When *GitHub* meets *CRAN*: An Analysis of Inter-Repository Package Dependency Problems

Alexandre Decan, Tom Mens, Maëlick Claes and Philippe Grosjean
COMPLEXYS Research Institute
University of Mons, Belgium
Email: { first . last } @ umons.ac.be

***Abstract*—When developing software packages in a software ecosystem, an important and well-known challenge is how to deal with dependencies to other packages. In presence of multiple package repositories, dependency management tends to become even more problematic. For the R ecosystem of statistical computing, dependency management is currently insufficient to deal with multiple package versions and inter-repository package dependencies. We explore how the use of GitHub influences the R ecosystem, both for the distribution of R packages and for inter-repository package dependency management. We also discuss how these problems could be addressed.**

***Keywords*-software repository mining; software ecosystem; package dependency management; software evolution; software distribution; GitHub; CRAN**

I. Introduction

There are many popular languages, tools and environments for statistical computing. On the commercial side, among the most popular ones are SAS, SPSS, Statistica and Stata. On the open source side, the R language and its accompanying software environment for statistical computing (www.r-project.org) is a strong competitor, regardless of how popularity is measured [1].

R forms a *package-based software ecosystem*. Its package management system provides an easy way to install third-party code and datasets alongside tests, documentation and examples [2]. The main R distribution installs a few *base* and *recommended* packages.

Thousands of additional packages are developed and distributed through different repositories. *CRAN*, the *Comprehensive R Archive Network* (see cran.r-project.org), constitutes the official R repository offering both source and precompiled stable packages compatible with the latest version of the R environment. Getting one's package accepted on *CRAN* can, however, be a painful process, due to the strict quality policy imposed by *CRAN*. In addition, the large number of available non-archived packages ($>$ 7,400 in November 2015) is becoming a bottleneck [2] and leads to package dependency problems: *"the number of packages on* CRAN *and other repositories has increased beyond what might have been foreseen, and is revealing some limitations of the current design. One such problem is the general lack of dependency versioning in the infrastructure."* [3]

R packages can also be distributed on other official repositories such as *BioConductor* (bioconductor.org) and several smaller repositories such as *Omegahat*. There are also commercial R packages being sold by companies such as *Revolution Analytics*. Finally, many open source R packages are being developed on public version control repositories such as *R-Forge* (dedicated to R packages) and *GitHub* (general purpose).

Given the increasing popularity of *GitHub* as a platform for distributed software development ($>$ 10 million repositories since December 2013, of which several thousands of actively maintained R packages) this paper sets out to explore how *GitHub* is used for developing and distributing R packages. Leek's blog [4] summarises some of the concerns: *"one of the best things about the R ecosystem is being able to rely on other packages so that you don't have to write everything from scratch. But there is a hard balance to strike with keeping the dependency list small."*

To study how *GitHub* is influencing the R ecosystem, in particular with respect to package dependencies, we focus on the following two research questions:

*RQ*1: *To which extent do R developers distribute their packages on* GitHub*?*
Driven by its increasing popularity, we analyze the development and the distribution of R packages on *GitHub*. We observe that more and more new packages are hosted on *GitHub*. While *CRAN* already distributes many of them, we bring supporting evidence that *GitHub* is increasingly used as a distribution platform for R packages.

*RQ*2: *To which extent do R packages suffer from inter-repository dependency problems?*
While the metadata of each R package specifies its *dependencies* to other packages, this is currently insufficient to deal with multiple package versions and inter-repository package dependencies. We confirm our observations through e-mail interviews with several R package maintainers that are actively involved on *GitHub*.

In particular, we illustrate that many R packages hosted on *GitHub* are subject to inter-repository dependency problems prohibiting their automatic installation. Based on our observations, we provide actionable guidelines for the R development community to deal with the aforementioned

problems. In particular, to address inter-repository package dependency management, R package users and developers would benefit from a package installation manager that relies on a central listing of available packages, and a package dependency manager that provides built-in support for inter-repository dependency version constraint satisfaction.

The remainder of this paper is organized as follows. Section II presents related work. Section III explains the data extraction process. Section IV provides an overview of the R package ecosystem on *GitHub* and *CRAN*, and addresses the first research question. Section V deals with the second research question. Section VI discusses the threats to validity of our research, Section VII presents future work, and Section VIII concludes.

II. Related Work

In previous work [5], we explored the ecosystem of R packages, and found that such packages are developed and distributed through a variety of platforms, including *CRAN*, *BioConductor*, *GitHub*, *R-Forge* and many others. For the distribution of R packages, *CRAN* is the official source, while for their development *GitHub* has become the most popular choice. This is why *CRAN* and *GitHub* are the focus of the current paper.

In [6], we studied the maintainability of *CRAN* packages in terms of errors discovered by the R CMD check tool, and how this relates to package dependencies and package updates. Based on these insights we created a web-based dashboard for helping *CRAN* package maintainers to deal with such issues [7]. However, we did not consider dependencies to external package sources such as *GitHub*.

Other researchers have studied the evolution of R packages, though mainly restricted to the official *CRAN* distribution. Ooms [3] discussed the problems with dependency management of R packages, and proposed ways to overcome these problems. Germán et al. [8] studied the evolution of *CRAN* by comparing the characteristics, growth, dependencies and community structure of core packages and user-contributed packages. They also analyzed the user and developer communities by studying mailing list traffic.

While not being specifically related to R packages, *GitHub* has become an even more popular subject of research. Without aiming to be complete, we point to some relevant references here. Dabbish et al. [9] focused on the social and community aspects of *GitHub*. Blincoe et al. [10] used user-specified cross-references between projects to identify ecosystems in *GitHub*. Vasilescu et al. [11] compared the involvement on *GitHub* with the activity on Stack Overflow. Vasilescu et al. [12] studied a large sample of *GitHub* projects developed in Java, Python and Ruby. They compared direct code modifications (commits) with indirect ones (pull requests) and related this to success or failure of continuous integration with TRAVIS-CI. Thung et al. [13] used the PageRank algorithm to identify influential developers and projects on a subnetwork of *GitHub*. Yu et al. [14] studied patterns of communities found in Github's social network.

Software package dependency analysis has been the subject of study of many researchers. Santana et al. [15] visually explored socio-technical relationships between software projects of an ecosystem. Gonzalez-Barahona et al. [16] studied the evolution of the size of Debian packages and the importance of the programming languages used by the applications contained in these packages. Germán et al. [17] described how to model and visualize dependencies between software components needed to build and run applications. Bavota et al. [18] carried out an empirical study of the evolution of inter-project dependencies in a subset of the Apache ecosystem, consisting of 147 projects. Abate et al. [19], [20] also studied problems related to package dependency management and package installation in the context of the Linux Debian package distribution.

Many of the solutions proposed above are likely to be applicable to R packages as well, but do not take into account the presence of multiple co-existing package distributions.

III. Data Extraction

Every R package has a *DESCRIPTION* file presenting its metadata. Fig. 1 shows an example of such a file. Among others, this file lists the packages it depends upon. For our analysis, we consider as dependencies the packages that are listed in the *Depends* and *Imports* fields only, as these are the ones that are required to install and load a package.

```
Package: SciViews
Type: Package
Title: SciViews GUI API - Main package
Imports: ellipse
Depends: R (>= 2.6.0),
         stats, grDevices, graphics, MASS
Enhances: base
Version: 0.9-5
Date: 2013-03-01
Author: Philippe Grosjean
License: GPL-2
```

Figure 1. Part of the *DESCRIPTION* file of the R SciViews package.

In order to answer our research questions, we extracted historical metadata from R packages hosted on *CRAN* and *GitHub*:

CRAN – The metadata of R packages on *CRAN* was retrieved using extractoR, a publicly available R package (github.com/ecos-umons/extractoR) that we already developed and used in earlier work [6]. It downloads the *CRAN* package sources, extracts their contents and stores the *DESCRIPTION* file metadata into data.frame objects. Using this tool we collected, since September 2013, the metadata of 7,871 packages with their associated versions and dependencies. This represents 49,393 distinct *DESCRIPTION* files.

We explored in detail the state of *CRAN* on 1st June 2015, which consists of 6,706 unarchived *CRAN* packages (whose history totalizes 44,459 distinct *DESCRIPTION* files).

GitHub – We collected from *GitHub* all repositories related to R that had a *Push* event that occurred between 1st January 2015 and 1st June 2015. We excluded all forks. This resulted in a huge list of candidate repositories that could contain R packages, R scripts, datasets, etc. To identify which ones contain R packages, we looked for the presence of a *DESCRIPTION* file in the repository. We restrict the repositories to the ones for which we found such a *DESCRIPTION* file at the root of the repository, because this is where R tools that aim to install R packages from *GitHub* (such as devtools::install_github) expect a package to be located. We are fully aware that, by doing so, we ignored a number of relevant R packages hosted on *GitHub*. Importantly, we filtered out the repositories belonging to the *GitHub* accounts cran and rpkg, as these two accounts were used to mirror (part of) the contents of *CRAN*. This resulted in 4,512 distinct R packages on *GitHub*, totaling 50,368 distinct *DESCRIPTION* files on *GitHub* (considering all versions of each such file).

In addition to this historical package metadata, we also identified R package maintainers that were active on *GitHub*. Through email we interviewed five of them (three male and two female to avoid gender bias) that were not affiliated in any way to the authors of this article, with respect to our research questions. To confirm our observations, anonymized and sanitised excerpts of these email messages have been inserted (with permission) at various places in this paper (marked with [21]).

In order to facilitate replicating our study, a *GitHub* repository (github.com/ecos-umons/SANER2016) has been created. It contains all the above datasets, Python notebooks, as well as a technical report containing the email interviews with R package maintainers.

IV. *RQ*1: To which extent do R developers distribute their packages on *GitHub*?

The R environment exists since 1993, and has continued to grow in popularity ever since. A similar thing can be said for the much more recent *GitHub* platform for distributed software development. Since its introduction in April 2008 it has become one of the most popular development platforms. It hosts over 10 million repositories since December 2013.

Because of *CRAN*'s longevity, its size and its historical role of being the official distribution platform for R packages, developers often choose to distribute their packages on *CRAN*. As reported by Karl Broman in his insightful R package tutorial [22], *"The main advantage to getting your package on* CRAN *is that it will be easier for users to install (with* install.packages*). Your package will also be tested daily on multiple systems."*

Some aspects of *CRAN*'s package policy, however, turn out to be quite restrictive in practice. For example, *CRAN* imposes cross-platform compatibility, it does not allow packages to have dependencies outside of *CRAN*, it imposes packages to stay up-to-date with *CRAN*'s most current environment and with the latest version of R. This refrains, or even prohibits, certain package developers of getting their packages on *CRAN*:

- *"It can be a painful process, so you want to get your package in order before you submit."* [22]
- *"Even with our current policy of aiming for back-compatibility we get a lot of complaints that we are asking too much."* [23]
- *"The non-transparent nature of the CRAN submission / rejection process is particularly at issue."* [24]

Moreover, the argument of getting a significantly increased visibility when distributing one's package on *CRAN* is questionable [22]: *"It used to be that putting your package on CRAN also gave it some exposure, but with >6000 packages, that's no longer quite true. To get the word out about your package, I'd recommend twitter, writing a blog, or writing a paper [...]."* As such, there is no longer a strict need to rely on *CRAN* as the official platform for distributing R packages.

Distributing software through *GitHub* is commonplace. For example, numerous Javascript and CSS packages are, sometimes exclusively, distributed on *GitHub* and can be installed with dedicated tools like NPM and Bower. One of the big advantages of distributing a package on *GitHub* is that it allows developers to make use of, and to focus on, a single platform for both package development and package distribution. Moreover, *GitHub* integrates other useful services, e.g., bug tracking, feature requests, task management, wikis, etc.

GitHub is also becoming an important and integral part of the R package ecosystem. Even maintainers of established *CRAN* packages are use *GitHub* as a development platform: *"I do have some [packages] on CRAN that are not on GitHub but that's because I did not get to it yet. I plan to have all of my CRAN packages in GitHub."* [21]

Due to *GitHub*'s primary purpose, it is mainly being used for *developing* R packages, facilitated by dedicated tools. In addition to this, Hadley Wickham's devtools package makes it easy to *install* R packages from *GitHub* and other forges, through various functions to download and install R packages. For example, the function install_github allows R packages to be installed directly from *GitHub*. By default, the latest package version will be installed, but optional parameters can be used to install a specific version. Another useful tool is Dirk EddelBuettel's drat package, which can be used to create one's own repository of R packages stored in *GitHub*. Using these tools, R packages could also use *GitHub* as a *distribution platform*.

To analyze the extent to which *GitHub* is used as a package distribution platform, we intend to show that *GitHub* is becoming more and more important, and that despite the fact *GitHub* packages distributed on *CRAN* are generally older than other *GitHub* packages, there are numerous packages including instructions to install them from *GitHub*. In order to achieve this goal we study the following subquestions:

- *RQ*1*a: How important has* GitHub *become for R packages?* We provide evidence that the number of new R packages on *GitHub* is growing faster than the number of new R packages on *CRAN*.
- *RQ*1*b: How old are* GitHub *R packages distributed on* CRAN*?* We provide evidence that *GitHub* packages that are not distributed on *CRAN* are younger than those distributed on *CRAN* and form a distinct population. We also show that the age of a package cannot be used as a major discriminent in a model to predict the distribution of a package.
- *RQ*1*c: Which* GitHub *R packages are distributed on* GitHub*?* We show that many *GitHub* packages contain instructions to install them from *GitHub* and that thus many of them are expected to be distributed on *GitHub*.

*RQ*1*a: How important has* GitHub *become for R packages?*

According to githut.info, in the last quarter of 2014, R was the 12th most represented language on *GitHub* (in terms of number of active repositories). This is a major increase w.r.t. the last quarter of 2013, when R was only ranked 22nd.

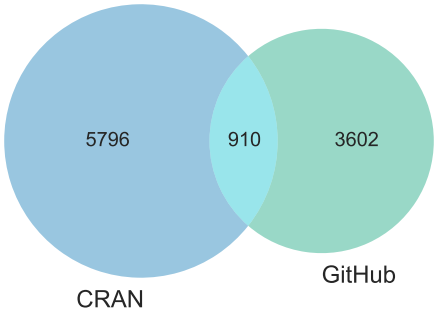

Figure 2. Number of R packages by source

Fig. 2 shows the number of *GitHub* and *CRAN* packages on June 1, 2015. At this date, ***CRAN* still hosts more packages than *GitHub*.** There are 910 packages belonging to both package repositories. This represents 14.0% of all *CRAN* packages that are also available on *GitHub*, and 20.2% of all *GitHub* packages that are also distributed on *CRAN*. This large overlap can be explained by the fact that both repositories still serve different purposes for a part of the R packages. One can expect that those R packages are developed on *GitHub*, while their stable releases are published on *CRAN*. This is for example the case for Amelia, ggplot2, dplyr, ... Our interviews with R package maintainers that were active on both *GitHub* and *CRAN* confirmed this way of working.

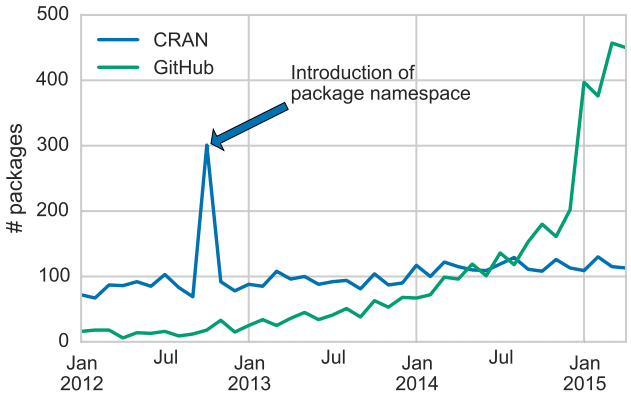

Figure 3. Number of new R packages, by month

Fig. 3 shows the number of newly created R packages, by month, on each platform. For *CRAN*, we see a more or less stable trend for the monthly number of new packages, with an exception of a big peak in the second half of 2012, due to the introduction of package namespaces in R. In contrast, for *GitHub* we see an increasing trend in the monthly number of new packages. Even more, since July 2014 **the number of new R packages on *GitHub* appears to be surpassing those on *CRAN*.** Since early 2015, the number of newly created packages is even more than three times higher on *GitHub* than on *CRAN*.

> **Summary.** *GitHub* already hosts many R packages, and there is an important acceleration of the number of new packages appearing on *GitHub* each month.

*RQ*1*b: How old are* GitHub *R packages distributed on* CRAN*?*

It is difficult to identify if a *GitHub* package is still in its development stage or if it is ready to be distributed.

We expect *GitHub* to be used as a development platform and *CRAN* as a distribution platform, so many packages will only end up in *CRAN* after some time, if they are considered to be sufficiently stable for being distributed and pass all necessary checks. This is, we expect that *GitHub* packages that are distributed on *CRAN* are older than *GitHub* packages that are not distributed on *CRAN*.

We define the *age* of a *GitHub* packages as the time between its very first version and the latest known commit. We compared the age of *GitHub* packages to see if this criterion can be used to distinguish packages that are already distributed on *CRAN* from those which are not.

Fig. 4 shows the distribution of the age of all 4,512 *GitHub* R packages, as well as the distribution of those 3,602

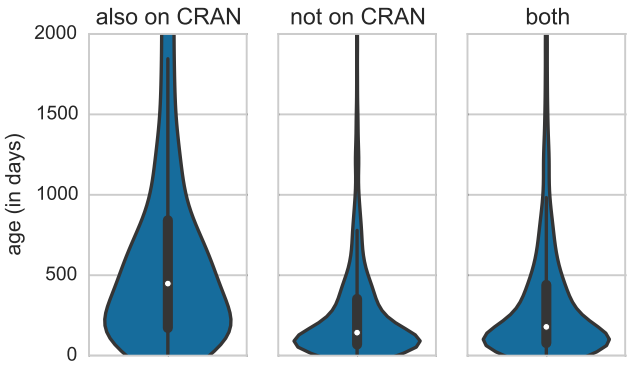

Figure 4. Violin plot (using a kernel density estimate) of the distribution of the age of *GitHub* packages.

R packages not found on *CRAN*, and the distribution of the 902 R packages also available on *CRAN*.

We statistically compared the age of the sample of *GitHub* packages that are distributed on *CRAN* with the age of the sample of those that are not. Since the samples were not normally distributed, we carried out a one-sided non-parametric Mann-Whitney-U test.

It is used to test the null hypothesis that the distribution of both populations are equal, or alternatively, whether observations in one population tend to be larger than observations in the other. We choose as alternative hypothesis that the population of *GitHub* packages distributed on *CRAN* is *older* than the population of *GitHub* packages that are not distributed on *CRAN*. The motivation behind this choice is quite logical: we expect *GitHub* to be used as a development platform and *CRAN* as a distribution platform, so many packages will only end up in *CRAN* after some time, if they are considered to be sufficiently stable for being distributed and pass all necessary checks.

As expected, the null hypothesis was rejected with significance level $\alpha = 0.01$. More specifically, we observed that **the majority of the packages (>75%) that are *not* distributed on *CRAN* are younger than the median of *GitHub* packages that are distributed on *CRAN*.** While the median age of *GitHub* packages distributed on *CRAN* is 448 days old, only 42.5% out of the 1,107 *GitHub* packages that are older than 448 days are actually distributed on *CRAN*. This is, the age of a package cannot be used as a major discriminant in a model to predict its distribution on *CRAN*.

Summary. *GitHub* packages distributed on *CRAN* are older than *GitHub* packages not distributed on *CRAN*, and constitute a distinct population. However, the age of a *GitHub* package cannot fully explain its distribution status.

RQ1c: Which GitHub *R packages are distributed on* GitHub*?*

We would like to study how many R packages are expected to be distributed and installed from *GitHub*. It is, however, difficult to identify if a *GitHub* package is still in its development stage or if it is ready for distribution. As far as we know, there is no sound and complete characterisation of when an R package is ready to be distributed on *GitHub*.

As an approximation, we looked for specific installation instructions from *GitHub* within all of the README files at the root of their *GitHub* repositories. We analyzed these README files using a regular expression corresponding to the use of the function install_github. Our results could include false positives (e.g., "this package cannot be installed using install_github(...)"). However, a manual verification of our results for many packages did not reveal such false positives.

We are also aware that our approach may be inaccurate, since R packages being distributed on *GitHub* may not necessarily have a README file that mentions install_github. Therefore, we can only compute a lower bound of the proportion of R packages that are distributed on *GitHub*. We obtained that **40.9% of all R packages on *GitHub* have a README file that contains instructions to install the package from *GitHub***. These packages correspond to 44.9% of the *GitHub* packages that are also on *CRAN*, and to 39.9% of the *GitHub* packages that are not on *CRAN*.

Summary. Many *GitHub* packages are intended to be distributed on *GitHub* as they contain instructions to install them from *GitHub*.

Discussion

The success of *GitHub* as a *development platform* for software packages seems obvious. It is more difficult, however, to quantify the use of *GitHub* as a *distribution platform*. Many package developers are already resorting to *GitHub* to distribute their software or libraries using tools like NPM or Bower. In addition to these tools, many packages can be downloaded directly from *GitHub* without having to rely on a package manager.

For R packages in particular, however, there is no commonly accepted package manager for *GitHub*. There are neither automatic processes nor absolute assertions than can be used to identify which of the R packages on *GitHub* are intended to be distributed.

Nevertheless, the interviews with R package maintainers confirmed that they actively use *GitHub* for distributing packages, for a variety of reasons. For example, some R package maintainers decide to host their packages on *GitHub* rather than *CRAN*, because their packages depend on external packages that are not accepted by *CRAN*. This is the case for the ANTsR package (stnava.github.io/ANTsR/) that

depends on cmake, as well as some packages that depend on commercial packages not available in *CRAN*.

From a quantitative point of view, we found many R packages that are distributed on *GitHub*, providing specific installation instructions. We also found that packages that are only available on *GitHub* are younger than the *GitHub* packages that are also distributed on *CRAN*.

Conclusion. R package developers are using *GitHub* as a *distribution platform* for R packages.

V. *RQ*2: To which extent do R packages suffer from inter-repository dependency problems?

The R package devtools provides a wrapper around R's built-in installation manager to facilitate the installation of R packages from different sources. Unfortunately, repositories such as *GitHub* have no central listing of available R packages. *GitHub* contains millions of *Git* projects filled with content from various programming languages. Even if we limit *GitHub* projects to those tagged with the R language, the vast majority does not contain an R package.

The lack of a central listing of packages prevents devtools to automatically find and install dependencies. Additionally, the same package can be hosted on multiple repositories and in different versions, making the problem of dependency resolution even more difficult.

To analyze the extent to which R packages hosted on *GitHub* are subject to inter-repository dependency problems we show that, while *CRAN* is the main source of packages to depend upon, the *CRAN* packages required by *GitHub* packages are the ones that are updated the most frequently. It shows that *GitHub* packages suffer from inter-repository problems because there are many packages with error status on *CRAN* caused by backward incompatible changes in the dependencies. In order to achieve this goal we study the following subquestions:

- *RQ2a: In which repository are package dependencies satisfied?* We provide evidence that, while *CRAN* is the main source of packages to depend upon, many *GitHub* packages have dependencies to packages not provided by *CRAN*.
- *RQ2b: Are* CRAN *packages more frequently updated than* GitHub *packages?* We provide evidence that *CRAN* packages required by *GitHub* packages are more prone to be updated than *CRAN* packages not required by *GitHub* packages.
- *RQ2c: How often do package updates cause backward incompatible changes?* We provide evidence that many *CRAN* package become broken due to *CRAN* packages updates. We show how this problem is addressed on *CRAN* and maintainers of *GitHub* packages do not benefit from *CRAN*'s solution.

RQ2a: In which repository are package dependencies satisfied?

We determined in which repository the dependencies of R packages hosted in *CRAN* or *GitHub* are satisfied. The results are summarised in Fig. 5. For each edge from repository A to repository B, the blue label on top represents the percentage of R packages from A that have a dependency satisfied by B, and the red label below the edge represents the percentage of dependencies of R packages in A that are satisfied by B. For example, 70.6% of all R packages on *GitHub* depend on at least one package in *CRAN*, while 85.3% of all declared dependencies in R packages on *GitHub* are satisfied by *CRAN* packages. Note that, if a package is hosted by both *CRAN* and *GitHub*, we count it as a *CRAN* package, because for dependency satisfaction we privilege the officially distributed package over its development version.

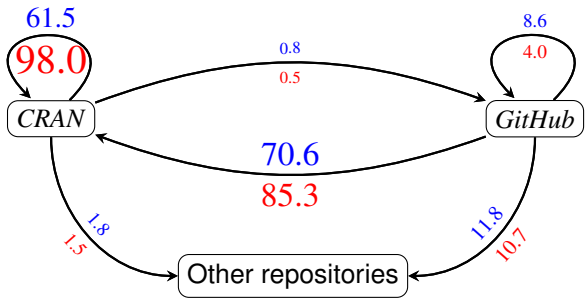

Figure 5. Inter-repository package dependencies for *CRAN* and *GitHub*. In blue: percentage of packages. In red: percentage of dependencies.

Because of the constraints imposed by *CRAN*'s daily R CMD check, *CRAN* is self-contained. 98.0% of its dependencies are satisfied within *CRAN*. Another 1.5% is satisfied by *BioConductor* and *Omegahat*, which are package distributions that are taken into account by the R CMD check.

The presence of 0.8% *CRAN* packages that depend on *GitHub* packages (totalling 0.5% of *CRAN*'s package dependencies) may seem to contradict this assertion, as the R CMD check does not allow the installation of packages from *GitHub*. However, these 0.8% dependencies target packages that are distributed on *BioConductor* or *Omegahat*, and for which a development version is available on *GitHub*.

This implies that nearly all *CRAN* packages satisfy their dependencies within *CRAN*. In addition, 70.6% of the R packages on *GitHub* depend on a package belonging to *CRAN* (totalling 85.3% of *GitHub*'s package dependencies). This strongly suggests that ***CRAN* is at the center of the ecosystem**, and that it is **nearly impossible to install R packages hosted on *GitHub* without relying on *CRAN*.**

Considering R packages on *GitHub*, 8.6% of them have a dependency in *GitHub* and 11.8% have a dependency that is not in *CRAN* or *GitHub*. The union of these R

packages on *GitHub* represents 767 *GitHub* packages that have a dependency not available on *CRAN*, and thus **767 *GitHub* packages that cannot be automatically installed currently.** This represents 17.0% (767 out of 4,512) of all *GitHub* packages, and 20.8% (748 out of 3,602) of all exclusive *GitHub* packages (that have no counterpart on *CRAN*).

Because of *CRAN*'s historically central position, its longevity and its important size in terms of number of packages, contributors might choose to develop packages depending only on *CRAN* packages. However, this does not appear to be the case. Indeed, we found that **there are three times more *GitHub* packages requiring a package that is not available on *CRAN* in June 2015 than in June 2014**.

Summary. A majority of *GitHub* packages needs to rely on *CRAN* to have their dependencies satisfied, and a non-negligeable proportion of *GitHub* packages cannot currently be installed automatically due to the lack of a central listing of available packages.

RQ2b: Are CRAN *packages more frequently updated than* GitHub *packages?*

Because *CRAN* is self-contained, its packages do not suffer from inter-repository dependency problems. These packages may, however, be affected by updates of the packages on which they depend. The solution imposed by *CRAN* to deal with package dependency updates is a continuous integration process based on the R CMD check tool. All *CRAN* packages are tested daily on different operating systems by running over 50 individual checks for common problems in the package structure, metadata, documentation, data, code, etc. If the check discovers an error in a *CRAN* package, its maintainer is notified and asked to resolve the problems before the next major R release. Failing to resolve the error will result in archiving the problematic package. As such, as it will no longer be included in the *CRAN* release until a new version is provided that passes the *CRAN* check. Moreover, all packages depending on it will fail the automatic installation. The above process requires *CRAN* package maintainers to quickly react to backward incompatible changes, and to adapt their packages to avoid obsolescence. This way, *CRAN* tries to ensure that all unarchived packages are mutually consistent.

Although devtools provides a function, called install_version to manually install a specific version of a *CRAN* package, it can be tedious to install such a version since one also needs to rely on a compatible version of its dependent packages. By default, only the latest available version of each dependency is automatically installed. Also, while maintainers could specify constraints on the required version of dependent packages, they rarely do it because those constraints are silently ignored by the R installer anyway.

As package updates may introduce backward incompatible changes and as there is no built-in automated support in R for specifying, retrieving and installing a specific version of a package dependency, a **package dependency update requires the depending package to be updated as well to avoid it becoming obsolete.** As *GitHub* packages are not concerned by the systematic integration process of *CRAN*, *CRAN* packages updates are more problematic for *GitHub*, exhibiting a typical case of the inter-repository dependency update problem.

Fig. 5 showed that 70.6% of all *GitHub* packages depend on a package belonging to *CRAN*. So how likely is it that a *CRAN* package gets updated? And does this differ between packages that are required by other *CRAN* packages and packages that are required by *GitHub* packages?

To respond to these questions, we use the statistical technique of *survival analysis* to estimate the probability of not updating a *CRAN* package for a certain amount of time. Survival analysis [25] creates a model estimating the survival rate of a population over time, considering the fact that some elements of the population may leave the study, and that for some other elements the event of interest does not occur during the observation period. In our case, the observed event is the moment on which an R package gets updated. The survival curve is shown in Fig. 6 using a Kaplan-Meier estimator.

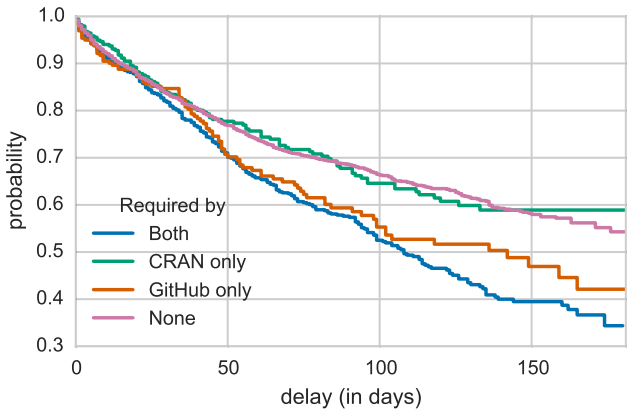

Figure 6. Probability that a *CRAN* package is not updated

For this survival analysis we considered all *CRAN* package updates over a six month period (from December 2014 to June 2015), representing a total population of 3,740 individuals. We observe that, starting from 36 days, those packages that are at least required by *GitHub* packages are more likely to be updated than the others. We confirmed this hypothesis using a one-sided non-parametric Mantel-Cox statistical test with significance level $\alpha = 0.05$. This test compares whether the generation process of observed events of the two populations are equal.

The null hypothesis states that both samples have identical

survival and hazard functions. The null hypothesis was rejected with p-value $= 0.03$ when comparing "*GitHub* only" and "*CRAN* only" populations, and was rejected with p-value < 0.001 when comparing "Both" and "*CRAN* only" populations.

This provides statistical evidence that the population of *CRAN* packages that are at least required by *GitHub* packages is significantly more prone to be updated than the population of *CRAN* packages that are only required by *CRAN* packages.

> **Summary.** *CRAN* packages that are required by some *GitHub* packages are more prone to be updated than other *CRAN* packages.

RQ2c: How often do package updates cause backward incompatible changes?

Package dependencies may be problematic in practice, since package updates may have undesired consequences on dependent packages. We will refer to this as backward incompatible changes.

For *CRAN*, based on the daily results of the R CMD check over a two-year period, we found that **41% of the errors in *CRAN* packages are caused by backward incompatible changes** in one of its dependencies [6]. For each update of a required package, we analyzed the status of all its dependent packages to identify if the update leads to an error status and thus, includes a backward incompatible change. We retrieved the status of the R CMD check from December 2014 to June 2015, for all *CRAN* packages that have dependent packages and for all of their dependent packages.

It would be desirable to report similar findings for *GitHub* packages depending on *CRAN* packages. Unfortunately, there are no historical R CMD check results for R packages on *GitHub*. In the future, this could perhaps be solved by building upon infrastructures such as R-Hub (github.com/r-hub), a recent proposal accepted by the R Consortium, that aims to automate and facilitate the checking or R packages that are not yet in *CRAN*.

For *CRAN*, Fig. 7 shows the total number of package updates and the number of package updates that lead to backward incompatible changes. These numbers are aggregated by two-week periods. The dashed red line represents the proportion of backward incompatible updates relative to the number of package updates. Over a total of 1,710 required packages, 643 packages (37.6%) were updated at least once, for a total of 1,029 updates. We identified 46 packages (7.2%) for which 59 updates (5.7%) caused a total of 84 errors in 60 different dependent packages. **On average, this represents one backward incompatible change per 20 updates.**

While these numbers are already relatively high, they only represent an underapproximation of the actual number

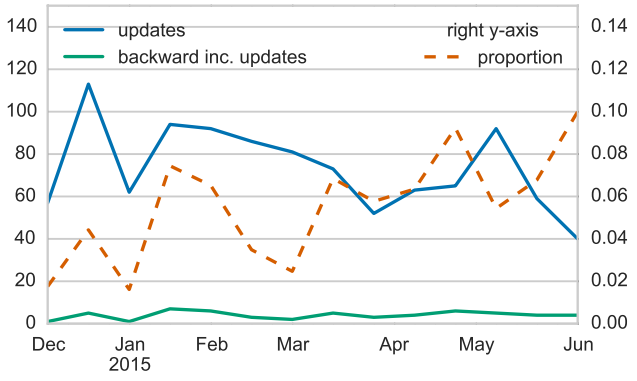

Figure 7. Number (left axis) and proportion (right axis) of updates for required *CRAN* packages

of updates that include a backward incompatible change. We assumed that a package update contains a backward incompatible change if the R CMD check for a package that depends on it results in an error on the same day. This implies that we are strongly dependent on the quality and on the coverage of the tests executed by the R CMD check.

The *CRAN* community has identified dependency updates as a concern that needs to be addressed:

- *"One recent example was the forced roll-back of the ggplot2 update to version 0.9.0, because the introduced changes caused several other packages to break."* [3]
- *"It is more and more of a pain if the package I'm depending on breaks. If it is just something I was doing for fun, it's not that big of a deal. But if it means I have to rewrite/recheck/rerelease my R package than that is a much bigger headache."* [4]
- *"[...] If I have to load a Depends package, it adds a significant burden: I have to check for conflicts every time I take a dependency on a new package. With Imports, the package is free of side-effects [...]"* [26]

All interviewed R package maintainers active on *GitHub* share this concern:

- *"[...] the risk of things breaking at some point due to the fact that a version of a dependency has changed without you knowing about it is immense. That actually cost us weeks and months in a couple of professional projects I was part of."* [21]
- *"A better systematic for dependency management together with making your codebase more robust against changes in dependencies is the thing that I actually spend most of my time cracking my brain about [...]"* [21]
- *"I had one case where my package heavily depended on another package and after a while that package was removed from* CRAN *and stopped being maintained. So I had to remove one of the main features of my package.*

Now I try to minimize dependencies on packages that are not maintained by "established" maintainers or by me [...]" [21]

- *"There have actually been a few times when I have rewritten a function in my own package because of that difficulty, especially with packages that themselves have many dependencies. The biggest issue we have is multiple layers of dependencies, some of which are on* CRAN *and some of which are not. That can be difficult to keep in sync, but usually, if your package is not on* CRAN*, you can just keep it using the older dependency for a while until you have time to sort that issue."* [21]
- *"It's a bit of a hassle when your package depends on other development versions, but there are changes in the latest version of* devtools *to make this easier."* [21]

Summary. Many dependent R packages become broken due to backward incompatible package updates.

Discussion

Since more and more packages are being developed and distributed outside of *CRAN*, we argue that inter-repository dependency management will become a major concern for the R community.

R package users and developers could benefit from a package installation manager that relies on a central listing of available packages. It is definitely feasible to achieve such a tool, since popular package managers for other languages such as JavaScript (e.g., bower and npm) and Python (e.g., pip) also offer a central listing of packages, facilitating their distribution through several repositories including *GitHub*.

To reduce the problem of backward incompatible changes in R packages hosted on *GitHub* due to *CRAN* package updates, package maintainers could deploy continuous integration processes (such as Travis CI) on their *GitHub* projects, in order to benefit from an equivalent of *CRAN*'s daily R CMD check. However, we observed that only 23.6% of the considered R packages hosted on *GitHub* have defined a configuration for Travis CI. Even with such a continuous integration process, the lack of a built-in support for dependency constraints satisfaction in R still forces developers to react to all the backward incompatible changes in each dependency of their package, even if the dependent package is stable or no more under development.

R package maintainers active on *GitHub* confirm the need for more systematic package dependency management: *"I personally think it's REALLY relevant to at least be ABLE to be very specific and rigid with regard to your dependencies. And I think the R universe could provide better tools to fit the needs of developers and professionals out there in a better way. But in that regard I like efforts such as* Packrat *and* checkpoint *very very much."* [21]

People from the R community have started to explore ways to improve how *CRAN* and the current R package management system work together [3]. Inspired by the way in which Debian Linux and npm manage their package distributions, two solutions are proposed. The first solution consist of having a testing and a development branch of the *CRAN* distribution. The development distribution contains the most recent but also more unstable packages, while the testing distribution is regularly frozen in order to release a stable snapshot of *CRAN* with each new version of R . While this solution would certainly benefit *CRAN*, it does not solve the problems for R packages hosted on *GitHub*. R package maintainers are already adopting such a solution, but at an individual level: *"Yes. I usually have a* cran *branch which matches the* CRAN *version. A* master *branch (or one that I set as default) would be the current development version. I might have some experimental branches as well."* [21] Having standardised support for this would benefit the community as a whole.

A second, more general, solution would consist in fundamental changes to the way in which R installs and loads packages: each package should be allowed to specify the version of its dependencies it requires, and provide a way to install and load multiple versions of the a package at the same time.

Conclusion. Better built-in support for dependency constraints satisfaction in R would turn out to be a major improvement, both for *CRAN* and *GitHub* R communities.

VI. Threats to Validity

Our analysis relied on information extracted from Git and *GitHub*. Many pitfalls should be taken into consideration when doing so [27], [28]. Some of them can be avoided, others or inherent to the limitations of the considered version control systems or hosting services. For example, how should forking be taken into account? In our analysis, we excluded all forks. We based ourselves on the packages still existing in *GitHub* in June 2015. We were not able to extract historical information from *GitHub* repositories that have been removed before that date.

For R packages hosted on *GitHub*, we assumed that their *DESCRIPTION* file always resides in the root directory of each Git repository, because this is where functions like devtools::install_github expect packages to be located, and because this avoided inclusion of repositories containing R code but that are not R packages. It may, however, have lead to the exclusion of some R packages. We also found that some *GitHub* accounts containing R packages (in particular, accounts cran and rpkg) actually served as partial mirrors of *CRAN*, or as a mean to expose R code to *GitHub*. These accounts were excluded from our analysis, but we have no guarantee that other accounts may also be mirrors of R packages developed or distributed elsewhere.

It is not trivial to determine whether an R package available on *GitHub* is ready for distribution. For all identified R

packages we checked whether they were ready for release by verifying the presence of a README file with specific installation instructions. Although a manual verification did not reveal any false positives, there may have been false negatives that we have not considered.

With respect to RQ2, we know that the CMD check on many R packages on *CRAN* results in an error status without an update of any of their dependencies (for example, if the language R itself or one of the base R packages evolves). This could influence the results for packages with many dependencies, as there are potentially more situations in which at least one dependent package gets broken at the same time of an update of its dependency.

VII. Future Work

Our study of inter-repository dependency problems can be extended in many ways. We only considered the subset of R packages contained in *CRAN* and in *GitHub*, that together account for more than 10,000 distinct R software packages. We could extend our analysis to other, smaller R packages sources, like *BioConductor* or *R-Forge*.

While we have confirmed many of our findings by contacting a few R package maintainers active on *GitHub* through e-mail, these results are not necessarily representative for the entire community. We are therefore planning a more extensive survey to get better insight in the use of *GitHub* as a package distribution platform and its impact on the R ecosystem.

Since recently, a number of tools and solutions are being proposed to address some of the problems that the R community is encountering related to package management. Emerging examples are: packrat, miniCRAN, drat, checkpoint and the recently funded R-Hub project. In the future, it would be useful to investigate how the take-up of such tools is reshaping the R community and affecting the way in which R packages are being managed.

We also aim to extend our package analysis by taking into account social metadata. For example, we could use the package author information to carry out a socio-technical analysis of the R package ecosystem. Similarly, we could take into account other data sources pertaining to R package development, such as mailing lists, issue trackers, activity on Q&A websites such as Stack Overflow, download statistics, and many more. These additional data sources would allow us to answer a whole range of new questions such as the following. Do we observe a difference in author activity and collaboration depending on the considered forge (*CRAN* or *GitHub*)? How much overlap exists between the communities of *GitHub* and *CRAN* authors of R packages? It would also be interesting to find out how the increasing usage of *GitHub* as a distribution platform impacts legacy distribution platforms like *CRAN* or *BioConductor*, with respect to their activity and their contributors.

The approach followed in this article could be generalised easily to other software ecosystems. However, we believe that some results are specific to the R package ecosystem due to the centrality of *CRAN* and its stringent requirements for package acceptance. It would be interesting to study other ecosystems (e.g. *Perl* modules available on *CPAN* and *GitHub*), and compare the differences and similarities with the R ecosystem.

VIII. Conclusion

In this article we explored to software ecosystem of R packages, focusing on the problem of inter-repository package dependencies in particular. Driven by its increasing popularity, we empirically studied the use *GitHub* as an alternative or complement to *CRAN* for R package development and distribution. We confirmed our observations through e-mail interviews with five R package maintainers that are active on *GitHub*.

We observed that more and more R packages are hosted on *GitHub*. While the *GitHub* packages distributed on *CRAN* tend to be older than those that are not, their age cannot fully explain whether they are distributed through *CRAN*. Additionally, many R package developers make use of *GitHub* as a distribution platform. Their packages contain instructions to be installed from *GitHub*, and are often exclusively distributed through *GitHub*.

In order to be able to install an R package, its dependent packages need to be available as well. While R provides an easy built-in way to install packages, this solution, in combination with the way in which *CRAN* distributes packages, is neither sufficient nor satisfactory to handle *GitHub* package dependencies [3]. Through interviews with R package maintainers on *GitHub*, we observed that the lack of support for dependency constraints in R is already a major concern. Contrarily to *CRAN*, distributed *GitHub* packages are not systematically monitored by a continuous integration process like the R CMD check in *CRAN*.

We showed that R packages hosted on *GitHub* suffer from inter-repository dependency problems. While most of the dependencies are provided by *CRAN*, many *GitHub* packages cannot be installed automatically due to a lack of central listing of all available packages. We also showed that *CRAN* package updates cause backward incompatible changes. Because *CRAN* packages that are required by *GitHub* packages are more prone to be updated, this problem is potentially worse for *GitHub* packages. We showed how this problem is currently addressed on *CRAN*, and how it affects *GitHub* packages.

To conclude, the R package-based software ecosystem would strongly benefit from an automatic package installation and dependency management tool, like the ones that are currently available for other package-based software ecosystems. The need for such automated tools for the R ecosystems was relatively low a few years ago, since *CRAN*

was, together with *BioConductor*, the major distribution platform for R packages. Today, this is no longer the case, since development platforms play an increasingly important role in the distribution of R packages.

ACKNOWLEDGMENTS

This research was partially funded by research project AUWB-12/17-UMONS-3 "Ecological Studies of Open Source Software Ecosystems" financed by the Ministère de la Communauté française - Direction générale de l'Enseignement non obligatoire et de la Recherche scientifique; as well as by research credit J.0023.16 "Analysis of Software Project Survival" financed by the F.R.S.-FNRS, Belgium.

We express our gratitude to Alexander Serebrenik for his very insightful comments on a draft version of this article. We thank Janko Thyson, Hana Sevcikova, Jessica Thompson, Hadley Wickham and Karl Broman for answering to some of our questions through e-mail.

REFERENCES

[1] R. A. Muenchen, "The popularity of data analysis software," r4stats.com, Tech. Rep., 2015, last consulted on 8 April 2015. [Online]. Available: http://r4stats.com/articles/popularity/

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

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

[4] J. Leek, "How i decide when to trust an R package," http://simplystatistics.org/?p=4409, November 2015.

[5] A. Decan, T. Mens, M. Claes, N. Tabout, and P. Grosjean, "On the development and distribution of R packages: An empirical analysis of the R ecosystem," in *Int'l Workshop on Software Ecosystems (IWSECO 2015). Co-located with ECSAW '15*. ACM , 2015.

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

[7] ——, "maintaineR: A web-based dashboard for maintainers of CRAN packages," in *Int'l Conf. Software Maintenance and Evolution*, 2014.

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

[9] L. A. Dabbish, H. C. Stuart, J. Tsay, and J. D. Herbsleb, "Social coding in GitHub: transparency and collaboration in an open software repository," in *Int'l Conf. Computer Supported Cooperative Work*, 2012, pp. 1277–1286.

[10] K. Blincoe, F. Harrison, and D. Damian, "Ecosystems in GitHub and a method for ecosystem identification using reference coupling," in *12th Working Conference on Mining Software Repositories*, ser. MSR, 2015, pp. 202–211.

[11] B. Vasilescu, V. Filkov, and A. Serebrenik, "StackOverflow and GitHub: Associations between software development and crowdsourced knowledge," in *SocialCom*. IEEE, 2013, pp. 188–195.

[12] B. Vasilescu, S. Van Schuylenburg, J. Wulms, A. Serebrenik, and M. van den Brand, "Continuous integration in a social-coding world: Empirical evidence from GitHub," in *Int'l Conf. Software Maintenance and Evolution*, Sept 2014, pp. 401–405.

[13] F. Thung, T. F. Bissyandé, D. Lo, and L. Jiang, "Network structure of social coding in GitHub," in *CSMR*, 2013, pp. 323–326.

[14] Y. Yu, G. Yin, H. Wang, and T. Wang, "Exploring the patterns of social behavior in GitHub," in *Int'l Workshop on Crowd-based Software Development Methods and Technologies*, 2014, pp. 31–36.

[15] F. W. Santana and C. M. L. Werner, "Towards the analysis of software projects dependencies: An exploratory visual study of software ecosystems," in *Int'l Workshop on Software Ecosystems (IWSECO)*, ser. CEUR Workshop Proceedings, vol. 987. CEUR-WS.org, 2013, pp. 7–18.

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

[17] D. M. Germán, J. M. González-Barahona, and G. Robles, "A model to understand the building and running inter-dependencies of software," in *Working Conf. Reverse Engineering*, 2007, pp. 140–149.

[18] G. Bavota, G. Canfora, M. Di Penta, R. Oliveto, and S. Panichella, "The evolution of project inter-dependencies in a software ecosystem: the case of Apache," in *Int'l Conf. Software Maintenance*, 2013, pp. 280–289.

[19] P. Abate, R. D. Cosmo, R. Treinen, and S. Zacchiroli, "Dependency solving: A separate concern in component evolution management," *Journal of Systems and Software*, vol. 85, no. 10, pp. 2228–2240, 2012.

[20] ——, "A modular package manager architecture," *Information & Software Technology*, vol. 55, no. 2, pp. 459–474, 2013.

[21] T. Mens, "Anonymized e-mail interviews with R package maintainers active on CRAN and GitHub," University of Mons, Tech. Rep., 2016.

[22] K. Broman, "R package primer – a minimal tutorial," http://kbroman.org/pkg_primer/, 2015.

[23] R devel mailing list, "R package dependency issues when namespace is not attached," http://r.789695.n4.nabble.com/R-tt4629828.html, 2015.

[24] GitHub ANTsR issue 8, "ANTsR: towards CRAN & standardization of development," https://github.com/stnava/ANTsR/issues/8, 2015.

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

[26] StackOverflow, "Writing robust R code: namespaces, masking and using the '::' operator," http://stackoverflow.com/questions/10947159/writing-robust-r-code-namespaces-masking-and-using-the-operator, 2015.

[27] C. Bird, P. C. Rigby, E. T. Barr, D. J. Hamilton, D. M. Germán, and P. T. Devanbu, "The promises and perils of mining Git," in *Working Conf. Mining Software Repositories*. IEEE Computer Society, 2009, pp. 1–10.

[28] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. Germán, and D. Damian, "The promises and perils of mining GitHub," in *Working Conf. Mining Software Repositories*, 2014, pp. 92–101.

[29] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing In Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.