2013 17th European Conference on Software Maintenance and Reengineering

The Evolution of the R Software Ecosystem

Daniel M German
Dept. of Computer Science
University of Victoria
dmg@uvic.ca

Bram Adams
Génie Informatique et Génie Logiciel
École Polytechnique de Montréal
bram.adams@polymtl.ca

Ahmed E. Hassan
School of Computing
Queen's University
ahmed@cs.queensu.ca

***Abstract*—Software ecosystems form the heart of modern companies' collaboration strategies with end users, open source developers and other companies. An ecosystem consists of a core platform and a halo of user contributions that provide value to a company or project. In order to sustain the level and number of high-quality contributions, it is crucial for companies and contributors to understand how ecosystems tend to evolve and can be maintained successfully over time.**

As a first step, this paper explores the evolution characteristics of the statistical computing project GNU R, which is a successful, end-user programming ecosystem. We find that the ecosystem of user-contributed R packages has been growing steadily since R's conception, at a significantly faster rate than core packages, yet each individual package remains stable in size. We also identified differences in the way user-contributed and core packages are able to attract an active community of users.

I. Introduction

Open source development has become a staple of modern software projects, with open source contributors forming an integral part of the software development workforce. Typically, a company remains in charge of the overall direction of a product, but tries to attract a community of open source contributors to build significant parts of its software portfolio. These contributors are not just low-cost labourers, they are typically well-motivated, talented and creative, extending the functionality of a software product in unforeseen ways. A popular example are the explosive growth and profits of user-developed iPhone and Android micro-apps.

To foster collaboration with open source contributors, software projects are typically conceived as so-called "ecosystems" [3], [4], [12], [17], i.e., they consist of a (relatively) closed core that provides the basic functionality, surrounded by an open halo of user contributions. For example, the Eclipse project consists of a basic workbench that can be customized into any kind of IDE or editor by means of thousands of plugins (grouped into subprojects). Linux distributions like Debian provide a Linux operating system kernel that can be turned into any kind of server or workstation by means of thousands of contributed libraries and user applications (distributed as packages).

Given the crucial role of ecosystems, it is important for companies and projects to sustain high-quality user contributions [4]. Keeping customers and contributors happy and productive does not only depend on effective project management, but also (and especially) on agile evolution and maintenance of an ecosystem according to the customers' and contributors' needs. Although there has been some initial research on understanding how ecosystems are able to evolve effectively [6], [9], [10], [34], most of this research considers the core or the user contributions in isolation, or does not distinguish at all between both. This is not sufficient, since ecosystems need to reconcile different stakeholders, agendas and development philosophies, while safeguarding product quality and profits.

This paper studies and contrasts the evolution of the core and user contributions (packages) of the R ecosystem [27]. We chose R as subject ecosystem because it is an immensely popular, vibrant end-user programming ecosystem of statistical analyses, visualizations and datasets based on a dialect of the S statistical programming language [18]. "End-user programming" means that the majority of contributors and core members are not software engineers by trade, but statisticians and scientists. This makes the success and scale of the R ecosystem even more impressive. For these reasons, we believe R to be an ideal subject system for exploring four research questions related to the size, long-time evolution, structure and community of user-contributed packages compared to core packages:

Q1) What are the Code Characteristics of Core and User-Contributed Packages?

We find that user-contributed packages typically are smaller, and contain less documentation than core ones.

Q2) How do Core and User-Contributed Packages Evolve over Time?

On average, the size of a package remains stable over time, whether it is core or user-contributed.

Q3) What is the Dependency Structure among Core and User-Contributed Packages?

On average, packages have few dependencies. User-contributed packages build more on core packages than on other user-contributed packages.

Q4) How is the Community surrounding Core and User-Contributed Packages?

User-contributed packages are discussed less and by less people than core packages. Building a community around core packages takes months compared to a year for user-contributed packages.

This paper is organized as follows. Section II presents the GNU R ecosystem, followed by a discussion of our case study

1534-5351/13 $26.00 © 2013 IEEE
DOI 10.1109/CSMR.2013.33

setup (Section III). The case study results are presented in Section IV and discussed in Section V. We conclude this paper with a discussion of related work in Section VII and our conclusion in Section VIII.

II. The GNU R Ecosystem

There are many definitions of ecosystems [3], [4], [12], [17], but here we adopt the one from Jansen et al. [12], which defines a software ecosystem as (numbers added by us):

> a set of (1) businesses functioning as a unit and interacting with a shared market for (2) software and services, together with (3) the relationships among [the businesses].

For example, the micro-app ecosystems of Apple and Google have (1) as stakeholders Apple/Google, thousands of 3rd party developers and millions of users worldwide. These stakeholders leverage (2) the iOS and Android mobile operating systems, SDKs and APIs, and thousands of 3rd party mobile applications. Users and 3rd party developers depend on (3) the massive App Store and Google Market distribution platforms to buy and sell their micro-apps, for a nominal fee or a small share of the profits.

Although ecosystems have been around for more than a decade, little research has been performed on them [3], [12]. Most of the existing research focuses on formally modeling ecosystems [5], [20], migrating a project into an ecosystem [3], [4], [11], and managing the huge volumes of knowledge spread across the stakeholders of an ecosystem [7], [29]. The majority of this work considers levels (1) and (3), and targets ecosystems on *web* and *mobile* platforms.

Existing work on *desktop* ecosystems has primarily focused on operating system-centric ecosystems like the Linux kernel [9] and Debian ecosystem [10], and developer-centric ecosystems like the Eclipse ecosystem [6], [15], [16], [28], [34]. Desktop ecosystems for end-user programming have seen almost no research. These ecosystems, featuring the likes of Microsoft Excel and high-level database report generators, are based around a (typically domain-specific) language that enables non-software engineering users with a good domain understanding to quickly build their own custom applications. As such, these ecosystems are hugely popular and are believed to be "the holy grail of software platforms" [3].

In order to understand how end-user programming ecosystems evolve, this paper studies the evolution of the libre GNU R desktop ecosystem [27], at the software level (2). Studying this ecosystem at levels (1) and (3) is future work. R's business is statistics and data analytics, with as major stakeholders companies, researchers and governments. The ecosystem is built around the R statistical programming language [18] (which is based on the S language) and nine repositories hosting 4,338 (March 2011 [19]) user-contributed packages that enhance the basic R with advanced statistical analyses, visualization toolkits and comprehensive data sets. In addition to the standard, libre GNU R, the R ecosystem comprises various commercial offerings (e.g., [24], [26]) that provide advanced graphical user interfaces or enhanced scalability.

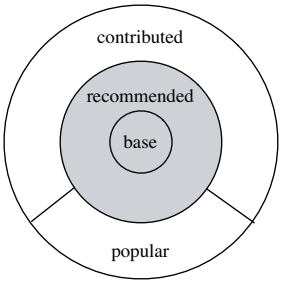

Fig. 1. Breakdown of the different kinds of R packages. The grey packages are the core packages of R, i.e., they are shipped with the basic installation.

We chose the R ecosystem for our case study since it is one of the most successful end-user programming ecosystems for statistical analysis [19], [25], developed primarily by non-software engineers. In March 2011, R was the 27th most popular programming language in the world [31] and it was estimated to be used by 43% of all data miners worldwide [25]. Furthermore, R has a flourishing community of developers and users, which communicate through hundreds of mailing lists, fora, web blogs and books. The R mailing list traffic almost doubles the corresponding traffic of its commercial competitors [19]. To facilitate development, the R ecosystem uses the R-Forge development infrastructure, which provides version control, defect management, and other features similar to those of SourceForge [30].

III. Setup of the Case Study

This paper performs an empirical study on the R ecosystem to explore how a successful ecosystem is able to evolve according to the conflicting requirements of different stakeholders, while maintaining its quality. Our study considers the source code, churn, dependency and community dimensions of R in order to understand the scale of user-contributed packages (Q1), the long-term evolution of such packages (Q2), how the user-contributed packages fit into the core system (Q3) and how a community is formed around user-contributed packages (Q4). This section explains the R data that we are using as well as how we extracted this data.

A. R Package data

R consists of a small kernel, complemented by so-called "packages" that extend the basic functionality of R. There are four different sets of packages. These are described in Table I, and their relationship is illustrated in Figure 1. Similar subsets of packages exist in other ecosystems, like LaTeX and Perl.

We mirrored the `src` subdirectory of the CRAN R repository (http://cran.r-project.org/, which is the largest R package

TABLE I
Sets of R packages used in this study.

Dev. by	Set	Size	Description
R team	*Base*	13	Core pkg. Installed by default.
Users	*Recommended*	15	Core pkg. Installed by default.
	Popular	179	Found to be commonly installed by a sample of users. Does not include *Recommended*, nor *Base*.
	Contributed	2,733	Rest of user-developed pkgs.

repository, amounting to 16 GB of data or roughly two thirds of all currently existing R packages [19]. The `src` subdirectory of the R repository is divided into three parts: the source code of the releases of the R system kernel (`base` subdirectory), the user-contributed packages (`contrib` subdirectory), and a pre-release of R (`base-prerelease` subdirectory, not used in our study). The `base` directory contains every release of R, from version 0.49 (April 23, 1997) to the latest version, i.e., 2.12.2 (Feb. 25, 2011), for a total of 80 versions.

Core packages can be divided into two types: *Base* packages, maintained by the R team, and *Recommended* packages, which are preselected user-contributed packages of which hand-picked versions are included in the `library/` subdirectory of a particular R release. The source code of the *Recommended* packages is shipped as .tar.gz files that need to be further decompressed and built. The latest release of R contains 13 *Base* packages, and 15 *Recommended* packages. To access the source code of earlier versions of *Base* packages, we downloaded all historical R releases.

The `contrib` subdirectory of the CRAN R repository contains the latest release of all user-contributed packages distributed via CRAN (including all versions of *Recommended* packages), as well as all older releases (in `contrib/Archive`). The 2,927 different user-contributed packages on CRAN occupy 15 GB of data and include 19,593 archived releases. We used these archived releases for our historical analysis of non-*Base* packages. The first user-contributed package is dated Oct. 8, 1997 (`ratetables`, version 1.0-2), and the last one March 27, 2011 (`spi`, v1.0).

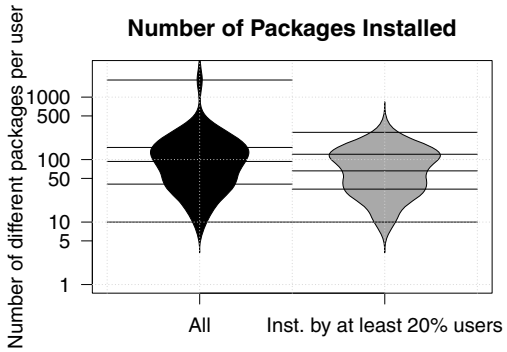

Fig. 2. Bean plots showing the distribution of the number of *Contributed* packages installed per user (52 users). The left side contains all installed *Contributed* packages (1,879 different ones), whereas the right side contains only the *Popular* packages, i.e., those packages installed by at least 20% of the 52 users (194 different packages). The horizontal lines represent the quantiles.

It is very likely that not all 2,927 user-contributed packages are frequently downloaded and installed. To account for this, we also considered a subset of user-contributed packages that are *Popular* packages, i.e., packages that are installed by a significant number of users. To determine the set of *Popular* packages, we use statistics of the packages installed by 52 users, data that was compiled for an official competition to predict what packages a user would likely install [35]. Gonzalez et al. [10] used a similar approach to define *Popular* packages in Debian, while for LaTeX various lists of *Popular* packages can be found online.

The distribution of the number of user-contributed packages installed by each of the 52 users is presented in Figure 2. This Figure contains two beanplots [13]. A beanplot is a boxplot that also plots the density of a distribution instead of just a box, making the plot more informative. As can be observed on the left bean plot, the median number of installed packages was 94.5, with a minimum of 10, and maximum of 1,879 installed by one user. To avoid such outliers, we decided to only keep those user-contributed packages that had been installed by at least 20% of the 52 users (10). The resulting set contains 194 different packages. After removing the 15 *Recommended* packages, we name the remaining 179 packages *Popular* packages. All other user-contributed packages are called *Contributed* packages. Figure 1 and Table I show how these two sets of packages relate to the core packages.

B. Mailing List Data

To analyze the user and developer community in the R ecosystem, we studied the main mailing lists of the R project, i.e., the R-help (users) and R-devel (developers) lists. We did not include the low-volume R-announce and R-packages lists, nor the 21 "Special Interest Groups", such as R-sig-DB and R-sig-teaching. The latter focus on specific scientific fields, which would bias our data.

We downloaded the mailing list archives of R-help [22] and R-devel [21] for the period 1997–2010, and converted them to the standard mbox format using a Perl script. We then used the MailboxMiner tool [2] to extract individual emails from the mbox files, group emails into threads, resolve email aliases and store the resulting data into a relational database. Attachments were ignored.

We used R and Perl scripts to analyze the extracted data. In particular, we linked emails to packages using a lightweight, case-sensitive regular expression search [1] for the packages' names in the email bodies:

```
(?<![a-zA-Z0-9])${PACKAGE}(?![a-zA-Z0-9])
```

This expression requires the package name to be free-standing, i.e., not preceded or followed by a letter or digit. Furthermore, some package names, such as "its" and "graph", clash with regular English words, but since their effect is constant across time, we did not exclude them from our data.

For each package, we only searched through those emails that were sent after the first release of the package. We did not exclude emails sent after the demise of dead packages, since people can still discuss those afterwards. We did not remove quoted email text, since we are not interested in the raw number of occurrences of a package name in an email, but rather whether or not an email (semantically) talks about a particular package.

IV. Empirical Case Study

Q1. What are the Code Characteristics of Core and User-Contributed Packages?

Motivation — Since different stakeholders are involved with the development of different types of packages, it is important to compare the packages across the four sets in terms of characteristics like size and documentation to understand the volume and size of contributions that one might expect in an ecosystem.

Approach — With respect to the number and size of packages and documentation, we present two metrics: the number of files per package, and the file length (in SLOCs for source code, and raw number of lines for documentation files). We concentrate on documentation files (extension .rd – displayed when a user requests help within R), R source code (.r), and aggregated C/C++ files (.c,.cpp, .h, and .hpp – we call them C for brevity). All metrics for *Base* packages are calculated in terms of the latest version of R (version 2.12.2), and for the other packages in terms of their latest version.

Results — **Packages are composed of many different types of files.** Figure 3 shows that the distribution of file types across the four sets of packages is fairly similar: documentation files dominate (.rd), followed by R, C, and .rda (R data files).

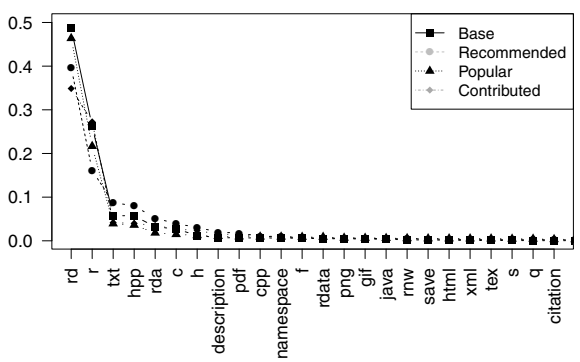

Fig. 3. Proportion of files per file extension across the four sets of packages. The extensions shown correspond to 95% of all files. Packages are dominated by documentation files (.rd), followed by R source code (.r). After that, the most common language is C/C++, followed by Fortran. R data files (.rda) are also very dominant. *Base* and *Recommended* packages have a slightly larger proportion of documentation files.

Packages are extensively documented, with user-contributed packages having less than core packages. In all sets of packages, the proportion of .rd files is the largest (the median of the ratio of rd files to source ones is 1.5, 1.25, 0.89 and 1). As depicted in figure 4, these files account for a median of 5.3k, 3.6k, 1.7k, and 0.6k lines (for the sake of brevity, when reporting four values, they will correspond to *Base*, *Recommended*, *Popular* and *Contributed* respectively). The high percentage of documentation files to any other type suggests that packages are extensively documented.

User-contributed packages have significantly less source code than core packages. As can be observed in Figure 5, the core packages have more code than user-contributed packages. Their median sizes are 13.1k, 6.2k, 3.2k, and 0.9k SLOC.

Packages are implemented primarily in R, distantly followed by C. As shown in Figure 5, R code dominates, with median sizes of 7.3k, 3.5k, 1.8k, and 0.7k SLOC. C has a bimodal distribution, with core packages using C more than user-contributed (median sizes of 184, 850, 0, and 0 SLOC). Given R's target audience, the R language itself is the preferred way of extending functionality.

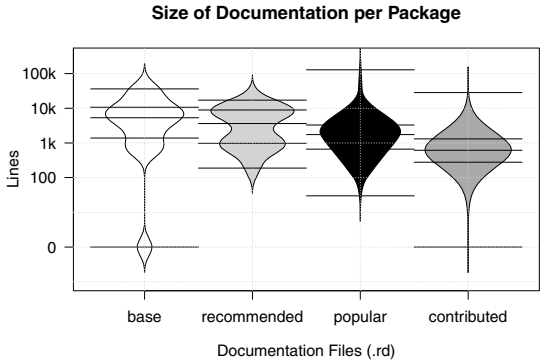

Fig. 4. Size of documentation files per package. The number of lines of documentation decreases from *Popular* to *Contributed* packages (statistically significant with $p < 0.01$). The differences in size between the other three sets of packages are not statistically significant ($p > 0.01$).

For each analyzed metric (length of documentation, and SLOC of all/C/R source code), *Base* packages are larger than *Recommended* packages, which are larger than *Popular* packages, which are larger than *Contributed* packages, i.e., user-contributed packages are smaller than core packages. All packages contain a large proportion of documentation files.

Q2. How do Core and User-Contributed Packages Evolve over Time?

Motivation — If we consider one release of the R ecosystem, question one has shown that there are clear differences between core and user-contributed packages in terms of code and documentation characteristics. Question two considers how core and user-contributed packages have evolved over time to analyze whether those packages are one-off releases or whether they are actively maintained over time. For example, how does the design of the ecosystem and packages evolve? Do the same files or packages keep on growing or are new files or packages continuously added? How well is the documentation maintained, and how does the change rate of documentation compare to the change rate of source code?

Approach — For *Base* packages, we analyzed all versions included in the official R releases, for the other packages (including *Recommended* packages) we used the releases found in the `Archive` section of the CRAN repository. For each

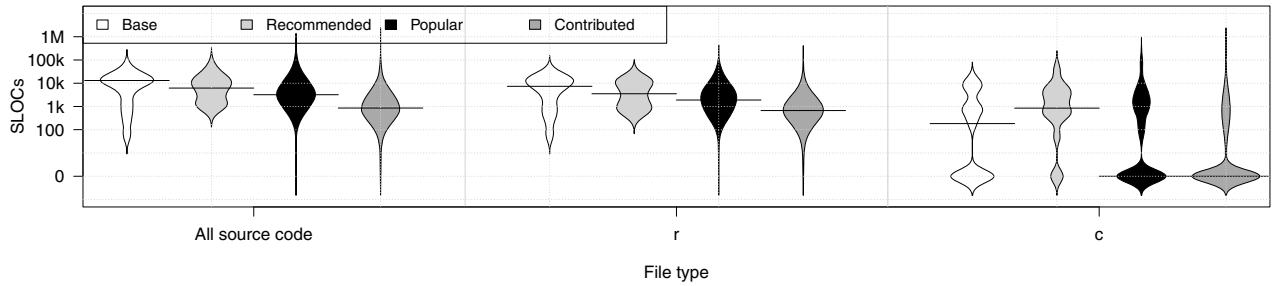

Fig. 5. Size of source code per package, in SLOCs. The only difference that is statistically significant is that *Contributed* packages are smaller than *Recommended* packages, and that they contain less R and C/C++ code ($p < 0\ 01$).

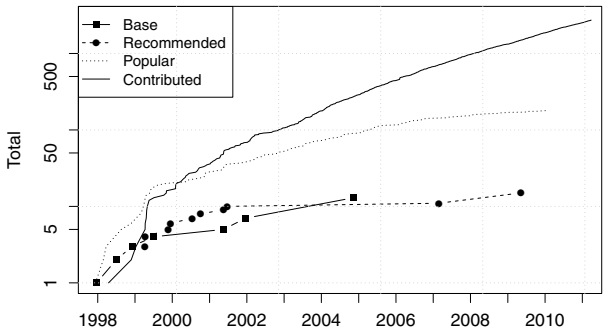

Fig. 6. Number of packages over time. *Contributed* packages are growing faster. *Popular* and *Base* packages have remained stable since 2005.

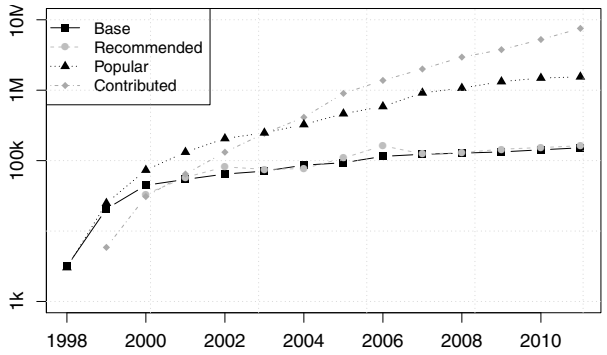

Fig. 7. Evolution of the overall size of all packages' source code. User-contributed packages account for almost 10M SLOC (20 times larger than the basic R).

version of every package, we measured the number of files, the number of lines of source code (in particular the size of R and C/C++ code) and the number of lines of documentation files (.rd).

Results — **Fast growth of user-contributed packages, with a small, stable core.** From Figure 6, we can see that the community is adding more *Contributed* packages every year, i.e., the number of packages follows a super-linear trend (note that the Y axis is in log-scale) [19]. On the other hand, new packages are not gaining popularity (becoming *Popular*) as fast as they are added, nor are they easily added as part of the distribution (*Recommended*).

Figure 7 shows how the size of *Contributed* packages is growing super-linearly, the size of *Popular* packages is growing more slowly, and the size of core packages remains more or less stable (0.15M, 0.15M, 1.5M, and 8.3M SLOC today). If we consider that the source code of the latest release of R is 0.5M SLOCs, user-created packages are contributing a large amount of functionality to the R ecosystem.

The less core a package is, the less releases it has. The number of releases per package, as depicted in Figure 8 shows that more widely used or core packages generally have more releases. *Base* packages are not shown, since they are versioned with the actual R releases (i.e. had up to 81 releases).

The size of an average package remains stable. The evolution of the size of source code (Figure 9) shows that the only packages that have shown some growth are the *Base* ones, but only during their first years. On the other hand, user-contributed packages show a larger variance (the

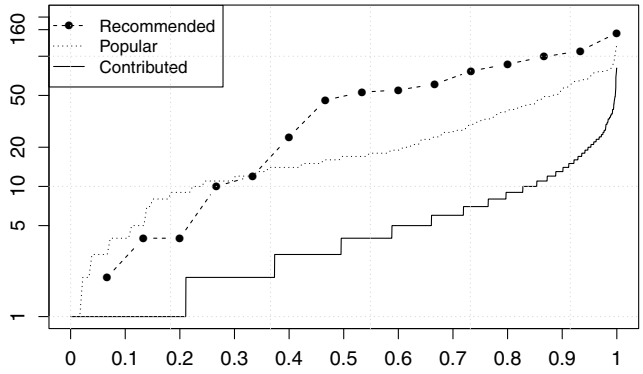

Fig. 8. The proportion of releases per package is larger for *Recommended* packages, followed by *Popular* packages.

smallest packages contain only a handful of lines of code). The evolution of documentation is similar source code's.

***Recommended* packages are the oldest, while *Popular* packages are older than *Contributed* packages; most packages are well-maintained.** *Recommended* and *Popular* packages are older than *Contributed* packages. *Recommended* packages have been growing very slowly (Figure 10), most of them dating back to the same period in 2000–2001, with only a handful of packages from the period 2001–2009. This suggests that R developers are conservative in their decisions to add new packages to the R release. Most packages have been updated in the last year (see Figure 11), suggesting that *Contributed* packages are not a cemetery of code, but rather full of continuous activity. It appears that R developers monitor

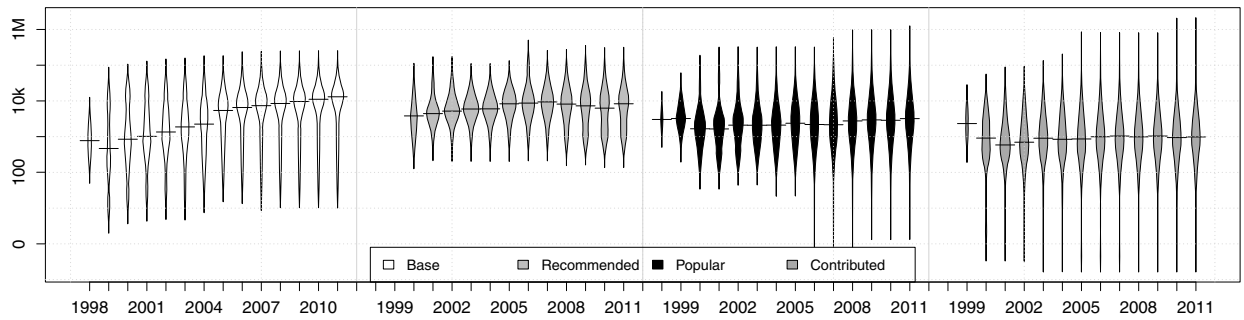

Fig. 9. Evolution of the size of source code of packages. Median package size has remained stable over time. User-contributed packages have a larger variance.

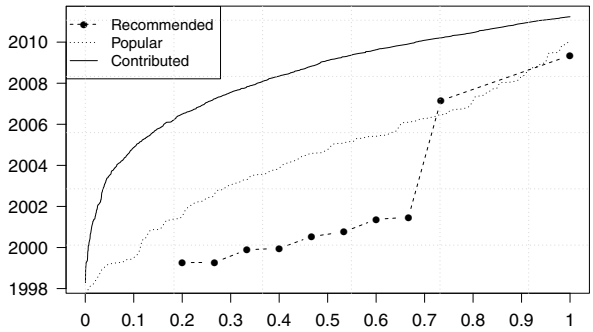

Fig. 10. Date in which a package was created. Core packages are older.

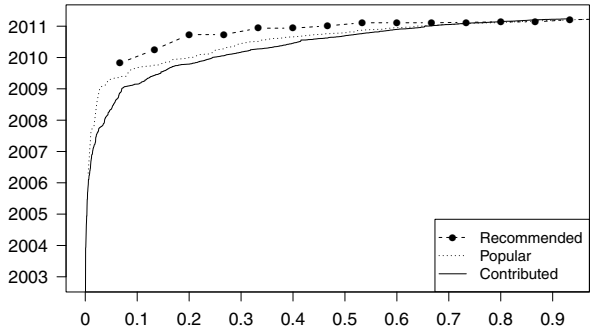

Fig. 11. Last date of a package's release. More than 95% of packages have been updated in the last 2 years. *Recommended* packages are slightly more recent than the other packages.

the packages frequently. If the packages break (either during compilation or testing) and their maintainer does not fix the problem, the package is removed (55 packages to date).

> Packages are typically well-maintained. Over time, most packages have remained fairly stable in size, while the number of packages increases.

Q3. What is the Dependency Structure among Core and User-Contributed Packages?

Motivation — As suggested by Figure 1, *Contributed* and *Popular* packages are built on the core R system, requiring the functionality provided by the *Base* and *Recommended* packages. Previous studies on other ecosystems [4], [10] have shown that the dependency structure of an ecosystem, in particular between the core and *Contributed* packages, can be very complicated. Similar to API evolution [36], changes to a core package can cause defects that propagate to possibly dozens of dependent *Contributed* packages. Furthermore, the more dependencies a package has, the more complicated its deployment becomes and the more advanced package management systems need to be. In this question, we analyze the dependency structure of the R project to understand how the project is able to reduce change and defect propagation.

Approach — In R, each package is expected to contain a file called `DESCRIPTION` similar to the `control` and `spec` files used to describe packages in Debian and RedHat Linux distribution respectively. The `Requires` entry in the `DESCRIPTION` file lists all packages that are needed before the package can be installed. By extracting the `Requires` entries of all the R packages, we built a complete inter-dependency graph of the packages, similar to that of Linux distributions [8]. This graph permits to understand how packages build on the features of others. Unfortunately, *Base* packages are not listed in the `Requires` field, hence we did not analyze how these packages are built upon.

Results — **The number of dependencies per package is consistently low across all package sets.** Figure 12 shows the number of dependencies per package, that is, the number of packages required by a given package to be installed. As can be observed, almost 1/3 of all packages have no dependencies, and 73% have less than 2 (for reference, Debian 4.0 had a median of 3 dependencies per package [10]). A Wilcoxon rank sum test shows that the three distributions in the Figure are not statistically different (i.e., the number of dependencies is independent of the package's set).

The less core a package is, the less packages depend on it. Figure 13 shows the number of dependents per package (i.e., the number of packages that require the package). As can be observed, 72% of the *Contributed* packages are not required by any other package (similar to Debian 4.0 packages [10]). On the other hand, *Popular* packages (median of 6 dependents) and *Recommended* packages in particular (median of 30 dependents) are significantly likely to be required by other packages. As expected, packages closer to the core tend to perform lower-level functionality needed by user-contributed packages. For example, *Recommended* packages like `lattice`

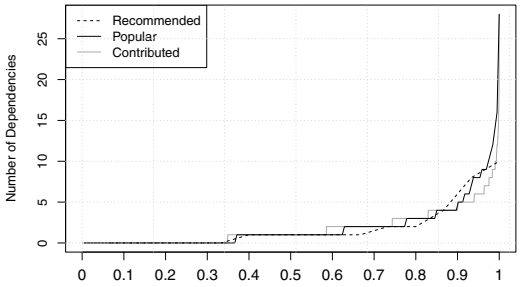

Fig. 12. The number of dependencies per package in R, i.e., the number of packages that is required to be installed by each package. Most packages have few dependencies (1 out of 3 have none).

and `MASS` provide extremely popular data visualization and general statistical functions, respectively.

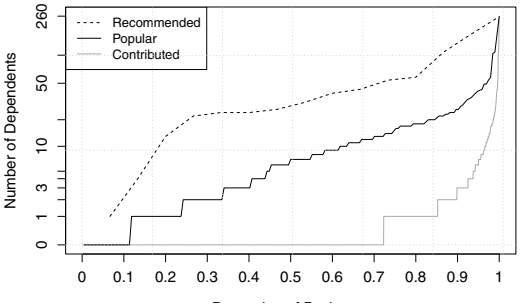

Fig. 13. Number of dependents per package, i.e., the number of packages that require (depend on) a particular package. Most packages do not have dependents. As expected, popular packages have significantly more dependents.

Most packages have few or no dependencies. On the other hand, *Popular* and *Recommended* packages are more likely to be required by other packages.

Q4. How is the Community surrounding Core and User-Contributed Packages?

Motivation — Since ecosystems rely on the involvement of a community of users and developers [4], user contributed packages are generally developed by stakeholders external to the core system's company or project. Hence, the main channels of communication with those stakeholders are mailing lists and IRC chats, in addition to irregular gatherings, such as the yearly useR! conference for R stakeholders [32]. This means that we can use email and IRC traffic to quantify the degree of interest of users and developers into the ecosystem, as well as the active maintenance and development areas.

Approach — First, we measure the number of email messages and senders in the R-help and R-devel mailing lists across the four sets of packages. The R-help mailing list is a measure of community participation and interest, whereas the R-devel mailing list, which is aimed at developers of packages, measures development and maintenance activity. Second, for each package we calculate the total number of messages and the time from the first version of a package to the first, tenth, hundredth and thousandth message, to measure how fast a community builds up around a new package.

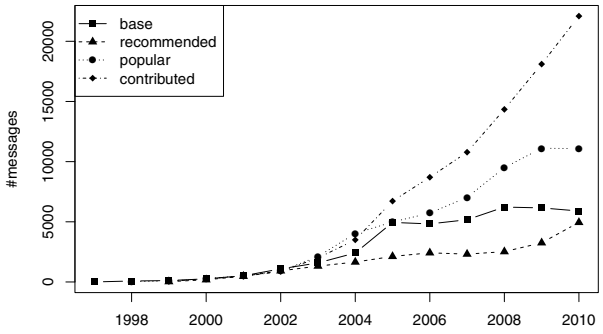

Fig. 14. Number of messages per set of packages per year (R-help).

Results — **Mailing list traffic is dominated by *Contributed* and *Popular* packages.** Figures 14 and 15 plot the total number of messages to respectively the R-help and R-devel mailing lists, broken down across the four sets of packages. The majority of traffic on both lists targets *Contributed* packages. Since 2003, this traffic has seen a continuous growth on the R-help list, but it has stagnated on the R-devel list from 2006, except for a peak in 2009.

Traffic on *Base* and *Recommended* packages has stagnated on both mailing lists since 2005. Although *Base* packages are still the second largest source of traffic on the R-devel mailing list, their traffic used to be near that of *Contributed* packages (until 2004 in R-help and 2006 in R-devel). Whereas the traffic of *Contributed* (and later *Popular*) packages boomed, the *Base* and *Recommended* traffic stopped growing or even reclined. The last five years, the core packages have seen an almost stable volume of traffic.

The user communities of R packages grow unbounded, whereas developer communities stagnate. Figures 14 and 15 correlate strongly with the number of people sending messages each year (plots not shown), with a Spearman correlation of more than 0.93. Since 2005, the number of people participating in the development communities of the R-devel mailing list has remained more or less stable, in contrast to the ever growing yearly number of users participating on the R-help mailing list. The maximum number of active message senders is around 3,000 for the R-help mailing list (2010) compared to 500 for the R-devel list (2009).

The less core a package is, the lower its concentration of messages and message senders. Figure 16 plots for each package set the distribution of the number of R-help messages per package. The corresponding plot of the number of message senders, and the plots for the R-devel mailing lists are similar, and hence not shown. Whereas the total volume of traffic of *Contributed* and *Popular* packages was the highest over time (Figures 14 and 15), their traffic per package is relatively low, i.e., many packages have to share the spotlight. On the other hand, the steady flow of messages about core packages focuses entirely on a small set of packages, leading to a

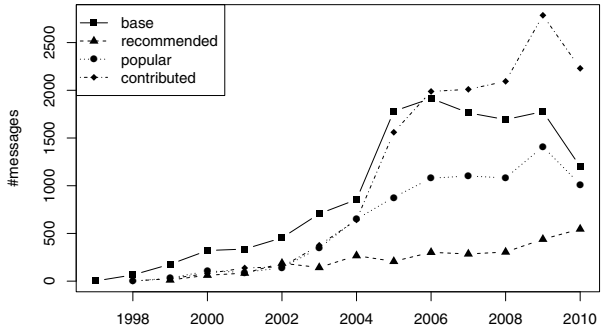

Fig. 15. Number of messages per set of packages per year (R-devel).

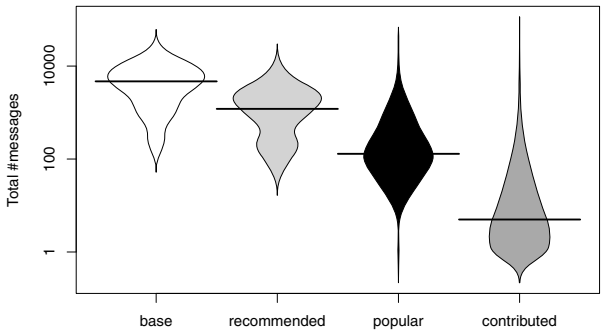

Fig. 16. Distribution of the number of messages per package and set (R-help).

high concentration of messages per package. Similarly, the core packages are discussed by more message senders than *Contributed* packages.

It appears to take one year (median) to start up a community for *Popular* and *Contributed* packages, whereas core packages only take a couple of months. Figure 17 plots for each set of packages the distribution of the time to the first, tenth, hundredth and thousandth message of each package (R-help; the beanplots for R-devel are similar). Core packages have the lowest start-up time. The median time to the first and tenth message of a *Base* package is only 1 week and 82 days respectively, compared to 31 days and 369.5 days for *Popular* packages. As found earlier for Q2, new *Contributed* packages need to face fierce competition to receive more popularity.

It is also interesting to compare *Popular* packages to *Contributed* packages, since the former were successful at gaining momentum, whereas the latter failed to do so. Figure 17 suggests that *Popular* packages in general took less time to build up a community than the other contributed packages. The latter take two months longer to their first message, and 5 months longer to their tenth message. In other words, popular contributed packages seem to be picked up sooner.

Note that not all packages are discussed one thousand, one hundred or even ten times. Table II shows that the probability of reaching 1, 10, 100 or 1,000 messages on a mailing list decreases the less core a package is. For example, only 0.52% of the *Contributed* packages (9 packages) on R-help and 0.1% (1 package) on R-devel reach 1,000 messages, compared to 69.2% (9 packages) and 23.1% (3 packages) of the *Base*

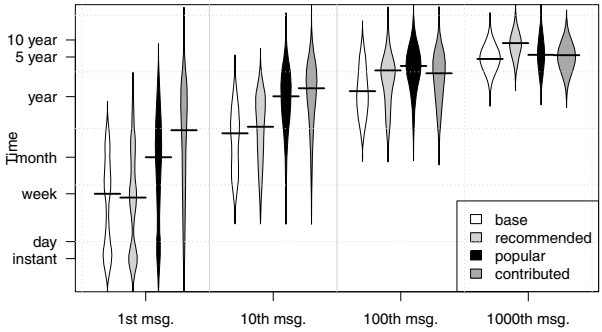

Fig. 17. Distribution of the time to the 1st/10th/100th/1000th message of a package per set in the R-help mailing list (R-devel is similar).

TABLE II
PERCENTAGE OF *Base*, *Recommended*, *Popular* AND *Contributed* PACKAGES REACHING 1, 10, 100 OR 1,000 MESSAGES ON THE MAILING LISTS.

	R-help				R-devel			
	1	10	100	1,000	1	10	100	1,000
base	100	100	100	69.2	100	100	92.3	23.1
recomm.	100	100	86.7	33.3	100	93.3	53.3	0
pop.	100	97.7	44.9	2.3	100	58.5	4.0	0.6
contrib.	100	31.2	6.5	0.52	100	11.1	1.3	0.1

packages. These numbers confirm the findings of Figure 16.

User-contributed packages are discussed by less people than core packages. Communities are build around core packages in months compared to a year for user-contributed packages.

V. DISCUSSION

The results of our case study are summarized in Table III. Many of these results are obvious (e.g., core packages are more documented, are older and have more dependents than user-contributed). Others are not: the size of packages is similar among all, most all well maintained and most have few dependencies. We believe that this is due to the infrastructure created by R to provide an efficient and automatic dependency management system that frees the end-user from the complexities of dependencies between packages (all the user has to do is `install.package("name")`). The R ecosystem also strongly relies on conventions and best practices, for example regarding documentation, release continuity and mailing list traffic. Packages are tested nightly. If their tests fail (packages must have tests) their are quarantined, and eventually removed if not fixed. The policies, documentation and infrastructure provided by the R ecosystem appears to increase both the number of contributors and user productivity, and improve its success. However, these are hypotheses that should be tested with further research.

Regarding the R community, we found that user contributions need to fight for attention. It is not clear, however, how users deal with this. What are their motivations to create and contribute packages? What is the typical evolution and maintenance process of a user-contributed package? How much effort is invested in creating and maintaining an R package?

TABLE III
SUMMARY OF OUR ANALYSIS RESULTS.

	core	user-contributed
#documentation and code	more	less
#packages	low and stable	super-linear
age	older	younger
# releases	more	less
#dependents	more	less
total traffic	less	more
traffic per package	more	less
community start-up time	months	year
average size	stable	
maintenance	well-maintained	
#dependencies	low	
communities	user growing, developer stable	

How many people participate? How important are the features of R-Forge to the success of a package? And perhaps most importantly, the R ecosystem provides an interesting case study on how non-software engineers build and maintain software systems. An area that we have not touched on is the social and socio-technical networks created by contributors and how they relate to the evolution and success of R. Future work should concentrate on them.

We have seen how each set of packages evolves over time in terms of size and maintenance, and that it can take a long time for packages to build up a community. Yet, how does a package go from being created to being *Popular*, and how are *Popular* packages promoted to *Recommended* packages? This problem is similar to the problem of determining which packages to include on the first CD of a Linux distribution. In the R ecosystem, blogging seems to have a large impact on package popularity. Aggregators like R-bloggers [23] highlight sites around the Web that use R, frequently presenting demonstrations of what R is capable of doing, including source code. Similarly, books teaching R introduce users to packages (or have their own accompanying packages), and have a direct impact on what users use. Being an open source project, the success of R is likely to be driven by its users (and the media attention created around it, such as [33]).

The above discussion highlights many of the different facets of ecosystems that are as yet unexplored. We believe our paper to be a first exploratory step towards unraveling the secrets of software ecosystems.

VI. THREATS TO VALIDITY

Threats to construct validity [37] relate to whether our measurements quantify what we intended them to. Our definition of *Popular* packages depends on the results of a poll [35], however informal discussions with fellow researchers learns that the results are representative. The mailing list analysis is based on regular expressions, and might lead to false positives. We did not analyze dependencies on *Base* packages, since this information was not provided in the package specification files. We plan to use static source code analysis techniques for this.

Threats to internal validity concern rival hypotheses that can explain our findings. We assume that mailing list popularity is an indicator of community building. However, it might just be an indicator of fault-proneness or complexity of a package (i.e., many questions about it). Also, we only studied mailing list communication, no instant messaging or blogs.

External validity relates to the generalization of our study results. We only studied one major end user-programming ecosystem, which is open source. Hence, we need to perform similar case studies on related ecosystems like LaTeX, Perl and PHP, as well as analyze closed source systems.

VII. RELATED WORK

As mentioned in Section II, the closest related work is the research on the software evolution and community of operating system- and application-centric desktop ecosystems.

Research on operating system-centric desktop ecosystems has especially focused on the Linux kernel [9] and the Debian Linux distribution [8], [10], which consists of the Linux kernel and thousands of libraries and end user applications. The Linux kernel ecosystem, just like R, grows at a super-linear rate, especially because of the massive contributions of device drivers. Contrary to R contributions, those drivers are typically cloned and customized [14], rapidly increasing the kernel code size. Legacy drivers tend to linger around, whereas R promptly removes unused packages.

Debian is built around a sophisticated, dependency-aware package management system and a number of large package repositories, similar to R. The size of the Debian ecosystem doubles in size every 2 years, totaling over 300 MLOC in 2007 [10]. This growth is caused by the continuous growth of the larger packages and addition of small, new packages. The latter is facilitated by the high degree of reuse of core libraries, rapidly increasing the number of dependencies between packages. Still, large portions of the code did not change in a period of 9 years.

We found that the growth rate of R approaches that of Debian: overall, the size of user-contributed packages has doubled in the last two years, primarily because many new packages are added. The number of dependencies per package is similar to that of Debian packages.

The ecosystem of Eclipse has been widely studied. Wermelinger et al. [34] identified a stable core of Eclipse plugins whose dependencies have remained stable over time. This definition of core is different from ours, which is based on the conceptual architectural of R (not its evolution). Furthermore, the number of Eclipse plugins has grown by a factor of five over a period of 6 years. In contrast, the number of R packages has doubled in the last three years, with 25% of the current packages being created in the last 15 months.

Two studies analyze whether the evolution of Eclipse's core [16] (roughly equivalent to our *Base* and *Recommended* packages) and 3rd party Eclipse plugins [6] (equivalent to our *Popular* and *Contributed* packages) follow Lehman's laws of software evolution. The core Eclipse plugins adhere to the laws of continuing change and growth, but not of increasing complexity [16]. The number of *Base* and *Recommended* Eclipse plugins has tripled over time, growing on average 260 kLOC per major release. Businge et al. [6] found support

for Lehman's laws of continuing change, self-regulation and continuing growth for the evolution of 21 3rd-party plugins.

The studies mentioned thus far do not consider the community aspect of ecosystems, like documentation, bug reports or mailing list traffic. Schackmann et al. [28] performed an empirical study of the change request process of the *Base* and *Recommended* Eclipse plugins. They found that the quality of change requests and the time to triage a request vary significantly across plugins. Kidane et al. [15] analyzed the development mailing lists and bug repositories of 33 Eclipse subprojects (collections of plugins). They found that adding new features slows down the bug fix process and that more distributed social networks foster the development of new features. Finally, Yu studied the mailing lists of the Linux kernel to analyze different ecosystem collaboration patterns between companies [38]. We analyzed the R-help and R-devel mailing lists to measure the size, growth and dynamics of the R ecosystem community.

VIII. Conclusion

In this exploratory empirical study, we have shown that R is a flourishing ecosystem of user-contributed packages that is growing super-linearly, with a strong set of core packages.

In these ecosystems, it becomes impossible to separate the evolution of the underlying system from the evolution of its community of users and contributions. After all, the power of R is in the features that user-contributed packages implement. Hence, ensuring a healthy community around contributions of high quality (size and documentation) and well-maintained (new releases and collaboration) packages is essential according to our findings. In future work, we plan to continue studying the factors that facilitate healthy evolution in other ecosystems, both end-user as other kinds of ecosystems.

References

[1] A. Bacchelli, M. Lanza, and R. Robbes. Linking e-mails and source code artifacts. In *Proc. of the 32nd ACM/IEEE Intl. Conf. on Software Engineering (ICSE) – Volume 1*, pages 375–384, 2010.

[2] N. Bettenburg, E. Shihab, and A. E. Hassan. An empirical study on the risks of using off-the-shelf techniques for processing mailing list data. In *Proc. of the 25th IEEE Intl. Conf. on Software Maintenance (ICSM)*, pages 539–542, 2009.

[3] J. Bosch. From software product lines to software ecosystems. In *Proc. of the 13th Intl. Software Product Line Conference (SPLC)*, pages 111–119, San Francisco, California, 2009.

[4] J. Bosch and P. Bosch-Sijtsema. From integration to composition: On the impact of software product lines, global development and ecosystems. *J. Syst. Softw.*, 83:67–76, January 2010.

[5] V. Boucharas, S. Jansen, and S. Brinkkemper. Formalizing software ecosystem modeling. In *Proc. of the 1st Intl. Wrksh. on Open Component Ecosystems (IWOCE)*, pages 41–50, 2009.

[6] J. Businge, A. Serebrenik, and M. van den Brand. An empirical study of the evolution of eclipse 3rd-party plug-ins. In *Proc. of the Joint ERCIM Wrksh. on Software Evolution and Intl. Wrksh. on Principles of Software Evolution (IWPSE-EVOL)*, pages 63–72, 2010.

[7] M. Cataldo and J. D. Herbsleb. Architecting in software ecosystems: interface translucence as an enabler for scalable collaboration. In *Proc. of the 4th European Conf. on Software Architecture (ECSA): Companion Volume*, pages 65–72, 2010.

[8] D. M. German, J. M. González-Barahona, and G. Robles. A model to understand the building and running inter-dependencies of software. In *Proc. of the 14th Working Conf. on Reverse Engineering (WCRE)*, pages 130–139, 2007.

[9] M. W. Godfrey and Q. Tu. Evolution in open source software: A case study. In *Proc. of the Intl. Conf. on Software Maintenance (ICSM)*, pages 131–142, 2000.

[10] J. M. González-Barahona, G. Robles, M. Michlmayr, J. J. Amor, and D. M. German. Macro-level software evolution: a case study of a large software compilation. *Empirical Softw. Engg.*, 14:262–285, June 2009.

[11] S. Jansen, S. Brinkkemper, and A. Finkelstein. Business network management as a survival strategy: A tale of two software ecosystems. In *Proc. of the 1st Intl. Workshop on Software Ecosystems (IWSECO)*, September 2009.

[12] S. Jansen, A. Finkelstein, and S. Brinkkemper. A sense of community: A research agenda for software ecosystems. In *31st Intl. Conf. on Software Engineering (ICSE): Companion Volume*, pages 187–190, May 2009.

[13] P. Kampstra. Beanplot: A boxplot alternative for visual comparison of distributions. *Journal of Statistical Software, Code Snippets*, 28(1):1–9, 10 2008.

[14] C. J. Kapser and M. W. Godfrey. "cloning considered harmful" considered harmful: patterns of cloning in software. *Empirical Softw. Engg.*, 13:645–692, December 2008.

[15] Y. H. Kidane and P. A. Gloor. Correlating temporal communication patterns of the eclipse open source community with performance and creativity. *Comput. Math. Organ. Theory*, 13:17–27, March 2007.

[16] T. Mens, J. Fernández-Ramil, and S. Degrandsart. The evolution of eclipse. In *Proc. of the 24th IEEE Intl. Conf. on Software Maintenance (ICSM)*, pages 386–395, September 2008.

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

[18] F. Morandat, B. Hill, L. Osvald, and J. Vitek. Evaluating the design of the r language: objects and functions for data analysis. In *Proc. of the 26th European Conf. on Object-Oriented Programming (ECOOP)*, pages 104–131, 2012.

[19] R. A. Muenchen. The popularity of data analysis software. http://bit.ly/eJJOif. April 2011.

[20] O. Pettersson, M. Svensson, D. Gil, J. Andersson, and M. Milrad. On the role of software process modeling in software ecosystem design. In *Proc. of the 4th European Conf. on Software Architecture: Companion Vol.*, pages 103–110, 2010.

[21] R-devel list. https://stat.ethz.ch/pipermail/r-devel/. March 31, 2011.

[22] R-help list. https://stat.ethz.ch/pipermail/r-help/. March 31, 2011.

[23] R-bloggers. http://bit.ly/7ydKyD. April 2011.

[24] http://bit.ly/bmpDeK. April 2011.

[25] Rexer analytics 4th annual data miner survey 2010. http://bit.ly/fryQGr. April 2011.

[26] R+. http://bit.ly/xbiSOv. April 2011.

[27] GNU R. http://bit.ly/FQZd. April 2011.

[28] H. Schackmann, H. Schaefer, and H. Lichter. Evaluating process quality based on change request data — an empirical study of the eclipse project. In *Proc. of the Intl. Conf. on Software Process and Product Measurement (IWSM/MENSURA)*, pages 227–241, 2009.

[29] D. Seichter, D. Dhungana, A. Pleuss, and B. Hauptmann. Knowledge management in software ecosystems: software artefacts as first-class citizens. In *Proc. of the 4th European Conf. on Software Architecture (ECSA): Companion Vol.*, pages 119–126, 2010.

[30] S. Theul and A. Zeileis. Collaborative software development using r-forge. *The R Journal*, 1(1):9–14, 2009.

[31] Tiobe index. http://bit.ly/rItE. April 2011.

[32] useR! 2011. http://bit.ly/fQGPc2. April 2011.

[33] S. Vance. Data Analysts Captivated by R's Power. Business Computing, The New York Times. Jan. 6, 2009.

[34] M. Wermelinger and Y. Yu. Analyzing the evolution of eclipse plugins. In *Proc. of the 2008 intl. working conf. on Mining Software Repositories (MSR)*, pages 133–136, Leipzig, Germany, 2008.

[35] J. M. White and D. Conway. Using Data Tools to Find Data Tools, the Yo Dawg of Data Hacking. http://bit.ly/aHl5TQ, Oct 2010.

[36] Z. Xing and E. Stroulia. API-Evolution Support with Diff-CatchUp. *IEEE Trans. Softw. Eng.*, 33:818–836, December 2007.

[37] R. K. Yin. *Case Study Research: Design and Methods - Third Edition*. SAGE Publications, London, 2002.

[38] L. Yu, S. Ramaswamy, and J. Bush. Software evolvability: An ecosystem point of view. In *Proc. of the IEEE Intl. Wrksh. on Software Evolvability*, pages 75–80, 2007.