max. depth of dependency chain (*MaxDepthDeps*)	0 – 15	0–2–6	3.143
Number of rev. dependencies (*RevDeps*)	0 – 1,029	0–0–0	22.312
Number of transitive reverse dependencies (*TrRevDeps*)	0 – 5819	0–0–0	156.721
Max. depth of reverse dependency chain (*MaxDepthRevDeps*)	0 – 15	0–0–0	1.379
Transitive impact (*TrImpact*)	0 – 529	0–0–0	15.904
Current/historical R code size (LLOC)	0 – 156,200	95–435– 1232	3,288.827
Current/historical C code size (LLOC)	0 – 470,600	120–492– 1529	15,862.45
R code size difference between current and original version (LLOC)	-56,730 – +75,970	0 – +4 – +166	1,961.782
C code size difference between current and original version (LLOC)	-88,000 – +177,600	0 – 0 – +74	8,405.324

Table 16: Package quantitative metrics for the R ecosystem (CRAN and Bioconductor repositories)

Metric	Range	Quartiles	SD
Active status	True/False	–	–
Dates of activity (version releases)	1997-12-10 – 2015-12-31	2010-04-01– 2012-12-21– 2014-09-19	–
Productivity (*Ctbs*)	1 – 1,207	1–1–2	16.440
Networking activity (*ClbPairs*)	0 – 2,013	1–3–6	38.853

Table 17: Community member quantitative metrics for the R ecosystem (CRAN and Bioconductor repositories)

8.2. Classifying the Ecosystem

Before beginning any attempt to examine the qualitative aspects of the R ecosystem based on the metrics above, it is important to place it in context. A common and "obvious" classification of SECOs, which we will adopt here as well, is by their business aspects, based on three criteria [49]:

1. **What motivates** the establishment and/or participation in a software ecosystem. Is it done for (monetary) profit or other reasons?
2. **How open** is a software ecosystem, i.e., how easily are new participants and their applications are allowed into the ecosystem by its owner/maintainer?
3. **Who owns** the software ecosystem and its components, are they proprietary or open-source?

These three classifications operate in parallel, highlighting different aspects of the SECO. The classification ranges from entirely closed, for-profit and proprietary ecosystems on the one extreme to fully open, non-profit and open-source ecosystems on the other, with every kind of hybrid in between. In some cases, e.g., OS-based ecosystems, various layers of the platform may be characterized by different levels of the above characteristics.

In R we have a case of a programming language-based, fully open-source ecosystem, which is "prima facie" non-profit, although of course many of the contributors act out of personal interest, they create or help maintain functionality that is of use to themselves, and the monetization of some aspects can not be excluded, as can be seen in Microsoft's entry into the R ecosys-

tem. Nevertheless, the fundamental nature of R is that of a community-driven, open-source ecosystem largely "developed by statisticians for statisticians", whose use does not require advanced programming skills [60], hence an ecosystem driven to satisfy specific user needs, rather than oriented towards offering tools and frameworks for developers to build upon.

As regards openness, both CRAN [15] and Bioconductor [1, 3] offer guidelines as to package content, structure, documentation, and compatibility that the member packages have to observe. Both repositories also provide a screening and monitoring process, at the beginning as well as at regular intervals in a package's lifecycle, which removes packages that are no longer up-to-date. Nevertheless this is not a high threshold, nor is there a tight oversight of developers' activities between releases, or of any changes that might break dependencies to other packages.

R is therefore in many ways a typical, volunteer-based open-source ecosystem, albeit distinguished by its specialized focus and the absence (until recently) of any other similar ecosystems in its chosen domain. This is important because free, open-source ecosystems display certain peculiar characteristics that have to be taken into account. For example, as past empirical studies have shown [58, 39], such ecosystems are dominated by a large number of "lone wolves" who work alone on their projects with no explicit interaction with other community members, and further by a large number of small "work groups" of a handful of developers who co-operate. A number of developers are more active in the community, co-operating with different members, and serve to bind the community together, whereas the majority focus on their own products. This is a pattern that is also encountered in R, as discussed in Section 7.

The average R user is not the typical "end-user" encountered in other areas of software engineering, who is presumed to be technically ignorant and only interacts with a system through a visual interface. Use of R requires a minimum of technical skills and understanding of programming principles. The average R user is thus somewhere between a developer and an "end-user". Nevertheless, we observe that R perfectly fits the characteristics Bosch [19] defined for an "End-User Programming Software Ecosystem": it is domain-oriented, with application developers that have a good domain understanding but not necessarily extensive programming experience, and who are focused on the "creative composition" of extant offerings to satisfy their particular needs, rather than create new functionality. This job is left to a small team of ecosystem-builders, who have sufficient motivation to do this.

However, as Bosch noted, the added value that each R user can generate for himself through the ecosystem is the main success factor for ecosystems of this kind, and in this regard, R is undoubtedly successful.

8.3. Health of the R Ecosystem

Having analysed the current and past state of the R ecosystem, and determined the driving factors behind its evolution, we move to the main point of **RQ3**: how can the current state of the R ecosystem be qualified? For this purpose we use the concept of "ecosystem health", first defined by [40] as comprising three aspects:

1. *Productivity*: "a network's ability to consistently transform technology and other raw materials of innovation into lower costs and new products".
2. *Robustness*: the ability to adapt to and survive unforeseen environmental changes.
3. *Niche Creation*: increasing diversity through creation of new niches, including the demise of old, well-established niches, that give birth to new ones.

Productivity encompasses the ecosystem's ability to grow and expand its content, whether in this is the creation of new software components, addition of functionality to existing ones, or the generation of additional ecosystem-related knowledge in the form of manuals, tutorials, etc. [42]. Productivity is measured by the *recent* activity in the ecosystem, and ideally should indicate continuous growth in most metrics. From the metrics we have analysed so far, the most relevant are the quantitative metrics *marketplace acquisition rate* (Figure 2), *marketplace use acquisition rate*, and *marketplace content acquisition rate*, all of which display steady, positive trends on the growth of the ecosystem, including *signs of "maturity"*: the growth rates in both the marketplace and the community have levelled out and are slightly declining, although they are still remarkable for CRAN, which grows by between a quarter and a third every year.

As we have not analysed any external sources, we cannot at this point quantify the growth of peripheral, add-on and derivative products. However, given the rise of "big data" in recent years, and the obviously growing interest in R among the end-users (rise in download rates, as shown in Figure 19), as well as the growing number of developers participating, it is not unreasonable to expect the picture to be similar there as well. Bug fix rate is also an

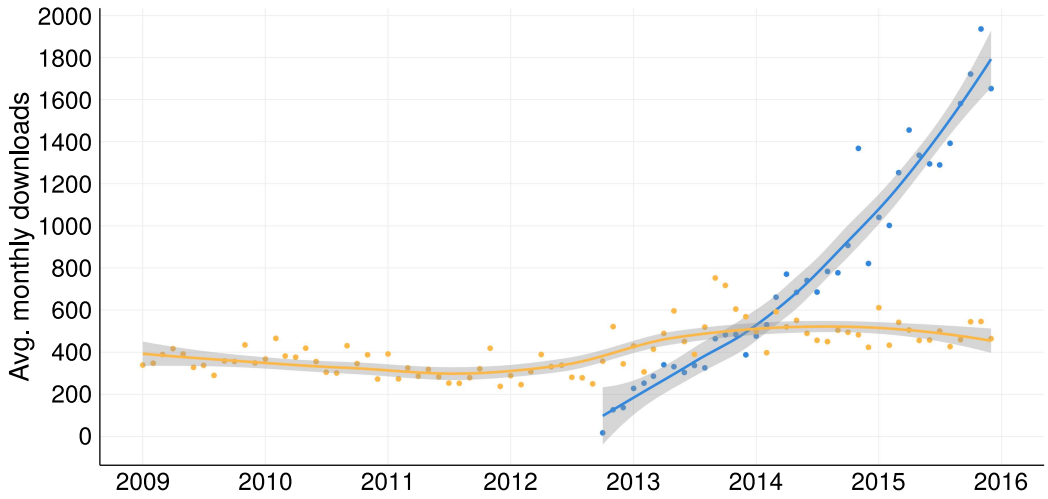

Figure 19: Increase of the download volume per package by repository (CRAN in blue, Bioconductor in yellow)

indication of productivity, and according to [24], R bugs are fixed usually within a few days.

Robustness describes an ecosystem's resilience in the face of external challenges, whether due to technological change or due to the emergence of competitors. Ecosystem *size* alone is a factor ensuring a level of robustness, but more critical is the stability and interconnectedness of the two networks that support the ecosystem: the software component network and the developer community [42]. Interconnectedness increases the contributors' commitment to the ecosystem, and increases the cost of leaving it in for alternatives.

Likewise, on the end-user end of the food chain, scale of use and satisfaction are key indicators of whether the customer base remains loyal [42]. As discussed above, the ecosystem is large and growing. Section 6.2.2 shows that, although the ecosystem is still rapidly growing in *width* and overall complexity, it does not appear to be growing in equivalent *depth*, as demonstrated by the *maximum depth of dependency* metrics.

While there is a significant proportion of packages with a large *maximum depth of dependency*—some 22% in CRAN and 16.9% in Bioconductor have a depth of five or greater—overall the depth of the dependency tree is relatively shallow, and most of the packages actually contributing major extensions in functionality are located in the lower levels: only three packages of those with more than 100 *RevDeps*—*data.table*, *ggplot2*, and *Hmisc*—are located

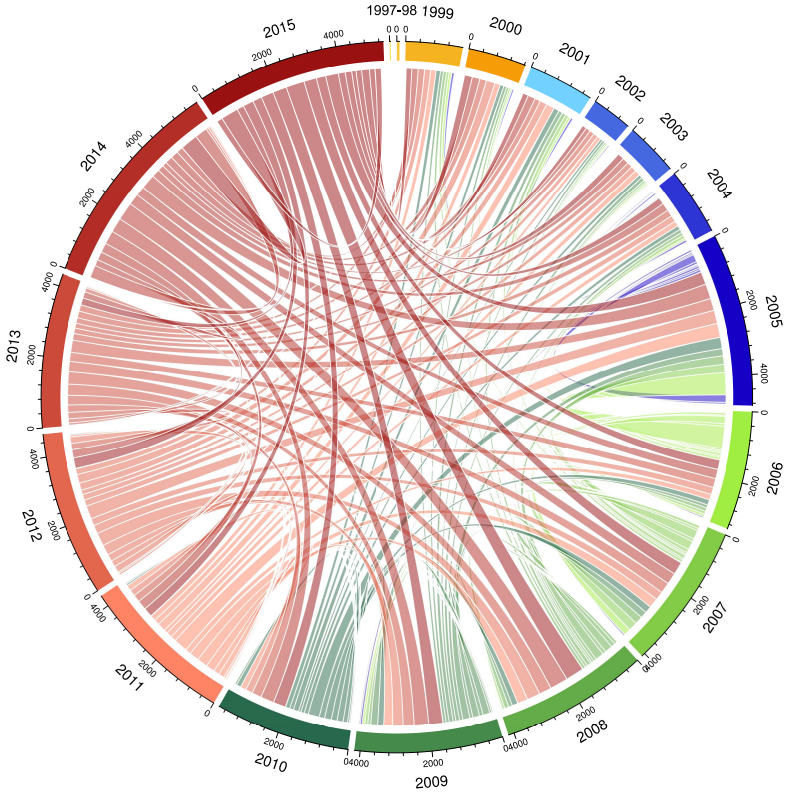

Figure 20: Chord diagram of the dependency links between packages and their dependencies by year in which they were published.

in levels 5 and 6. Published in 2006, 2007, and 2003, respectively, their location at this depth is not a result of any increase in network complexity in the intervening years, but was built in from the beginning. In addition to the shallow dependency network, it is clear that the core packages of the ecosystem, as can be seen in Section 6.3 are found among the older packages.

What this means is that the software marketplace of R has a *relatively static structure*. It relies upon a small group of fundamental packages, and while new such packages continue to appear over time, few large chains of reuse appear outside this subset. Coupled with the evidence on an increasing number of single-author and single-version packages, packages that satisfy a specific end-user need rather than serve to expand the ecosystem, and the increased presence of "lone wolves", it reinforces the view that the R marketplace has many leaves and minor branches (packages) but only a few

big branches (fundamental packages). On the one hand, this is an indication of continuity and limited obsolescence, which are seen as virtues for an ecosystem's robustness [30]. On the other hand, this behaviour chimes with and confirms R's nature as an end-user-oriented ecosystem. Despite its undoubted success, therefore, *the structure of the R software market is determined and limited by its own relatively narrow scope.*

One of the problems R faces is a result of its very success: its rapid expansion has overstretched the capabilities of its dependency versioning and handling system [37]. As cannot load multiple versions of the same package, backwards compatibility is often hard to maintain through updates [53]. As Claes et al. [24] discovered, most bugs are introduced to packages via changes in their dependencies. In view of the structure of the dependency network, this potentially implies a heavy workload for package maintainers when they depend on a package that is frequently updated; given the relatively limited technical base of the ecosystem however, and the evidence for the continued survival of single-version packages, it is still unclear to what extent such changes actually trigger the need for corrections across their reverse dependencies. A more detailed examination of this question, which would have to take into account the source code of the dependencies, is left for future work.

A further aspect related to robustness in the software network is the *deprecation rate of packages.* At least in CRAN, there appears a significant proportion of packages that are removed as no longer compatible with the most up-to-date R version, or due to other problems, but that rate is relatively stable, indicating that this is a "normal", or at least sustainable, rate for the ecosystem. It is also noteworthy that many of the deprecated packages, and deprecated or archived versions of packages, continue to be downloaded, albeit in much smaller numbers than their up-to-date analogues. It would be interesting to correlate download trends with the period where a package or version is registered as "active" as opposed to the period where a package or version is archived, but, at least for packages, we do not possess the data on when they were marked as "archived".

In the developer community network, a similar picture emerges. On the one hand, R developers are generally active and collaborate with each other across different age groups, while almost half of the community members ("authors and maintainers") have a significant role in the development and maintenance of R packages. Furthermore, at least as far as we could see with our limited data, the turnover rate of the ecosystem community is very low. On the other hand, Figure 13 shows that *the degree of the average developer's*

participation in the ecosystem is rapidly declining, at the same time as more people join it. As a result, here too the community is dominated by a few closely connected members, who not coincidentally are also members of the core groups maintaining CRAN and/or Bioconductor. The influx of new developers however increases the diversity of the developer pool, and is a strong sign of continuing, and indeed increasing, popularity. The same trend also emerges from the examination of overall download trends on the end-user side of the community.

Niche creation presents a more mixed picture. Measuring niche creation boils down to measuring diversity in an ecosystem and its self-renewal ability. As already discussed above, despite its growth, the R ecosystem's scope remains essentially the same. Within this restriction, however, there is evidence for increasing variety: new fundamental packages continue to emerge and support new extensions of the software marketplace, like *RJSONIO*, *dplyr*, or *raster*, as mentioned in Section 6.2.2. New subsidiary marketplaces have become available, like Bioconductor, GitHub, Omegahat, and additional resources such as the documentation aggregator RDocumentation [10]. It is notable that in terms of downloads, CRAN continues to be the main driving factor for R's popularity; special-purpose offshoots like Bioconductor are swept along in the overall growth of the ecosystem's use, but have limited potential. The situation may change, however, with the rise of GitHub as an alternative hosting and development environment [29].

In terms of technical diversity, the R ecosystem allows developers to port packages written in other language or languages into R. The list of the languages present in the software marketplace—we counted ninety—runs the gamut, from R, C/C++ and Fortran to JavaScript and Assembly. The R ecosystem has many tools for interfacing with databases or online data collections such as those offered by the Google ecosystem (e.g., GoogleMaps). While it is impossible to analyse the exact composition of the developer and end-user community, surveys show that R is mostly used by academia as well as by enterprise users [60], as well as by relative newcomers to data analysis, benefiting from its relative ease of use, its open-source nature, and its widespread use as a teaching tool in academia [21].

That said, the basic structure of the ecosystem has not changed, and the great age of the fundamental packages shows that whatever changes are occasioned as a result of changes in the platform and/or the wider technical environment tend to happen within these same packages, rather than through their substitution by newer rivals. This is mirrored in the community, where

the older members and the core groups, which these members are commonly part of, play a major role, precisely due the close association of these contributors with the small subset of packages sustaining the ecosystem. It is clear that the majority of the community focuses towards satisfying their own particular needs, rather than developing the ecosystem further. While this has not inhibited R so far, it may become a problem as general-purpose programming languages like Python, and newer rivals like Go, have come to offer an equally rich suite of analytical tools.

9. Discussion

9.1. Threats to Validity

Our work represents a high-level examination of the composition and history of the R ecosystem, with the aim of detecting trends and patterns among the ecosystem members, and examine, as far as possible, their origin and significance for the ecosystem.

9.1.1. Internal Validity

As with all data-driven analyses, the quality of our work is determined by the quality of our data. In so far as the packages were available online, we were able to retrieve the entire content of both repositories.

The dataset we obtained was rich, but had its limitations. Documentation was often missing or contained format and content errors, which we tried to address by several iterations of clean-up and correction phases, as described in Section 5. For instance, despite guidelines by the R Foundation, version numbering was extremely inconsistent across packages, which precluded us from using this information for analysis of package evolution. The large variety of author name formats and role documentation, even within the same document, meant that our automated parsing of author names in particular had to be coarse-grained. It is likely to have missed some names or counted some variants of the same name as different authors, and was unable to distinguish the scale and importance of each author's contribution to a package.

In addition, the package data we obtained was relatively coarse-grained. It would have been useful to examine the evolution of a package across all its changes, and not only those present at every uploaded version. In CRAN this presents the problem that we cannot judge why a specific version was uploaded to the repository and one or more intermediate ones (if they existed

at all) were not. However, for the purposes of our analysis of the ecosystem as it is offered to the end-users, this is not a major limitation. Similarly, Bioconductor's versioning system affected our examination of that repository, where many packages contained practically no changes between versions. In addition, it skewed the release date and version data for packages and community members alike, although our examination of community evolution by yearly cohorts mitigated its effect on the latter. This is a problem that could be solved, at least for a large part of the data set, by exploiting sources like GitHub. Fortunately we were able to remove one of the limitations present in our previous work, concerning the download data for Bioconductor, because more data with respect the package download history has been made available since by the Bioconductor project. This in turn allowed us also to greatly improve the data set regarding package release dates.

9.1.2. External Validity

We have not included in this study three important data sources: detailed data on source code structure or function calls, as well changes in the code and their propagation through the dependency network; detailed data on developer activity, e.g., commits, that could be gathered from sources like GitHub; and finally data from subsidiary sources like message boards, bug reports and fixes, etc. that would provide more info on community (including end-user) interaction with the R ecosystem, and more details for the evolution of individual packages. Such a study would require a different tool and data set than the one we employed here, and focus more on the technical aspects of the marketplace as a modular software product, or even on the characteristics of individual packages, rather than the high-level, evolutionary and behavioural analysis of the entire ecosystem that we undertook. Mining and exploiting these sources, and combining them with the present approach, will be the subject of future work.

9.1.3. Construct Validity

The analysis we performed was in the first instance exploratory, both in terms of modelling the R ecosystem and in testing out various metrics for the purpose. We were inspired by similar studies in the available literature, and based on our own experience we had some ideas of what we might find. Nevertheless we tried to let the data speak for themselves, and did not set out to prove or disprove any preconceptions on what our results would look like.

The wide scope of our data set is both an asset and a problem: on the one hand, it reduces problems with individual packages to statistical insignificance, but on the other, it may have obscured important trends that a more focused examination of specific ecosystem member categories might reveal.

As mentioned in Section 6.2.1, a further limitation arises from our consideration of packages and their dependencies from a quantitative rather than qualitative view. As a result, interactions determined by package content, and the nature of functionality shared via dependencies could not be measured. Our dependency network analysis therefore considers all packages idempotent, and measures their impact by the connections to them, rather than the content of these connections.

9.1.4. Conclusion Validity

The aforementioned problems in data quality affected many packages individually, but our sample size was large enough that this was not prohibitive, especially since our focus was on the interactions between the ecosystem members and the relative values of the metrics we formulated. The metrics we derived were mostly complete, homogeneous, and reliable.

The statistical methods we employed (see Section 5 for an overview) are fairly standard for this type of analysis, and we took care to use methods, such as *Cliff's* δ, that are recommended as robust. We furthermore did not rely on the results of any single test, but combined several different metrics and statistical methods to examine our dataset.

9.2. Insights

In terms of understanding the behaviour and forming a model of the R software ecosystem, the data we gathered and the resulting metrics were more than sufficient to gain an insight into its overall structure, dynamics, and evolution. Our data allowed us to determine and classify ecosystem actors and their interactions in both the marketplace and the community, i.e., analyse the "network level" [42] of the ecosystem. This is sufficient from the perspective of an ecosystem manager, or a company wishing to study an ecosystem before entering it; more work in a different direction would be required to assist, for instance, a developer seeking to optimally integrate a particular product in the ecosystem.

Our metrics allowed us to detect the presence of a few basic contributor categories: "lone wolves", "network builders", the "package owners", authors and/or maintainers, as opposed to the occasional, or "average" community

participant. Likewise we were able to identify the most important packages that provide the main extensions to the R core, and which form the pillars of the ecosystem's software marketplace. The close relationship between these packages and the core community members was not unexpected, in accordance with the so-called "Conway's Law" [27], but the scale of this identification was. Such results confirm Iansiti and Levien's [41] view on the vital role played by "keystone" members in the evolution of the ecosystem, rather than models of collaborative development without a dominant actor.

In general, our analysis in both the marketplace and the community shows a broad but relatively shallow network, dominated by a few "big" actors that supply fundamental extensions, and which once established retain their position. According to Bosch [19], this is a hallmark feature of end-user-oriented ecosystems, particularly in open source ones, since any financial reward is low to non-existent. A study of the Ruby ecosystem by Kabbedijk and Jansen [46] also found similar results, with the core of the ecosystem in both packages ("gems" in Ruby parlance) and contributors comprising about 10% of the total. One should not jump to the conclusion, however, that the structurally important packages are the only ones that matter in an ecosystem, and that those with few reverse dependencies, for instance, are unimportant or redundant. As Hornik pointed out [37], the particular needs and views of the end-user communities must be taken into account, and it is difficult to judge a package's "usefulness" or quality even within a community with such a relatively narrow scope as statistical software.

Of interest beyond the scope of the R ecosystem is also the evident shift in user roles and behaviour, in terms of collaboration and contribution volumes, once the R ecosystem hit a relatively "mature" phase, and once it became popular, for reasons likely external to itself. Most of these newcomers display a "selfish" behaviour, feeding off the ecosystem without much involvement on their part. Of course, that is what R was designed for, and it is probably also a common feature of software ecosystems: as the ecosystem matures, becomes established, and its variety of offerings increases, the rate of re-composition by developers to cater to their own specific needs increases even more quickly. This points to a possibly common characteristic of FOSS, end-user-oriented ecosystems: a small, highly-motivated and active core community, providing a small set of stable core offerings, and a large mass of very purpose-specific offerings built upon them, which experience a high rate of deprecation and renewal. It is also likely that the same tendencies will appear in closed, industrial ecosystems that form around a

keystone actor or community, for much the same reasons. Thus the keystone actor(s) will be active in maintaining and enlarging the marketplace since it represents a considerable investment, whereas peripheral actors will enter the ecosystem if they think they can profit from it (whether through monetarization or through satisfying their own needs). Once this is satisfied, few of them will seek to advance to keystone status in turn.

R also provided the opportunity to study the effects of two different marketplaces, one generic and a branched-off niche-oriented one, on the ecosystem. We found that, although Bioconductor has a clearly distinct identity in functionality, structure, and community, its members generally behave like those of the parent repository, with which Bioconductor retains significant links, both in technical dependency and in terms of overlapping communities. Thus Bioconductor is still mostly an extension of CRAN, rather than an independent marketplace. A comparison with other software ecosystems with multiple marketplaces would be interesting for arriving at a typology of marketplaces and how their interdependencies shape their evolution.

Overall, we consider that the method we have employed to analyse the R ecosystem is applicable to all ecosystems at the "network level". The exact composition of the marketplace and community and the particular trends detected for the R ecosystem, are, however, the result of several unquantifiable factors: R's nature as an end-user-oriented, language-based ecosystem; the initial design decisions made by the R team; the evolution of the data analytics community in recent years; or the lack, until relatively recently, of major FOSS alternatives like GO and Python's statistics tools; and, of course, the fact of our measurement of the R ecosystem at a particular phase of its development. As such, although we are able to analyze R's "health" with the frameworks suggested in literature, it is difficult to quantify the impact of external factors, or make predictions about its future. Furthermore, it remains to be seen which of our observations in R can be transferred to closed, commercial, or industrial ecosystems, whose the participants have other motivations, organization structure, and skill sets, and where the technical basis and requirements of the software components differ considerably from R's.

10. Future Work

In this study, we have compared the two main repositories for the R ecosystem, and the communities that have formed around them. We have

detected and measured the narrow dependency base of the R ecosystem, and demonstrated the strong dependence of Bioconductor on CRAN. R is an example where early comers have disproportionately shaped the ecosystem, and have been able to dominate it, as focal points both of the software marketplace, as well as of the developer community.

We have observed that newer members increasingly reuse extant offerings, but that despite an explosion in the size of the ecosystem, the "fundamental" functionality offerings increase more slowly. Instead, the same period of rapid growth has seen the increase of offerings that are target-specific and consume extant offerings, but do not in turn lend themselves to reuse. Furthermore, we have discovered that, on the one hand, R developers do co-operate across age cohorts, but that on the other, the collaboration rate among newer users is falling, and the group of "lone wolves" is increasing.

As pointed out in previous sections, the lack of sources such as GitHub, which apart from a new software development model also appears to complement, if not supplant, traditional repositories like CRAN, and which would allow a more comprehensive examination of author interaction, is a deficiency that we intend to rectify in future work. A further topic of interest could be more detailed examination of code content, changes, and interaction, as well as linking code metrics with the community.

In order to generalize our findings, a comparative analysis of ecosystems of different types, e.g., free and commercial, OS-based and application-based, web and mobile, or open and closed, is required, with the aim of discovering commonalities and differences in their structure and evolution. Work in this direction already exists in the literature, but we intend to apply our approach to a few carefully selected software ecosystems to complement it. In addition, as other ecosystems may offer more information in some areas than R, examination of specific aspects that proved difficult in R may be undertaken with better results there. We hope that our work will prove useful to other studies of ecosystems as a point of comparison, and will contribute to the emergence of a taxonomy of software ecosystems from the closed, tightly controlled, limited-membership, and commercially-oriented to the most open, free-licensed, and diverse.

At the other end of the scale, being able to identify the composition of ecosystems, the relationships between their members, and, ultimately, between the ecosystems themselves, may also allow us to extend our work in the direction of high-level application integration architectures and strategies in a software ecosystem environment.

References

[1] Bioconductor Package Guidelines. https://www.bioconductor.org/developers/package-guidelines/. Accessed: 2017-05-05.

[2] Bioconductor Version Numbering. https://www.bioconductor.org/developers/version-numbering/. Accessed: 2017-05-05.

[3] Bioconductor Package Submissions. https://www.bioconductor.org/developers/package-submission/. Accessed: 2017-05-05.

[4] CLOC - Count Lines of Code. https://github.com/AlDanial/cloc. Accessed: 2017-05-05.

[5] GEPHI. https://gephi.org/. Accessed: 2017-05-05.

[6] Omegahat. http://www.omegahat.net/. Accessed: 2017-05-05.

[7] OpenRefine. openrefine.org. Accessed: 2017-05-05.

[8] R-Forge. https://R-Forge.R-project.org/. Accessed: 2017-05-05.

[9] R Studio. https://www.rstudio.com/. Accessed: 2017-05-05.

[10] RDocumentation. https://www.rdocumentation.org/.

[11] Related Projects. https://www.r-project.org/other-projects.html. Accessed: 2017-05-05.

[12] Rmetrics. https://www.rmetrics.org/. Accessed: 2017-05-05.

[13] The 2015 Top Ten Programming Languages. http://spectrum.ieee.org/computing/software/the-2015-top-ten-programming-languages. Accessed: 2017-05-05.

[14] The Comprehensive R Archive Network. https://cran.r-project.org/. Accessed: 2017-05-05.

[15] Writing R Extensions. https://cran.r-project.org/doc/manuals/r-release/R-exts.html. Accessed: 2017-05-05.

[16] D. Alami, M. Rodríguez, and S. Jansen. Relating health to platform success: Exploring three e-commerce ecosystems. In *Proceedings of the 2015 European Conference on Software Architecture Workshops*, ECSAW '15, pages 43:1–43:6. ACM, 2015.

[17] O. Barbosa and C. Alves. A systematic mapping study on software ecosystems through a three-dimensional perspective. In M. A. C. Slinger Jansen and S. Brinkkemper, editors, *Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry*, pages 59–81. Edward Elgar Publishing, 2013.

[18] T. Berger, R.-H. Pfeiffer, R. Tartler, S. Dienst, K. Czarnecki, A. Wsowski, and S. She. Variability mechanisms in software ecosystems. *Information and Software Technology*, 56(11):1520–1535, 2014. Special issue on Software Ecosystems.

[19] J. Bosch. From software product lines to software ecosystems. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 111–119, Pittsburgh, PA, USA, 2009. Carnegie Mellon University.

[20] J. Bosch and P. Bosch-Sijtsema. From Integration to Composition: On the Impact of Software Product Lines, Global Development and Ecosystems. *Journal of Systems and Software*, 83(1):67–76, 2010.

[21] L. Burtch. SAS vs R vs Python: Which Tool Do Analytics Pros Prefer? `http://www.burtchworks.com/2016/07/13/sas-r-python-survey-2016-tool-analytics-pros-prefer/`. Accessed: 2017-05-05.

[22] P. R. J. Campbell and F. Ahmed. A three-dimensional view of software ecosystems. In *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, ECSA '10, pages 81–84, New York, NY, USA, 2010. ACM.

[23] M. Claes, T. Mens, and P. Grosjean. maintainer: A web-based dashboard for maintainers of cran packages. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 597–600, Sept 2014.

[24] M. Claes, T. Mens, and P. Grosjean. On the maintainability of CRAN packages. In *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*, pages 308–312. IEEE, 2014.

[25] M. Claes, T. Mens, N. Tabout, and P. Grosjean. An empirical study of identical function clones in CRAN. In *9th IEEE International Workshop on Software Clones, IWSC 2015, Montreal, QC, Canada, March 6, 2015*, pages 19–25. IEEE, 2015.

[26] N. Cliff. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological Bulletin*, 114:494–509, 1993.

[27] M. E. Conway. How do Committees Invent? *Datamation*, 14:28–31, 1968.

[28] A. Decan, T. Mens, M. Claes, and P. Grosjean. When GitHub Meets CRAN: An Analysis of Inter-Repository Package Dependency Problems. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 493–504, March 2016.

[29] A. Decan, T. Mens, M. Clas, and P. Grosjean. On the development and distribution of R packages: An empirical analysis of the R ecosystem. In *Proceedings of the 2015 European Conference on Software Architecture Workshops*, ECSAW '15, pages 41:1–41:6. ACM, 2015.

[30] E. den Hartigh, M. Tol, and M. Visscher. Measuring the health of a business ecosystem. pages 221–246, 2013.

[31] O. Franco-Bedoya, D. Ameller, D. Costal, and X. Franch. Queso: a quality model for open source software ecosystems. pages 209–221, 2014.

[32] S. Fortunato,M. Barthlemy. Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, 104(1):36–41, July 2007.

[33] R. C. Gentleman, V. J. Carey, D. M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry, K. Hornik, T. Hothorn, W. Huber, S. Iacus, R. Irizarry, F. Leisch, C. Li, M. Maechler, A. J. Rossini, G. Sawitzki, C. Smith, G. Smyth, L. Tierney, J. Y. Yang, and

J. Zhang. Bioconductor: open software development for computational biology and bioinformatics. *Genome Biology*, 5(10):R80, 2004.

[34] D. M. German, B. Adams, and A. E. Hassan. The evolution of the R software ecosystem. In *Proceedings of the 2013 17th European Conference on Software Maintenance and Reengineering*, CSMR '13, pages 243–252, Washington, DC, USA, 2013. IEEE Computer Society.

[35] G. K. Hanssen. A longitudinal case study of an emerging software ecosystem: Implications for practice and theory. *J. Syst. Softw.*, 85(7):1455–1466, July 2012.

[36] G. K. Hanssen and T. Dybå. Theoretical foundations of software ecosystems. In *Proceedings of the Fourth International Workshop on Software Ecosystems (IWSECO)*, pages 6–17, 2012.

[37] K. Hornik. Are there too many R packages? *Austrian Journal of Statistics*, 41:59–66, 2012.

[38] K. Hornik. R FAQ. `https://CRAN.R-project.org/doc/FAQ/R-FAQ.html`, 2015.

[39] R. Hoving, G. Slot, and S. Jansen. Python: Characteristics identification of a free open source software ecosystem. In *2013 7th IEEE International Conference on Digital Ecosystems and Technologies (DEST)*, pages 13–18, July 2013.

[40] M. Iansiti and R. Levien. Strategy as ecology. *Harvard Business Review*, 82(3):68–81, 2004.

[41] M. Iansiti and R. Levien. *The Keystone Advantage: What the New Dynamics of Business Ecosystems Mean for Strategy, Innovation, and Sustainability*. Harvard Business School Press, 2004.

[42] S. Jansen. Measuring the health of open source software ecosystems: Beyond the scope of project health. *Information and Software Technology*, 56(11):1508–1519, 2014. Special issue on Software Ecosystems.

[43] S. Jansen, S. Brinkkemper, and A. Finkelstein. Business Network Management as a Survival Strategy: A Tale of Two Software Ecosystems. In S. Jansen, M. A. Cusumano, and S. Brinkkemper, editors, *Software*

Ecosystems: Analyzing and Managing Business Networks in the Software Industry, pages 29–42. Edward Elgar Publishing, 2013.

[44] S. Jansen and M. A. Cusumano. Defining Software Ecosystems: A Survey of Software Platforms and Business Network Governance. In S. Jansen, M. A. Cusumano, and S. Brinkkemper, editors, *Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry*, pages 13–28. Edward Elgar Publishing, 2013.

[45] S. Jansen, A. Finkelstein, and S. Brinkkemper. A sense of community: A research agenda for software ecosystems. In *Software Engineering - Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*, pages 187–190, May 2009.

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

[47] B. Kitchenham, L. Madeyski, D. Budgen, J. Keung, P. Brereton, S. Charters, S. Gibbs, and A. Pohthong. Robust Statistical Methods for Empirical Software Engineering. *Empirical Software Engineering*, pages 1–52, 2016.

[48] K. Manikas and K. M. Hansen. Reviewing the Health of Software Ecosystems - A Conceptual Framework Proposal. In *Proceedings of the 5th International Workshop on Software Ecosystems, hosted by 4th International Conference on Software Business (ICSOB 2013)*, pages 33–44, 2013.

[49] K. Manikas and K. M. Hansen. Software Ecosystems - A Systematic Literature Review. *J. Syst. Softw.*, 86(5):1294–1306, May 2013.

[50] D. G. Messerschmitt and C. Szyperski. *Software Ecosystem: Understanding an Indispensable Technology and Industry*. MIT Press, 2003.

[51] J. F. Moore. Predators and prey: a new ecology of competition. *Harvard Business Review*, 71(3):75–86, 1993.

[52] M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences of the United States of America*, 103:8577–8582, 2006.

[53] J. Ooms. Possible Directions for Improving Dependency Versioning in R. *The R Journal*, 5:197–206, 2013.

[54] S. Brin, L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In *Proceedings of the seventh International Conference on the World Wide Web (WWW1998), April 14-18, 1998, Brisbane, Australia*, pages 107–117

[55] K. Plakidas, S. Stevanetic, D. Schall, T. M. Ionescu, and U. Zdun. How do Software Ecosystems Evolve? A Quantitative Assessment of the R Ecosystem. In *Proceedings of the 20th International Systems and Software Product Line Conference, September 16-23, 2016, Beijing, China*, pages 89–98. ACM, 2016.

[56] R. Spauwen and S. Jansen. Towards the roles and motives of open source software developers. In *Proceedings of the 5th International Workshop on Software Ecosystems, hosted by 4th International Conference on Software Business (ICSOB 2013)*, pages 69–80, 2013.

[57] M. Kendall. A New Measure of Rank Correlation. In *Biometrika*, 30(1/2):81–93, 1938.

[58] S. Syed and S. Jansen. On Clusters in Software Ecosystems. In *Proceedings of the 5th International Workshop on Software Ecosystems, hosted by 4th International Conference on Software Business (ICSOB 2013)*, pages 19–32, 2013.

[59] M. M. M. Syeed, K. M. Hansen, I. Hammouda, and K. Manikas. Socio-Technical Congruence in the Ruby Ecosystem. In *Proceedings of The International Symposium on Open Collaboration*, OpenSym '14, pages 2:1–2:9. ACM, 2014.

[60] The DataCamp Team. Choosing R or Python for data analysis? An infographic. `https://www.datacamp.com/community/tutorials/r-or-python-for-data-analysis`. Accessed: 2017-05-05.

[61] D. J. Watts, S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.