maintaineR: A web-based dashboard for maintainers of *CRAN* packages

Maëlick Claes, Tom Mens, Philippe Grosjean
Software Engineering Lab, COMPLEXYS Research Institute, University of Mons
Email: firstname.lastname@umons.ac.be

***Abstract*—The *R* development community maintains thousands of packages through its *CRAN* archive network. The growth and evolution of this archive makes it more and more difficult to maintain packages and their interdependencies, and the existing tools that aim to help developers in this process no longer suffice. We propose *maintaineR*, a web-based dashboard that allows *CRAN* package developers to understand and deal with the implications and problems raised by package updates. The dashboard complements existing tools, such as the *R* CMD check tool, by providing additional support such as the visualisation of package dependencies and reverse dependencies, package conflicts, cross-package function clones, and so on.**

I. Introduction

Many generic web-based dashboards exist to help developer communities with specific maintenance activities. For dedicated software developer communities involved a specific software ecosystem (including specific programming languages, development processes tools, guidelines, rules and hardware infrastructure), targeted web-based dashboards are however not always available, or need to be improved to accommodate the specific needs of developers.

The *maintaineR* dashboard focuses on the *R* project community.[1] *R* offers a free and open source language and environment for statistical computing and visualisation. Its community is composed of thousands of developers, involved in maintaining thousands of packages. The *CRAN* package repository[2], available since 1997, is the primary source of *R* packages. While other *R* package repositories are available (e.g., Bioconductor and R-forge), they are considerably smaller in size.

The number of available *CRAN* packages is growing very rapidly and has reached a size of over 5000 packages, which is considered by some as "too many" [1]. In addition, limitations of *R*'s dependency versioning system have been reported and possible directions for improvement (such as staged package distributions and versioned package management) have been proposed [2]. Another problem is a lack of coordination between maintainers of dependent packages. Packages may cease to function correctly because of unexpected changes made to the packages they rely upon.

Therefore, there is a need for more specific tools dedicated to R package developers, that allow them to gain insight in, and deal with, the implications and problems raised by package updates. Being able to address such problems *a priori* during package development and maintenance, i.e., long before submitting it to CRAN, will reduce the effort of maintaining contributed CRAN packages.

We have developed *maintaineR*, a web-based dashboard to alleviate the above problems. It is available for download at github.com/maelick/maintaineR, together with clear installation instructions and a screencast of the tool in action. *maintaineR* is more specific and fine-grained than what is currently available to *CRAN* maintainers. It helps them to identify and avoid problems that could break their own package or those of others *before* sending it to *CRAN*. The tool is based on a fine-grained function-level analysis of dependencies, conflicts and clones (copy-paste reuse of code) between packages.

II. Related Work and Existing Tools

maintaineR aims to help in understanding and supporting the maintenance of software *ecosystems* and their components. We adhere to the definition by Messerschmitt [3] who defines a software ecosystem as "a collection of software products that have some given degree of symbiotic relationships." Stated differently, a software ecosystem constitutes an evolving collection of software projects/components/packages that share a common goal, as well as human and technical resources.

Several tools have been proposed to analyse, understand and visualise software ecosystems and their evolution. For example, Neu et al. [4] developed *Complicity*, a web-based application aiming to support software ecosystem analysis through interactive visualisations. Perez et al. [5] presented *SECONDA*, a software ecosystem analysis dashboard. The goal of *maintaineR* is similar in spirit, namely to provide a web-based dashboard for analysing and understanding the *CRAN* ecosystem and the packages it contains.

Another type of software ecosystems are the so-called software distributions, which are collections of software components that are bundled together in such a way that they can be installed and used "as is" by an end-user for its intended purpose. From a maintenance point of view, this raises additional problems, since the maintainers of the software components need to avoid broken dependencies or other conflicts whenever they upgrade a component. Di Cosmo et al. have formally studied this problem of coinstallability and provided tool support for addressing it [6], [7]. *CRAN* may benefit from similar support. In particular, more fine-grained analysis of *CRAN* package dependencies and how these affect maintenance and update of packages is needed. Although some empirical studies of *CRAN* have been carried out [8], [9], we are not aware of any web-based applications aiming to provide sophisticated support for understanding and improving the maintenance of *CRAN* contributed packages.

[1] http://www.r-project.org
[2] http://cran.r-project.org

Of course, many generic web-based tools are available that provide insight in the evolution of software products by analysing historical data extracted from version control repositories using a combination of metrics, visualisation and statistics. Well-known examples of these are SonarQube™ that provides an extensible open source platform for managing code quality[3], and GitHub that includes a variety of views on the version control activity of ongoing projects, including network visualisations and historical visualisations. The main difference is that *maintaineR* offers dedicated support for *CRAN*, taking into account the specificities of the *R* language and the processes and tools used by *CRAN* package maintainers.

III. About CRAN check

CRAN follows a strict policy[4] that packages need to adhere to before being accepted on the repository[5]. *CRAN* package developers can use the *R CMD check* command at any time to detect possible problems in their contributed packages. This tool is also used to ensure conformance of accepted packages to the *CRAN* quality policy, and a daily check on the full set of packages is applied to detect which packages no longer pass the test. After certain package updates, other dependent packages may cease to work correctly if the functions they were relying on have changed. Packages can also fail the check if a new *R* release introduces, changes or removes features.

While packages are never removed from *CRAN*, they can eventually be archived. Maintainers have to ensure that their packages still pass the check. If this is not the case, maintainers have to fix the problems before the next non-minor release of R or to stick to a strict deadline short after this release. If they don't, their buggy packages will be archived. A major issue with *CRAN* is that it is only guaranteed to work properly with the last version of *R* and vice versa. Moreover, different versions of the same package cannot be installed together.

While the *R CMD check* is useful for *CRAN* maintainers to detect problems with their package, it does not give sufficient information about the origin of the failure. Therefore, *R* maintainers could benefit from a more specific tool using knowledge about all previous packages versions in order to identify causes of current errors and avoid future possible errors. This is what *maintaineR* aims to provide.

IV. Tool Presentation

This section presents *maintaineR*, the web-based dashboard for supporting *CRAN* package maintainers. In particular, the tool helps with analysing and visualising the implications and problems raised by package updates. The tool is still in a prototype phase, so its functionalities are likely to evolve in the future, based on feedback that we will receive from the community during the annual R User Conference.

Subsection IV-A presents *maintaineR*'s main architecture, the technologies that have been used for creating it, and the main functionalities. Subsection IV-B presents the historical analysis and visualisation provided. Subsection IV-C discusses the implemented support for analysing package dependencies.

[3] http://www.sonarqube.org
[4] http://cran.r-project.org/web/packages/policies.html
[5] http://cran.r-project.org/web/checks/

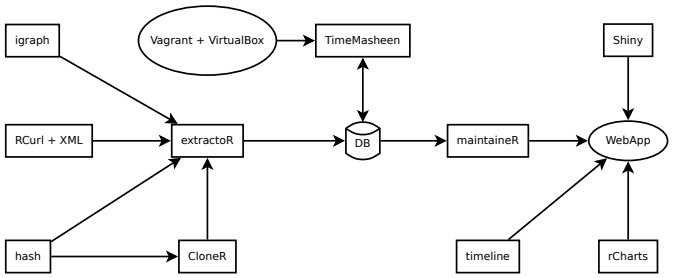

Fig. 1. Dependency graph of the components of the back-end and front-end of our tool. Rectangles represent R packages.

SubsectionIV-D explains how package conflicts are detected. Finally, subsection IV-E explains the support for detecting cross-package function clones.

A. Overall architecture

Since *maintaineR* targets *R* maintainers, its implementation relies mainly on *R* technologies and *CRAN* packages. The front-end is a web-based dashboard built using *Shiny*[6], a web application framework for *R*. For graph manipulations we used the package *igraph*. For graph visualization we used packages *rCharts* and *d3Network*, which both rely on the *d3.js* Javascript library. Packages *RCurl* and *XML* were used to fetch package metadata.

Fig. 1 presents the overall architecture of the tool, which is divided in two parts. The back-end (left part of figure) fetches, processes and writes data to an SQL database. The front-end consists of the *Shiny* web application and a command-line interface.

The back-end contains different components. Package metadata and content have been extracted with *extractoR*, a bundle of *R* packages we have developed previously for the purpose of empirically analysing *CRAN* [9]. We implemented a new package *CloneR* to find duplicate functions inside R code and we added an extension to *extractoR* to run *CloneR* on all the previously extracted *CRAN* packages.

To get information on conflicting functions between packages we used *Vagrant* and Oracle's *VirtualBox* to recreate virtual machines corresponding to a development environment closely approximating the situation at the time each package was released. *TimeMasheen* is the package responsible for initializing the virtual machines, feeding them with the packages that were available at the time of the release of the corresponding *R* version, loading them in an *R* process and finally writing the dynamic content of the loaded package in the database.

The web front-end is used to render and control data queried from the database. Executing the Shiny web application opens in a web browser the page http://127.0.0.1:3000, where 3000 is replaced by the port number on which Shiny runs. A list of all available *CRAN* packages appears, and selecting one of these will get the user to the package view. This view is divided in six tabs (as can be seen in Fig. 5):

- **Summary** shows a table containing the context of the package's *DESCRIPTION* file.

[6] http://shiny.rstudio.com

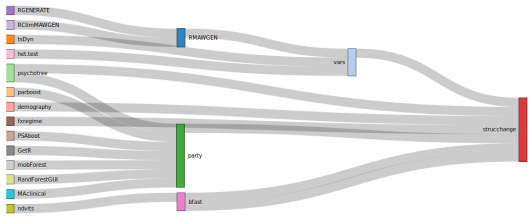

Fig. 3. Dependency view of the reverse dependencies for package `sctrucchange`, visualised as a Sankey diagram.

- **History** shows the release history of the package, its dependencies and/or its reverse dependencies.
- **Dependency list** shows the list of all dependencies and reverse dependencies.
- **Dependency graph** visualises the dependencies or the reverse dependencies of a package either as a graph or a Sankey diagram.
- **Namespace** shows all public objects declared in the package namespace and the potential conflicts with objects using the same name in other packages.
- **Clones** displays all functions that are present in all other *CRAN* packages.

B. Historical view

An example of the historical view is given in Fig. 2. By default, it shows on a timeline the release dates of the package. As shown in Fig. 2 it is possible to restrain the timeline to a shorter period and to add dependencies to and reverse dependencies from other packages. The ability to visualise globally the history of release dependencies is useful for a developer in order to spot recent changes when the package encounters an error during a *CRAN* web check.

C. Package dependency analysis

It is well-known that the presence of dependencies may cause problems during the evolution of component-based systems [6], [7]. We have empirically studied this problem for *CRAN* [9] and observed that package quality and maintainability varies with the operating system considered. We also observed that a non-negligible amount of errors are caused by dependency updates and need to be fixed by the maintainers. Maintenance effort hence needs to take into account changes made to package dependencies.

The *CRAN* policy recommends not to break reverse dependencies (i.e., packages that depend directly on it). However, this is only a recommendation and not a rule. Furthermore, while the *CRAN* website lists for each package all reverse dependencies, there is no tool to check that changes made to a package won't impact these other packages.

It is therefore important to offer package maintainers an easy way to visualise the direct and indirect dependencies and reserve dependencies of their packages. Fig. 3 shows such a visualisation generated by our tool for one of the *CRAN* packages.

Name	Type	Conflicts
abc	function	gvcm.cat_1.6 forams_2.0-4 pomp_0.49-2
cv4abc	function	None
cv4postpr	function	None
expected.deviance	function	None
postpr	function	None

Fig. 4. Namespace view for the *CRAN* package `abc`.

Reverse dependencies show the packages that depend on a given package and may help the package maintainer to minimise the ripple effect of any changes he desires to make. Ideally, a package update should not require updates or changes to packages that depend on it. Showing to the maintainer the dependencies and reverse dependencies of a given package, may help him find the causes of any errors introduced by package dependencies, as well as warn him about potential errors or conflicts introduced by this package in its reverse dependencies.

D. Detection of package function conflicts

R resolves function and variable names using environment objects, which are hash tables associating names to objects exported by the package namespace. When a variable or a function is referenced, the interpreter cycles through a list of environments. When two packages define the same function name, the last imported function will mask the first imported one. Although it is still possible to call the first function by specifying the package name with a special notation, this can lead to conflicts. For example, suppose that package *A* depends on packages *B* and *C* and uses a function *f* defined in *B*. If a new version of package *C* introduces a new function with the same name *f*, there is a chance that this creates a conflict.

Fig. 4 shows the **Namespace** view for a given *CRAN* package, displaying which function and variable names are exported by the package namespace, and which potential conflict this introduced with (particular versions of) other packages. To achieve this, we used *Vagrant* and *VirtualBox* to create a virtual machine running on Debian for each version of *R*. On all these virtual machines we installed all available *CRAN* packages at the time of the release of the machine's *R* version. We loaded all these packages one by one and we exported the list of objects available in the package environment.

E. Detection of function clones

As depending on another package can be costly from a maintenance perspective, it is common for *CRAN* package maintainers to copy-paste portions of code from other packages rather than depending upon them. Developers also frequently write local functions inside the body of other functions (like a closure). Often these functions do not reference any local variable and could therefore easily have been defined in the global scope of the package. Not doing so prevents dependent packages to reuse these functions.

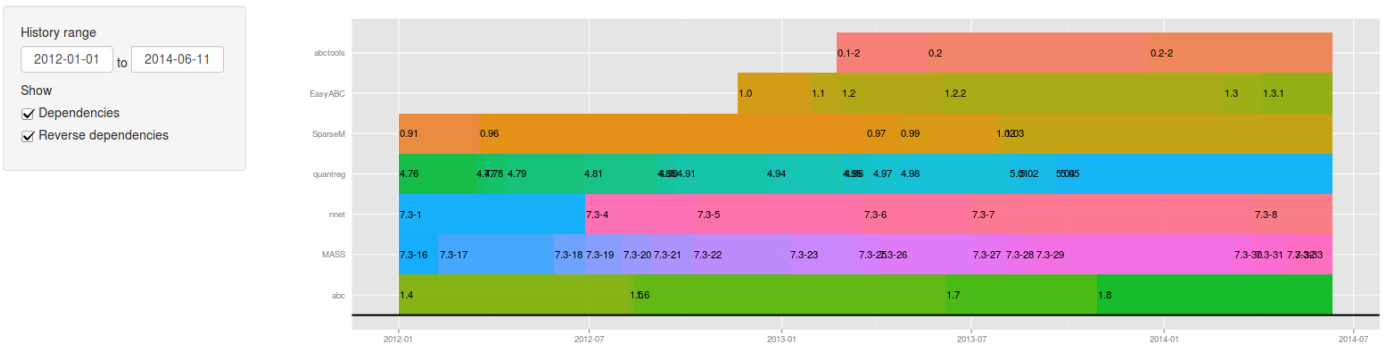

Fig. 2. Screenshot of the **History** view for releases of package `abc`, its dependencies and reverse dependencies.

Fig. 5. **Clones** view for the *CRAN* package `memisc`, filtered to show only clones of at least 3 lines and an AST of size at least 10.

The **Clones** view of our tool, illustrated in Fig. 5 determines which global or local functions defined in the package are perfect duplicates (so-called "Type 1" clones) of a function defined in another package. Clones are detected by parsing the *R* code of all packages, performing a depth-first traversal of the abstract syntax tree, computing a hash for each node, and adding all function definitions in a hash table using the computed hash key. All keys associated with more than one package correspond to cloned functions.

V. Conclusion and Future Work

We presented *maintaineR*, a web-based dashboard for analysing maintainability of *CRAN* packages, by offering analyses and visualisation of the package release history, package dependencies, potentially conflicting function names across packages, and identical function clones. We will actively promote take-up of our dashboard by the *CRAN* developer community, and we intend to improve the tool based on feedback received from *CRAN* package maintainers. We intend to use the tool ourselves for carrying out empirical analyses of the *CRAN* ecosystem.

Our tool can be extended in many ways. We will improve the current functionalities with new visualisations and more sophisticated analyses (for example, support for Type II and III clone detection). Next to the current package-oriented view, we aim to provide a maintainer-oriented view, as well as a global ecosystemic view of *CRAN*'s socio-technical ecosystem. Through our virtual-machine-based approach, we also aim to provide support for reproducibility of research studies that relied on older *CRAN* package versions.

Acknowledgments: The *maintaineR* tool was developed in the context of ARC research project AUWB-12/17-UMONS- 3.

References

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

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

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

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

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

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

[7] 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.

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