How do software ecosystems evolve? A quantitative assessment of the R ecosystem.

Konstantinos Plakidas
Software Architecture Research Group
University of Vienna, Austria
konstantinos.plakidas@univie.ac.at

Srdjan Stevanetic
Software Architecture Research Group
University of Vienna, Austria
srdjan.stevanetic@univie.ac.at

Daniel Schall
Siemens Corporate Technology
Vienna, Austria
daniel.schall@siemens.com

Tudor B. Ionescu
Siemens Corporate Technology
Vienna, Austria
tudor.ionescu@siemens.com

Uwe Zdun
Software Architecture Research Group
University of Vienna, Austria
uwe.zdun@univie.ac.at

ABSTRACT

In this work we advance the understanding of software ecosystems research by examining the structure and evolution of the R statistical computing open-source ecosystem. Our research attempts to shed light on the following intriguing question: what makes software ecosystems successful? The approach we follow is to perform a quantitative analysis of the R ecosystem. R is a well-established and popular ecosystem, whose community and marketplace are steadily growing. We assess and quantify the ecosystem throughout its history, and derive metrics on its core software components, the marketplace as well as its community. We use our insights to make observations that are applicable to ecosystems in general, validate existing theories from the literature, and propose a predictive model for the evolution of software packages. Our results show that the success of the ecosystem relies on a strong commitment by a small core of users who support a large and growing community.

CCS Concepts

•**Software and its engineering** → *Software architectures;*
•**Human-centered computing** → **Empirical studies in collaborative and social computing;**

Keywords

software ecosystems, R, empirical study, predictive model

1. INTRODUCTION

A software ecosystem is a cooperative model of software development and market featuring complex interaction dynamics between the participating actors. Modelling of software ecosystems is a widely recognized challenge in the field of software architecture.

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 [13]. The concept and practice of *software product lines* (SPLs) emerged as part of this trend. As a result of global alliances and the demand for rapid innovation cyles, companies employing SPLs increasingly expand their platform outside their organizational boundaries, thereby transitioning to a *software ecosystem* (SECO) [4, 9].

Popular examples for such ecosystems are Android, iOS, and Eclipse, to name a few. In addition to these company-driven ecosystems, many r open source ecosystems have emerged, such as R, Python, and Ruby. A number of definitions for the SECO concept have been proposed in the literature [4, 16, 17]. Despite their differing emphasis, they all consider a SECO as the sum of several elements: a common *software base* upon which a *network* or *community* of users with *shared interests* or *values* has built up a collection of derivative software products (a *software market*) in order to satisfy certain *needs* and accrue *benefits*, whether monetary or otherwise. The evolution of the SECO is driven by the motivation of individual users as well as by the social relationships and interaction between them [10, 13, 15]. As summed up by [9] — 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 the more traditional approach for software product development with segregated roles, a low level of collaboration, and closed processes.

Transitioning from an SPL company to a SECO company or building up a new SECO is indeed challenging for a number of reasons. Opening up a platform for a wider community demands for a careful design of the platform core elements versus the application specific extensions. A broader set of nonfunctional requirements (NFRs) needs to be considered including both development NFRs as well as operational NFRs. Important NFRs with regards to open SECOs are evolvability, composability, and maintainability, enabling the SECO and its developers to establish a healthy offering of user applications. Building up a new SECO is

SPLC '16, September 16-23, 2016, Beijing, China
© 2016 ACM. ISBN 978-1-4503-4050-2/16/09...$15.00
DOI: http://dx.doi.org/10.1145/2934466.2934488

often seen as a ‘chicken-egg’ problem — how to attract platform users if a limited set of applications is available for the platform and how to attract developers to implement applications if only a limited number of users are part of the SECO community? Thus, it is important to understand the general principles of how successful ecosystems evolve over time.

In this paper we analyze a popular SECO specimen in the form of the R ecosystem. We perform a quantitative analysis of the characteristics of the ecosystem as a whole and its members, especially regarding their evolution over time. The resulting metrics are then used to gain a picture of the ecosystem's current state and history, and to compare it with other similar studies in literature.

The remainder of this paper is structured as follows. In Section 2 we present a brief overview of the main characteristics of SECOs and the areas where research has focused in the past. Section 3 presents the R ecosystem and our approach and motivation in examining it. Section 4 presents the results of our survey of the ecosystem, the analysis we performed on it, and our attempt to construct a predictive model from the data gathered. Section 5 sums up our insights and compares R with similar studies in literature. Section 6 discusses the problems we encountered gathering our data and the limitations of our research, and Section 7 discusses the future direction of our work.

2. RELATED WORK

The field of SECOs 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 [15, 1]. Attempts have been made to construct a framework for a comprehensive SECO taxonomy [4, 13]. SECOs are generally distinguished by the nature of the *software base*, ranging from a complete software platform or set of platforms to services and software standards, or by the focus of their platform from complete operating systems to specific applications; the ecosystem's *accessibility*, depending on the existence of any barriers or vetting process before participating in the ecosystem; the ecosystem's *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* [13, 15].

Business and management issues are particularly important for any company deciding to move towards the SECO model [4], and any such move opens the challenge of ecosystem governance. The platform developer and maintainer acts as a central coordinator and exercises a degree of control over the course of the ecosystem, but it must also collaborate with and adapt itself to the ecosystem community. As the ecosystem's surplus value is generated by the community, the platform developer and the community acquire a shared responsibility for the ecosystem's development and must find ways to collaborate [9]. Thus some attention has been given to the roles played by the actors within a SECO and their interactions. The platform maintainer and/or a small group around it act as the *keystone* organization which drives the overall development of the SECO, and the SECO community is formed on the one hand of *end-users* and on the other of *third-party actors* who build their own products on the platform [9]. Another important governance-related issue concerns the assessment of *ecosystem health*, which as been studied at some depth as a theoretical modelling problem with several indicator metrics proposed [15, 12].

The R ecosystem has grown from a niche tool for statisticians in academia to a popular solution in the broader data mining community. The R ecosystem in its state of March 2011 has already been the subject of an empirical study, having been analyzed by [6]. This study examined the code characteristics and dependency relationships between the R platform and externally developed products and contained an analysis of the user community via the impact of the publication and evolution of externally developed packages on mailing lists. The study used a sample of 52 users to determine the popularity (or rather the frequency of use) of these packages, considering the more “popular” packages as a distinct category from the rest. Our approach goes into a similar direction, but examines the evolution of the ecosystem in greater breadth and over a far greater period of time. We also examine additional parameters such as downloads and individual authors, and include the “spin-off” *Bioconductor* repository in our study. This is important because we can follow the co-evolution of two different marketplaces with different orientations and practices, and their respective communities, on top of the same platform, as well as the interaction between them.

3. OBJECTIVES AND METHODS

3.1 Research Design

We aim to extract metrics that will allow us to determine the specific characteristics of an ecosystem and to compare it, based on approaches found in literature, with other ecosystems. This will assist in the formulation of a common concept model for SECOs, by determining general principles applicable to all ecosystems, clarifying the main areas of and reasons for differentiation between ecosystems, and assessing the impact of various factors intrinsic to ecosystems in general or specific to each particular ecosystem on the software products offered. Our aim is not only to improve our understanding of ecosystem behaviour, but also to determine a set of recommended practices both for the design of a new ecosystem from scratch as well as for application within an existing ecosystem, and provide tools and metrics that will allow the assessment of an ecosystem's state and health for governance purposes.

As discussed in Section 1, a SECO comprises three main components: the *software platform*, the *community of users*, and the *marketplace*(s) where additional software products are offered. As a result, the main Research Questions we posed and looked to answer was:

RQ1: *How does the R software ecosystem evolve?* We quantify and assess the R ecosystem, not only at a specific time, but across its 18-year lifespan. The quantification will allow comparisons with other ecosystems, while the time aspect allows observations on the lifecycle and health of the ecosystem as a whole as well as of its components.

RQ2: *How do the community members collaborate, and how does this impact the software marketplace?* Since a SECO is a business and social network, it is necessary to consider the activity of the community, with the aim of detecting behaviour patterns and modes of activity that can be said to have a direct impact on the ecosystem.

RQ3: *What makes a SECO marketplace product “success-*

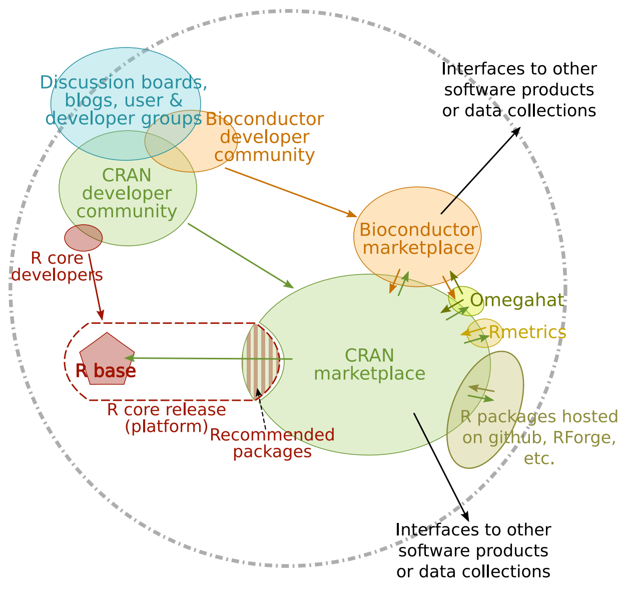
Discussion boards, blogs, user & developer groups
Bioconductor developer community
CRAN developer community
Interfaces to other software products or data collections
Bioconductor marketplace
R core developers
Omegahat
Rmetrics
R base
CRAN marketplace
R core release (platform)
Recommended packages
R packages hosted on github, RForge, etc.
Interfaces to other software products or data collections

Figure 1: Overview of the R ecosystem

ful"? Here we aim to define "success" of an externally developed product as part of a SECO, and determine metrics that can explain or even predict a package's success.

3.2 The R Ecosystem

As with similar empirical studies in the field, we chose a free, open-source ecosystem as it allows us to examine it in a depth unmatchable for closed or commercial, proprietary ecosystems [15]. 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 [11]. Its main application fields are statistical computing and graphics. In addition to the 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 [11]. The main repository for software products developed by the R community, known as packages, is the *Comprehensive R Archive Network* (CRAN[1]). In addition there is a number of related projects[2], chiefly the Bioconductor repository, which focuses on Bioinformatics packages, and RForge[3]. A large number of packages also exists independently on code repositories like GitHub. As many of the packages found on RForge and GitHub are also listed in CRAN or Bioconductor, we limited our study to the two latter.

R can be used either via the command line, or via GUIs and IDEs, of which RStudio[4] is one of the main representatives, developed under the supervision of Hadley Wickham, one of the main contributors to R.

3.3 Methods and Tools

To gather data, we built crawlers in Java that downloaded and parsed the entirety of the packages hosted on the CRAN and Bioconductor repositories, as well as the versions of the R core, and extracted information from the package documentation files and their code files. A problem was encountered in that the two marketplaces follow different versioning strategies. CRAN offers a repository where the package authors can upload new versions at will, and old versions are archived and still accessible. The only editorial oversight by CRAN is the removal of packages that are no longer maintained or have failed to incorporate changes to accommodate the latest R versions from the "current" package list, although they remain available in the archive. Bioconductor on the other hand bundles its marketplace in semi-annual releases, and hosts on its repository only the latest package version within each release. This impacted our research as we had access only to the last version committed before each Bioconductor version release, and no data on any intervening versions and changes (cf. Table 3, where the update periods for Bioconductor are practically identical to the semi-annual Bioconductor release cycle). To have a common ground, and as our focus is on software, we furthermore distinguished between those versions showing changes in the code from versions with changes in documentation files.

To determine package use ("popularity") within the ecosystem, we utilized the collected download data[5] from RStudio's CRAN mirror for the period from 01.10.2012 to 30.11.2015. For Bioconductor the download statistics provided on-site for the period from 01.01.2015 to 30.11.2015 were used. Again, while CRAN data distinguish between individual versions of each package, Bioconductor data refer to each package regardless of version.

The primary metrics extracted from our tools (cf. Table 1) concerned release date, identity and number of authors (including people whose code was re-used, in so far as they are included in the documentation) and of package maintainers, size of code in logical lines of code, dependencies to the R core or other packages, and the number of downloads. This data was then evaluated to create an image of the evolution of the individual packages, of the two marketplaces, and of the ecosystem as a whole, from 1997 until the end of 2015. CRAN[6] as well as Bioconductor[7] group their content by views by the functionality offered by each packages. Whereas the related documentation in Bioconductor tends to be complete, the reverse is true in CRAN, and views were not taken into account in this study.

In order to assess the quality of our data and confirm the validity of the metrics we selected for quantifying the R ecosystem, we created and compared prediction models for package popularity (expressed in downloads per day) based on the metrics we have gathered.

4. RESULTS

4.1 Platform Characteristics

The R core platform comprises the 14 base packages of R—the R base and compiler packages, as well as basic function, graphic, and programming support packages—and 15 "recommended" packages offering additional functions and datasets, as well as graphics support [11].

[1]https://cran.r-project.org/
[2]https://www.r-project.org/other-projects.html
[3]https://R-Forge.R-project.org/
[4]https://www.rstudio.com/
[5]http://cran-logs.rstudio.com/
[6]https://cran.r-project.org/web/views/
[7]https://bioconductor.org/packages/release/BiocViews.html

Table 1: Metrics directly measured or extrapolated from the data

Description	Unit	Range
Avg. number of downloads per day (*Popularity* or *Frequency of use*)	Download	0 – 2389
Number of authors	Author	0 – 170
Number of dependencies	Dependency	0 – 29
Number of reverse dependencies	Dependency	0– 690
Max. depth of dependency chain	Dependency	0 – 12
Number of versions	Version	1 – 182
Avg. time between two consecutive versions (*Update frequency*)	Day	0 – 1539
Number of versions w. changes to the source code	Version	1 – 174
Avg. time between consecutive versions with source code changes	Day	0 – 2064
Code size difference between current and original version	LLOC	-16160 – 185400
Current/Historical package code size	LLOC	0 – 185400
Time between first release and last releases (*Lifespan*)	Day	0 – 6437

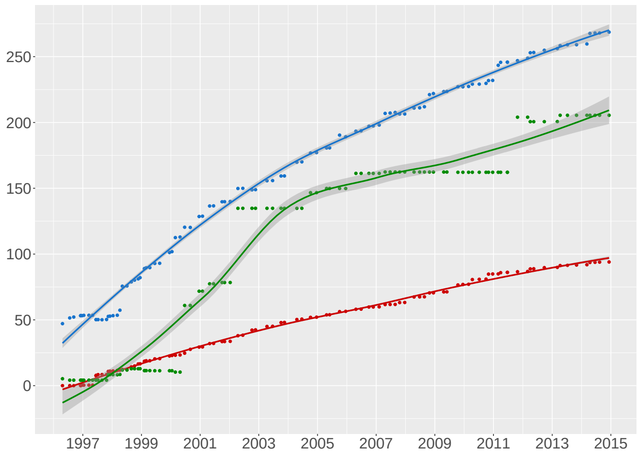

Figure 2: Evolution of the R core by code content (top: C, middle: Fortran, bottom: R)

We analyzed the composition of all R core versions available at CRAN, from `0.49` (23.04.1997) to `3.2.3` (10.12.2015) based on logical lines of code (lloc), i.e. excluding blank lines, comments, and logical arguments split across several lines. We observed the steady growth of the R core (excluding the "recommended packages") to the order of 1084% (from ca. 52,400 lloc to 568,137 lloc), and the expansion of the share of both R and especially Fortran in the code files of the R core. In the first versions, C code comprised almost 90% of the R core packages, with the remainder taken up by Fortran, and R almost non-existent. By version 1.0 of the R core, code written in R had expanded to 15.77% and Fortran-written code to 9.5%, in an overall expansion of the R core size of 228%. The proportion of R-written code has remained relatively stable despite the R core's growth by 473.7%, with the latest examined version at 16.54%. The major change has taken place in the Fortran code, which now comprises 36.17% of the R core packages. This is an expected result, since the R core naturally stresses performance, and C/C++ and Fortran are more suitable for such tasks [19]. The core team currently comprises 14 developers, with 32 having participated over the project's history[8,9].

[8]https://www.openhub.net/p/rproject/factoids
[9]https://www.r-project.org/contributors.html

4.2 Marketplace and Package Characteristics

4.2.1 *Marketplace Overview*

Both CRAN and the RForge repository reference individual packages and smaller repositories such as Omegahat, the R Extras repository[10], which contains several older packages that are now archived in CRAN, or the Rmetrics Computational Finance group[11]. These were judged as too small (ca. 100 packages in total) and niche-oriented to consider fully in our survey, although we did consider them in terms of package dependencies. Many Omegahat and Rmetrics packages are also hosted in CRAN, and these were included in our analysis as part of the latter repository. As of the date of our data collection, the CRAN marketplace hosted 8,943 individual packages and 54,490 versions (cf. Table 2). Out of these, 7,690 packages were displayed on the "current" list. 5,881 of the "current" packages had previous "archived" versions, while 1,809 were single-version packages, the oldest going back to March 2006. Of the 46,800 versions marked as "archived", 5,300 belong to 1,253 packages not on the "current" list, because they are no longer maintained or have failed to incorporate changes of the latest R versions.

The Bioconductor marketplace hosted 3,891 individual packages and 27,024 versions. 2,255 packages were included in the latest version (`3.2`, 14.10.2015). Of these, 1104 were function packages, 894 were annotation data packages (e.g. genome data on various organisms), and 257 were experiment data packages. Out of these "current" packages, 2,152 had previous "archived" versions, while 103 were single-version packages, the oldest dating to April 2014. 1,636 packages with 4,197 versions were only "archived" with no "current" versions, of which about 86% were annotation packages, and only 14% function packages (most of them having their last release with Bioconductor version `2.9`, November 2011 or earlier). While the high deprecation rate of annotation data packages (1556 or 63% of all such packages published in the repository) is expected as the old annotation packages are frequently discarded when they become obsolete, for the function packages this affects only 71 or 9% of the total, somewhat less than the approximately 14% of equivalent packages in CRAN.

[10]https://www.stats.ox.ac.uk/pub/RWin/
[11]https://www.rmetrics.org/

Table 2: Overview of marketplaces

	CRAN	**Bioconductor**
Packages total	8941	3891
of which in "current" release	7690 (86%)	2255 (58%)
Versions total	54490	27024
of which include code changes	48950 (90%)	12124 (45%)
Packages written only in R / C / Fortran	6324 (70.7%) / 68 (0.7%) / 62 (0.7%)	3525 (90%) / 115 (3%) / 115 (3%)
Packages with dependencies	4825 (54%)	2571 (66%)
Packages used as dependencies	2043 (22.8%)	505 (13%)

Table 3: Package statistics (for Bioconductor, only function packages were considered)

	CRAN	**Bioconductor**
"Average" (median values) package in each repository		
lloc/package (R / C / Fortran)	474 / 530 / 945	361 / 591 / -
number of dependencies	0	1
depth of dependency chain to the R core	2	3
number of subsequent versions	4	8
number of subsequent versions w. code changes	4	4
package age	2012-10-29	2012-03-19
release time between versions in days	129	159
"Average" package in the top 5% most downloaded packages in each repository		
lloc/package (R / C / Fortran)	1128 / 1094 / 1403	1878 / 1930 / -
number of dependencies	1	3
depth of dependency chain to the R core	1	3
number of subsequent versions	17	17
number of versions w. code changes	15	11
package age	2008-02-07	2007-12-21
release time between versions in days	118	168
"Average" package among those contributed by the most active 1% of contributors in each repository		
lloc/package (R / C / Fortran)	583 / 719 / 1019	1040 / 1023 / -
number of dependencies	1	2
depth of dependency chain to the R core	2	4
number of subsequent versions	7	15
number of versions w. code changes	6	7
package age	2010-06-23	2008-05-10
release time between versions in days	150	170

4.2.2 Code Characteristics

Our tools parsed the online package documentations and downloaded and examined the documentation and code files of the packages of both repositories. The data gathered was then analyzed to extract metrics on the code characteristics of the packages (code size and composition, evolution over time), their versioning (frequency of updates and type of changes), package dependencies, and the package authors. We then used these metrics to acquire an overview of the characteristics of the "average" package in each repository. In addition, we examined separately the characteristics of two special groups: the top 5% of the "popular" packages, as determined by number of downloads, and the packages contributed by the top 1% of the most involved community members (cf. Table 3), under the assumption that popular and well-established packages, as well as packages written by frequent and involved contributors, would display better qualities than the "average" package.

In contrast to the R core, the two marketplaces are dominated by packages written in R, with C playing a secondary but important role in both CRAN (25.5% of current releases, and 44.7% of the "popular" releases, include C content) and Bioconductor (28.7% of all function packages include C content, representing 96% of C use in the repository). A very small percentage. Fortran has a small presence (4.4%) in CRAN and a very marginal one in Bioconductor, and is predominantly used by older packages.

The tendency for R code content per package in CRAN overall displays a peak around the year 2006, and has been slightly, but steadily declining since. This is in contrast to Bioconductor, where it has slightly but steadily increased since 2004. The same slight but steady increase in R code content per package is observed in the popular packages in CRAN and the popular function packages in Bioconductor. Average C code content per package in CRAN also peaked around 2006, and has been very slightly declining since, whereas in Bioconductor it has been virtually steady. Again the picture is different for the most popular packages, where both CRAN and Bioconductor function packages display a small but steady growth. The picture for Fortran is similar, with overall use in CRAN declining, but a slightly increasing use in the most popular packages; in Bioconductor its use is too limited to be of value for comparison. The most popular packages are also larger on average than "ordinary" ones; the first, second, and third quartiles (Q1, Q2, Q3) of "popular" CRAN packages are 391, 1128, and 3443 R lloc/package, while for the repository overall they measure 190, 474, and 1132. The same holds true for C content and

Fortran content, and for the packages contributed by the most prolific authors, although here the difference is not so marked. The same findings are replicated in the Bioconductor repository for function packages (cf. Table 3).

A notable feature across both marketplaces is a proportion of about 20%-25% of packages where the current versions have less code content than their maximum code size; this could be attributed to the overall decline in package code content over time parallel to the increase in the number of packages, but is also present, and in a markedly higher proportion, among the "popular" packages, ranging from 34% for CRAN R packages to 49% for CRAN Fortran packages. Given that the "popular" packages are mostly older (slightly more than 50% in both marketplaces were first published before 2008) and therefore mostly unaffected by the large number of recent packages, this feature can be explained either as code refactoring or as the removal of features offered by the R core or by other packages; the exact reason is left to be determined in our future work. Somewhat counter-intuitively, in both repositories we observed an almost linear increase of R code content per package in relation to the number of dependencies per package.

4.2.3 *Dependency Structure and Density*

Dependency structure is of particular importance, since the increased interdependency of various software elements evident in SPLs and SECOs increases complexity and development overhead, and reduces the efficiency and the composability of the resulting software [5]. Only 54% of CRAN packages have a dependency to other packages of the repository, and only 34.8% have more than one. The figures remain approximately the same among the most popular CRAN packages (50% and 31.3%), and for the packages of the top authors (54% and 36.6%). In Bioconductor, the overall numbers are 66% and 32% for one and more than one dependency overall, but reach 79.2% and 59.2% for function packages respectively. This rises to 83% and 63.4% for "popular" packages and 80.4% and 68% for "popular" function packages. Similarly, for the packages contributed by the most active authors, 61% overall have dependencies and only 20.1% have more than one dependency, while the proportion is respectively 92.5% and 71.6% for function packages. CRAN packages depend on 1,738 individual packages, of which 93.1%, representing 97.8% of all dependency references, are from within CRAN itself, 5.4% are from Bioconductor, and the rest from Omegahat etc. 46.9% of all these dependencies occur only once and only 21% more than five times. In Bioconductor, there are 383 individual packages referenced as dependencies, of which 56.8% are from within Bioconductor, representing 77.4% of all dependency references, and 42.7% from CRAN. 53.4% are used once, and only 15.4% are used more than five times.

As is evident from Table 3, in comparison to CRAN, Bioconductor have a consistently higher number of dependencies to other packages as well as finding themselves more often at the end of a longer dependency chain of packages between themselves and the R core. Furthermore, 16 of the top 20 Bioconductor dependencies, which represent 54.7% of all dependency references in the repository, are from within Bioconductor, including the Biobase and BiocGenerics function packages that are the equivalent to the R core's base packages for the Bioconductor releases, biology-specific tools, and database interface packages like AnnotationDbi.

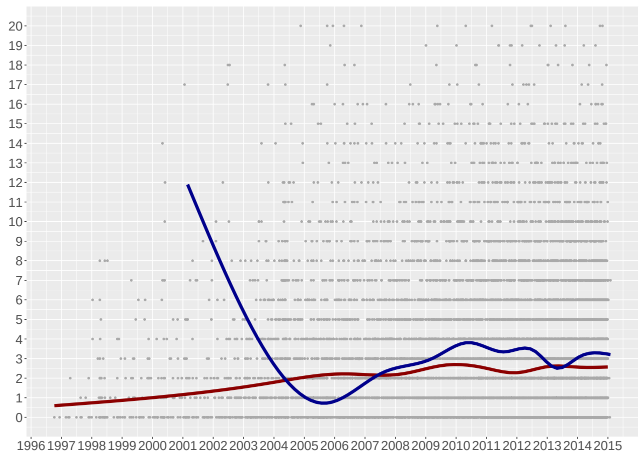

Figure 3: Evolution of number of dependencies per package in CRAN (red) and Bioconductor (blue) compared

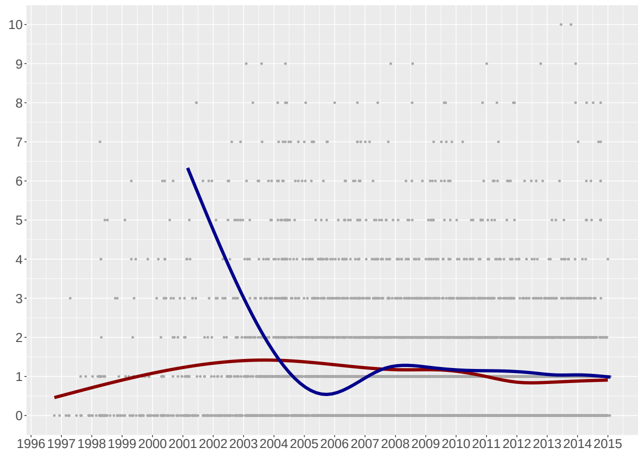

Figure 4: Evolution of max. depth of dependency per package in CRAN (red) and Bioconductor (blue) compared

The technical dependency of the Bioconductor repository from CRAN is thus limited to 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 sparsely (slightly more than 50% of CRAN dependencies in Bioconductor are only used once and 83.3% less than five times). This is also indicated through the maximum depth of the dependency chain of each package, i.e. the maximum distance between it and the platform through intervening dependencies. In CRAN, this is relatively low on average, but clearly higher on Bioconductor, as many packages of the latter depend on CRAN packages.

Of further interest is the evolution of the (average) number of dependencies and the depth of dependency as depicted in Figures 3 and 4 which shows that on the one hand the average number of dependencies increases over time in both repositories, albeit very slowly, and on the other that the max. depth of dependency is stable and even slightly declining, indicating once again that the "food chains" in R are relatively short, and that the the dependency structure is shallow rather than deep.

4.2.4 Highly Accessed Packages

In order to determine the driving factors of the CRAN repository, we focused particularly on the 50 most downloaded packages in CRAN. We found that they concern well-established (33 of them are older than 2012) packages that offer a major extension in R capabilities; e.g. the most popular package, Rcpp, offers integration of C++. Graphics packages (8), web technology integration packages (8), and of course mathematical packages also have a significant presence. 39 of them have no other dependency other than the R core, and the longest chain of dependency between a package and the core comprises 3 links. Of interest is also the fact that this subset includes both packages which strongly support the ecosystem as dependencies to other packages (e.g. the graphics package ggplot2 with 271 dependencies, or Rcpp with 235), as well as others that are not reused at all (7) or have a single dependency (7).

We also examined the top 50 Bioconductor packages. 39 packages were older than 2012, the most popular among them being Bioconductor infrastructure packages and the most common annotation/database interfaces. 11 of them had no dependencies, but 10 had five and more, and 17 had a chain of dependency greater than 3, with the longest being 6. Again in contrast to CRAN, only three packages had zero or one reverse dependency, and in general are far more interconnected with the rest of the marketplace, including all but four of the 20 Bioconductor top packages in terms of reverse dependencies.

4.3 Community Characteristics

Using the same tools we used to extract information on the packages, we also parsed the documentation files to extract information on the identity of the authors who contributed in them. Both CRAN and Bioconductor boast an exponentially growing community of authors. The CRAN community comprises ca. 11,000 people or entities, while the Bioconductor community comprises ca. 1860. About 400 authors participate in both repositories. As the author identities are real-life identities, we examined their behaviour and tried to gain insights on any structures, categories, or classification groups forming within the community.

In CRAN, the overwhelming majority of all authors have contributed to one (71.2%) or two (14.8%) packages. The majority of packages likewise has one (46.7%), two (24.7%), or up to three (13.6%) authors. Likewise, in Bioconductor, 66.3% of authors have contributed to one package and a further 15.4% to two packages, and most packages have one (56.4%), two (18.9%), or three (11.3%) authors. The most notable difference in terms of simple numbers is the degree where the most active authors contribute to the ecosystem. In CRAN, the top 1% of most active authors contributed in 1634 packages or 18.3% of the total, whereas on Bioconductor they contributed in no less than 2342 packages or 60% of the repository total. This is partly explained by the fact that most of the Bioconductor packages are annotation packages which as described experience a very high deprecation rate, but also because a few authors disproportionately dominate the repository: the top 10 authors account for 43.5% of all author references in the repository, in contrast to ca. 2.5% for their counterparts in CRAN.

In CRAN, as the main marketplace for R, we observe that most of the top active authors are the same as the members of the R Foundation. Our examination also shows that the same members have contributed to 30 of the 50 most downloaded packages in CRAN, 11 of which were written by a single author. Given the fundamental nature of these packages for expanding the ecosystem's functionality, this indicates a high degree of involvement of the platform development team, both in quantity and quality, in the marketplace built around the platform. A similar picture emerges also in Bioconductor, where the "Bioconductor Core Team"[12] is the second-most frequent author, and three of its members occupy places 1, 3, and 4 as individuals. A group of 378 people was detected participating in both repositories, but it is notable that the most active among them focus their activity disproportionately on one of the two repositories.

We also examined the number of authors in each package and collaboration between the community members. In CRAN, we counted 104,426 pairs of collaborating authors from 9,431 individual authors (85.7% of the total) in 4,612 packages (51.6% of the total). It must be noted however that one single package (`spatstat`) with 170 authors accounts for 28,056 pairs, and five more for further 21,424. Removing the top 1% as outliers, the numbers are reduced to 41,996 pairs involving 8,800 authors across 4,566 packages. In Bioconductor, we counted 17,248 pairs between 1,471 authors (87.4% of the total) in 1,692 packages (43.5% of the total). After likewise removing the top 1%, 14,906 collaboration pairs between 1,429 authors in 1,675 packages remain. It is certainly of importance that 99% of the 1092 authors of the most popular CRAN packages and 98% of the 191 authors of the most popular Bioconductor packages were engaged in collaboration with other authors, or that (disregarding outliers) the mean number of authors for popular packages was 3.26 in CRAN and 3.4 in Bioconductor, compared to 2.15 and 2 across the respective repositories overall. A very small proportion (0.98% in CRAN and 0.3% in Bioconductor) of packages contained no authors.

4.4 Prediction Models

One of the concerns driving our analysis of the R ecosystem was whether the behaviour of individual packages could be reliably modelled and predicted. Any such prediction would naturally rely on and reveal underlying structures among the data we have gathered. As the only relatively "objective" common metric that allowed packages to be directly compared with each other, we chose "popularity" in the form of downloads per day. In this section, we aim to identify the effect strength of individual package characteristics (i.e. the collected package metrics shown in Table 1) as independent variables on the "popularity" of packages as a dependent variable. We limited our examination to packages published until the end of 2013 in order to only examine only well-established packages. In Bioconductor we only included function packages in our analysis as annotation and experiment data packages display qualities (in terms of code content and lifecycle) that rendered them unsuitable for a direct comparison with most of the CRAN packages.

First we carried out a collinearity analysis according to [2] to exclude from the metrics we collected (cf. Table 1) those variables that were highly correlated with other variables Thus in CRAN the number of versions featuring code changes was found to be highly correlated with the number of overall versions and was excluded; in Bioconductor the lifespan variable was removed as it is highly correlated with the num-

[12]https://www.bioconductor.org/about/core-team/

ber of overall versions. This is no surprise, since as described in 4.2.2, CRAN packages are usually updated when there is a code change, whereas in Bioconductor due to the way we gathered our data the version numbers are in direct proportion to the package's age. We then performed a multivariate regression analysis using the *Eureqa* tool[13], which identifies the simplest mathematical formulas that can describe the underlying mechanisms in a specific data set. A goodness of fit measure was calculated based on the absolute deviation of the median (assuming X_i is the prediction and Y_i is the actual value):

$$A(accuracy) = \frac{\sum_i |Y_i - X_i|}{\sum_i |Y_i - median(Y_i)|}$$

The smaller A, the better the prediction. Based on the work of [14], the value (1-A) represents the proportion of the variation in the Y variable explained by the predictions. (1-A) values in the range 0 to 0.0372 represent a *small* effect size, values in the range 0.0372 to 0.208 represent a *medium* effect size, and values in the range 0.208 to 0.753 represent a *large* effect size. Furthermore, for good prediction models the residuals have to be normally distributed, which indeed is the case with our data. The influential points are the data points whose removal will cause a large change in the fit. They can be detected using Cook's distance contour lines [7]. When some points have a Cook's distance that is larger than 1, it suggests that the model is poor or might have outliers. In this case, our models do not have influential points. We have additionally validated our models using the cross-validation analysis to overcome the over-fitting problem [8]. Eureqa internally applies the cross-validation by diving data into a training and a validation set, but we applied a slightly more rigorous 5-fold cross-validation technique ([18]) that corroborated the results obtained by Eureqa.

Table 4: CRAN prediction models

Prediction Models	**1-A**
$532.18 + 9.29 * \sqrt[2]{nVer} + 2.48 * e^{-23} * aVer^3 * nVer^4$	0.12
$657.27 + 0.0002 * nVer^2$	0.08

Table 5: Bioconductor prediction models

Prediction Models	**1-A**
$max(2.02*e^3+9.37*e^{-7}*nRevDep*dDep*aCVer, 5.15*e^{-12}*nAut^3*nCVer^2)-4.02*dDep-4.96*e^{-13}*nDep*nVer*nCVer*nRevDep^2$	0.42
$max(nAut + 8.1 * mod(nRevDep, 1.13 * e^4), 2.7 * e^{-15} * nAut^4 * nCVer^2)$	0.35
$1.51 * nRevDep + 5.76 * mod(1.51 * nRevDep, 9.99 * e^3)$	0.31

The two best models we obtained for CRAN (Table 4) were found to be dependent solely on the number of package versions (*nVer*) and the update frequency (*aVer*), with a *medium* effect size. We also ran the analysis with the *nVer* and *aVer* metrics excluded from the set, in which case the best model obtained had a *small* effect size of 0.03. In Bioconductor, we obtained three models (Table 5). From these, and particularly from the third model, which depends only on it, it is evident that the number of dependencies on a package (*nRevDep*) metric is the most relevant here. We also examined subsets with only the "current" packages in both repositories, or with packages that have more than one version; in both cases, we obtained very similar prediction models in terms of accuracy and included metrics as those already presented above. In addition, we also analysed the subset of all packages published after 2013 for both repositories. We found that the results obtained did not change for packages published in early 2014, but that the prediction quality becomes increasingly worse after that.

From the above results we see that the metrics found to be relevant for CRAN do not appear to be so for the Bioconductor packages and vice versa. We can only speculate why the two repositories display such different behaviour. One plausible reason for why versioning is not relevant to Bioconductor may be the fact that we do not have access to the full number of versions per package, which means that we cannot reliably use the number of versions or the update frequency to compare packages with each other, as for most packages, these numbers coincide with the Bioconductor releases. The strong connection between the popularity and the number of reverse dependencies among Bioconductor packages, is also likely to be found in the more homogeneous and structured nature of the Bioconductor repository, which is characterized by a very large number of packages contributed by a small group of authors, and by a large degree of interdependency between the most important packages and the rest of the repository. CRAN displays a wide variety in the characteristics of its member packages, so that intrinsic characteristics like code size or number of (reverse) dependencies are unable to reliably predict the popularity. A prediction model for the behaviour of a random package in such a diverse marketplace as CRAN therefore appears to be unfeasible. Nevertheless, it is also clear that versioning can serve an indicator of whether a package is actively maintained, which is most often the case with relatively "successful" and long-living packages.

5. DISCUSSION

Our examination leads to a number of insights, which are partly confirmed by similar empirical studies in open-source ecosystems.

Regarding **RQ1** we observed that the R ecosystem continues to grow strongly, in both size and variety. In CRAN for instance, in 2013 there were 1,113 new packages representing in their "current" versions 1,545,387 lloc (or 10% of the total size of the repository at the time of measurement), 1379 packages with 2,322,550 lloc (15.9%) in 2014, and 1,660 with 2,010,562 lloc (13.8%) in 2015. Thus 40% of the marketplace's *new* content was created in the last three years alone. A similar development is discernible in Bioconductor as well, although not quite as pronounced due to the high turnover of packages. At the same time, over half of CRAN authors have joined since 2013, and over half of Bioconductor authors since 2012. Based on the ecosystem health metrics defined in [12], the R ecosystem displays a high degree of *productivity.* It is also a good example of *niche creation* ([12]), as it offers a large variety of solutions, connections to many other software products and ecosystems, and the ability to establish niche-oriented subsidiary marketplaces, of which Bioconductor is a very good example. In terms of "robustness" ([12]), we observe that although continually growing, the ecosystem has a relatively stable and old foundation of essential components (e.g. the 50 most used CRAN packages) which support and extend the actual `R core`. We note that although the latter has also continu-

[13]http://ccsl.mae.cornell.edu/eureqa

ally grown in size, it has remained relatively stable in terms of content since the refactoring of its base package in version `2.0`, and has not incorporated any marketplace packages into the limited set of *recommended* packages [11]. It is around this stable foundation that a community has emerged. In CRAN in particular, the resulting marketplace is relatively "shallow", with most packages depending directly on the `R core` or at most one other package; this is an indication of the heterogeneous nature of the CRAN marketplace, with a variety of authors and functionalities on offer. Bioconductor offers a far more homogeneous picture, with strong interdependencies between its packages. Its versioning system and the high degree in which it is dominated by comparatively few authors make it also closer to a separate and complete "spin-off" product rather than simply an aggregation of its component parts, as is the case with CRAN.

Comparing our results with a similar comparative study of five major open-source SECOs ([3]), several interesting observations arise. Of the ecosystems under discussion (eCos, Linux, Debian, Eclipse, and Android), the R ecosystem is most similar to Android, in terms of organization—both representing a tightly controlled platform with a very free and open marketplace—as well as in terms of platform-to-marketplace ratio: as of the date of our measurement, the R core comprised 568,137 lloc, which represents about 3,25% of the aggregated lloc of the "current" CRAN and Bioconductor packages. It is also far closer to Android in its dependency structure, with a lower percentage of packages with dependencies, and a much lower percentage of reverse dependencies, than any of the five ecosystems mentioned above. In terms of package characteristics, our findings generally replicate these of the earlier study ([6]). Our data reconfirms the dominance of R code in the community-contributed packages, the greater average size of "popular" over "average" packages, and the low number of dependencies

Concerning **RQ2**, our major insight is the crucial role played by the keystone teams in supporting the marketplace by providing packages that extend the functionality of the R core, as it confirms that active and early involvement by "insiders" is beneficial to the growth of the ecosystem while allowing to keep the platform itself to a relative minimum. Beyond this small core of dedicated editors, the great majority of the community consists of single-package participants. Of interest is also the community's division between two marketplaces. While there is a considerable number of users contributing to both marketplaces, there is a clear tendency for the two marketplaces to be maintained/driven by separate communities, or at least for authors to concentrate their attention more on the one than the other.

In determining the criteria for a "successful" package, in answer to **RQ3**, we settled on three characteristics that, in combination, might provide suitable indicators: popularity (indicated by the number of downloads), reuse (indicated by the number of dependencies from the package and/or the existence of spin-offs), and maintenance (indicated by a regular update schedule and of course the package's survival). As in the previous study of the R ecosystem ([6]), a strong correlation between package popularity and its number of versions is also evident (and strengthened further through the results of our prediction model presented in Section 4.4), while average package size appears to be relatively stable. As described above, we have found a strong relationship between maintenance and popularity.

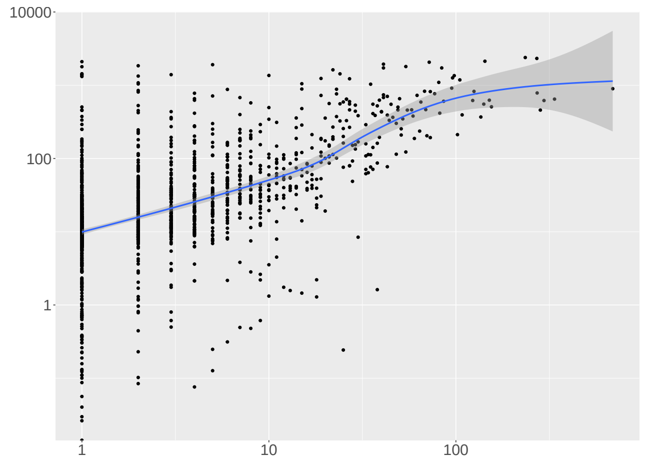

Figure 5: Downloads/day per number of dependencies upon a package in CRAN

Figure 5 also indicates a link between popularity and the number of dependencies on a package, which is expected as these packages are necessarily downloaded whenever another package dependent on them is installed. Although there are outliers in both directions, "popular" packages also tend to be significantly larger, and presumably feature-rich, in comparison to "average" packages. No definite assertions regarding **RQ3** can be made, but there are clear indications that well-established, frequently updated packages written by experienced and involved authors tend to dominate the field.

6. THREATS TO VALIDITY

The main threat to the validity of our conclusions derives from the limited nature of our data. Documentation was often missing or contained format and content errors, which we tried to address by spot-checking our data set and applying corrections for the errors and variations thus discovered. The automated parsing of author names in particular 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. For this, a more extensive study incorporating e.g. commit data from GitHub would be required.

The use of downloads as a "popularity" metric is also less than ideal, as many packages are downloaded as dependencies to other packages. However, as demonstrated, R packages have few dependencies and few of them are used as dependencies to other packages, meaning that this concern does not apply to a very large number of packages. In addition, most of the dependencies are to the "essential" packages that offer some major extension of functionality, meaning that these packages are likely to have been downloaded already with the very first additional packages installed by any R user.

For dependency analysis, we considered only the (non-core) packages explicitly mentioned in the "Depends" and "Imports" fields of the documentation and needed for the package to work, and not "soft" dependencies like "Suggests", which include a packages that might be required for additional features. In addition, we did not examine the code content in depth by implementing a parser to determine functions, but only recorded changes in code size.

The main limitation on our work came from the narrow origin of the data obtainable for Bioconductor, where our version data is incomplete due to the repository's versioning system. A more complete picture could have been obtained by consulting the package source in repositories like Github or the package homepage, but as the great majority of packages did not contain such information we decided to leave this for a more detailed study of the packages in the future. In addition, Bioconductor download data were limited to the previous 12 months, and did not distinguish between versions.

7. CONCLUSIONS AND FUTURE WORK

In this work, we presented a detailed analysis of the R ecosystem and analysed key success factors for success. The main conclusion is the crucial role of the *keystone*, even in so open and decentralized an ecosystem as R, in facilitating the expansion of the ecosystem and laying much of the groundwork for the marketplace and community to grow on. We also have seen that despite a huge increase of the ecosystem in size over time, the basic characteristics of the ecosystem components (dependency connectivity, popularity, etc.) tend to remain stable, and that the degree of connectivity between ecosystem members is proportional to (and indicative of) the homogeneity of the ecosystem. Even though our results derive from the study of a free, open-source ecosystem, we consider these insights generally applicable even for more closed, company-driven ecosystems.

Based on our presented work, we will extend our research to cover other similar free, open-source ecosystems (for example, Ruby or Python) as well as commercial ecosystems and closed industrial ecosystems. In this way we will be able to compare our findings across a variety of paradigms and determine which factors impact the evolution, structure and NFRs of an ecosystem as a whole and of its software components in detail. These insights will be helpful in testing the various SECO models proposed at times in literature, and in gathering a variety of strategies and recommended practices, based on real-life practice and experience, to be applied in ecosystem governance.

8. REFERENCES

[1] 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.

[2] D. A. Belsley, E. Kuh, and R. E. Welsch. *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity (Wiley Series in Probability and Statistics)*. Wiley-Interscience, 2011.

[3] T. Berger and S. Nadi. Variability models in large-scale systems: A study and a reverse-engineering technique. In *Software Engineering & Management 2015, Multikonferenz der GI-Fachbereiche Softwaretechnik (SWT) und Wirtschaftsinformatik (WI), FA WI-MAW, 17. März - 20. März 2015, Dresden, Germany*, pages 80–81, 2015.

[4] 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.

[5] 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.

[6] B. A. Daniel M. German 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.

[7] J. J. Faraway. *Practical Regression and Anova using R*. July.

[8] A. Field, J. Miles, and Z. Field. *Discovering Statistics Using R*. SAGE Publications, 2012.

[9] 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.

[10] G. K. Hanssen and T. Dybå. Theoretical foundations of software ecosystems. *Proceedings of IWSECO*, pages 6–17, 2012.

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

[12] 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.

[13] S. Jansen and M. A. Cusumano. Defining software ecosystems: A survey of software platforms and business network governance. In M. A. C. Slinger Jansen and S. Brinkkemper, editors, *Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry*, pages 13–28. Edward Elgar Publishing, 2013.

[14] V. B. Kampenes, T. Dybå, J. E. Hannay, and D. I. K. Sjøberg. Systematic review: A systematic review of effect size in software engineering experiments. *Inf. Softw. Technol.*, 49(11-12):1073–1086, Nov. 2007.

[15] K. Manikas and K. M. Hansen. Software ecosystems - a systematic literature review. *J. Syst. Softw.*, 86(5):1294–1306, May 2013.

[16] D. G. Messerschmitt and C. Szyperski. *Software Ecosystem: Understanding an Indispensable Technology and Industry*. MIT Press, 2005.

[17] A. F. Slinger Jansen 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.

[18] J. Starkweather. Cross Validation techniques in R: A brief overview of some methods, packages, and functions for assessing prediction models. 2012.

[19] H. Wickham. *Advanced R*. Chapman & Hall/CRC The R Series. CRC Press, 2015.