The R Quest: from Users to Developers

by Simon Urbanek

Abstract R is not a programming language, and this produces the inherent dichotomy between analytics and software engineering. With the emergence of data science, the opportunity exists to bridge this gap, especially through teaching practices.

Genesis: How did we get here?

The article "Software Engineering and R Programming: A Call to Action" summarizes the dichotomy between analytics and software engineering in the R ecosystem, provides examples where this leads to problems and proposes what we as R users can do to bridge the gap.

Data Analytic Language

The fundamental basis of the dichotomy is inherent in the evolution of S and R: they are not programming languages, but they ended up being mistaken for such. S was designed to be a *data analytic* language: to turn ideas into software quickly and faithfully, often used in "non-programming" style(Chambers, 1998). Its original goal was to enable the statisticians to apply code which was written in programming languages (at the time mostly FORTRAN) to analyze data quickly and interactively - for some suitable definition of "interactive" at the time(Becker, 1994). The success of S and then R can be traced to the ability to perform data analysis by applying existing tools to data in creative ways. A data analysis is a quest - at every step we learn more about the data which informs our decision about next steps. Whether it is an exploratory data analysis leveraging graphics or computing statistics or fitting models - the final goal is typically not known ahead of time, it is obtained by an iterative process of applying tools that we as analysts think may lead us further(Tukey, 1977). It is important to note that this is exactly the opposite of software engineering where there is a well-defined goal: a specification or desired outcome, which simply needs to be expressed in a way understandable to the computer.

Freedom for All

The second important design aspect rooted in the creativity required is the freedom the language provides. Given that the language can be computed upon means that a given expression may have different meaning depending on how the called function decides to treat it and such deviations are not entirely uncommon, typically referred to as non-standard evaluation. Probably the best example is the sub-language defined by the `data.table` package(Dowle and Srinivasan, 2021) featuring the `:=` operator which is parsed, but not even used by the R language.

Analogously, there is no specific, prescribed object system, but rather one is free to implement any idea desirable, as witnessed by the fact that there are more than a handful of object system definitions available in R and contributed packages. This freedom is what makes R great for experimentation with new ideas or concepts, but very hard to treat as a programming language.

We have a language that is built on the idea of applying tools and which allows freedom to express new ideas so the last important step is how to define new tools. R add-on packages(R Core Team, 2021) are the vehicle by which new tools can be defined and distributed to R users. Note that true to design goals, packages are not limited to R code but rather can also include code written in programming languages such as C, C++ or Fortran. That in turn makes it possible to write packages that expand the scope of tools to other languages such as Java(Urbanek, 2021) or Python(Ushey et al., 2022) simply by creating an R package which defines the interface.

Sharing Packages

But this is also where we are entering the realm of software engineering. Now we are in the business of *defining* the tools as opposed to just *using* the tools. It also means that the tools have to worry about programming interfaces, defining behavior and all those pesky things we as statisticians don't want to worry about. Although we originally started as R *users*, the moment we want to share any re-usable piece of code with others we are becoming *developers*. Since no developer would mistake R for a programming language, it is analysts with background in various fields which use statistics one way or another that are more likely to *use* R. However, as we become more comfortable with R, we

start using it as a programming language, not just analytic language, often because it is simply more convenient than having to learn a programming language. This explains the empirical evidence(Pinto et al., 2018) of R package authors not being trained software engineers, but often scientists from other fields and any consequences thereof.

However, as R packages started to emerge, it became clear that a loosely coupled structure is not enough and have to introduce software engineering concepts such as documentation and testing. R includes tools for automated checking for packages to be able to provide at least some basic guarantees. Packages provide examples which are supposed to be illustrative, but soon were used to perform limited testing. R itself is using the same package structure and it was clear early that a test suite is important and so was introduced. Consequently, the same facilities were available to packages, but only very few were using it. There are, however, no built-in tools for creating test suites. In core R those are hand-curated by experienced developers, but that does not scale to package space.

Over 18,000 packages are now present in the Comprehensive R Archive Network (CRAN), a repository which has arguably played major role in the success of R (Hornik and Leisch, 2002). This is not only a valuable resource for users, but today this rich collection of contributed R code in being used as an automated test-suite for R. This is no coincidence, the importance of software engineering concepts has been identified by the CRAN team long time ago and the tools in R have been enhanced for that purpose(Hornik, 2016). CRAN has been an invaluable asset for the development of R based on examples and limited tests alone. It allows us the R Core Team to test changes in R against code that was written by ingenious people that do not necessarily follow documentation, but instead write code that seems to work - possibly in ways not intended in the first place. Consequently, improving the quality and coverage of tests in packages has not only positive impact on the individual package, but on the quality of the entire CRAN ecosystem and R itself.

CRAN performs reverse-dependency checks where packages are not allowed to break dependent package which is an important software engineering concept. One can see CRAN as performing continuous integration and continuous testing if we consider all submitted packages as one big project. This is not universally liked among package authors, though. Some find it too tedious to be responsible for software in the way a software engineer would be - a concern which is also highlighted by the article.

Steal and Borrow

One perhaps surprising finding of the article was the analysis of code fragment re-use(Claes et al., 2015). A quite recent example how dangerous such practice is was a piece of badly written JavaScript code from Stack Overflow(StackOverflow) which was copied so often that it made it into the popular Unity game engine, effectively forcing browsers to lie about macOS versions(Chromium Bugs) just to not break millions of released products. R code fragments are less likely to have such world-wide impact, but can be equally frustrating. The historically relatively high cost of loading other packages was an incentive to simply copy fragments instead, but the performance impact has been diminishing with advancements in the R implementation. Still, I believe the exact reasons for fragment re-use deserve further examination and may reveal other, more benign motives.

Every Project Needs a Conductor

Another good example of introducing software engineering principles into the R world successfully is the Bioconductor project(Gentleman et al., 2004). The authors realized early that the project is too big for it to allow organic growth and have strongly encouraged the use of the S4 class system to build a class hierarchy specific to the tasks common to the Bioconductor packages. This enabled optimizations of implementation as a core part of the system as opposed to individual approaches in each package. Bioconductor was also encouraging unit tests and has maintained a build and reporting system similar to that of CRAN, in the early days even pioneering functionality that was later added to core R.

The Gospel of Data Science

I believe the Call to Action is a very timely contribution. Many R users start as statisticians or data analysts in some domain since that is the main strength of R. Consequently, a lot of R code is never publicly visible. Code written for data analyses is not software development and is not published as software. So any global statistics about R code have to be taken with that in mind. When considering R packages we are talking only about a fraction of the code written in R. However, building new tools is an important part of the R ecosystem and it has to be made clear that it is different from data analysis and thus requires different skills and tools.

The main realization here is that at some point an R user may become an R developer, crossing the line from analysis into software engineering. And we are often unprepared for that, in part because of our diverse background. When I asked my junior colleagues at the Labs what they find most challenging yet valuable, the top item was learning software engineering skills on the job. We were lucky to have both the authors of S as well as the authors of Unix on the same floor, so we were able to bridge the gap, but generally our schools don't prepare for that. That's why I believe we must teach statistical computing together with software engineering skills such as re-usability and testing concepts. The current popularity of data science which bridges both worlds is a good excuse to make it actually happen in practice.

Bibliography

R. A. Becker. A brief history of S. Technical report, AT&T Bell Laboratories, 11 1994. [p22]

J. M. Chambers. *Programming with Data: A Guide to the S Language*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 1998. ISBN 0387985034. [p22]

Chromium Bugs. Nearly all Unity WebGL games fail to run in Chrome on macos 11 because of userAgent. URL https://bugs.chromium.org/p/chromium/issues/detail?id=1171998. [p23]

M. Claes, T. Mens, N. Tabout, and P. Grosjean. An empirical study of identical function clones in CRAN. In *2015 IEEE 9th International Workshop on Software Clones (IWSC)*, pages 19–25, Mar. 2015. doi: 10.1109/IWSC.2015.7069885. [p23]

M. Dowle and A. Srinivasan. *data.table: Extension of 'data.frame'*, 2021. URL https://CRAN.R-project.org/package=data.table. R package version 1.14.2. [p22]

R. C. Gentleman, V. J. Carey, D. M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry, K. Hornik, T. Hothorn, W. Huber, S. Iacus, R. Irizarry, F. Leisch, C. Li, M. Maechler, A. J. Rossini, G. Sawitzki, C. Smith, G. Smyth, L. Tierney, J. Y. Yang, and J. Zhang. Bioconductor: open software development for computational biology and bioinformatics. *Genome Biology*, 5(10):R80, Sep 2004. ISSN 1474-760X. doi: 10.1186/gb-2004-5-10-r80. URL https://doi.org/10.1186/gb-2004-5-10-r80. [p23]

K. Hornik. Are there too many R packages? *Austrian Journal of Statistics*, 41(1):59–66, 2 2016. doi: 10.17713/ajs.v41i1.188. [p23]

K. Hornik and F. Leisch. Vienna and R: Love, marriage and the future. *Festschrift 50 Jahre Österreichische Statistische Gesellschaft*, pages 61–70, 01 2002. ISSN 2016-597X. [p23]

G. Pinto, I. Wiese, and L. F. Dias. How do scientists develop scientific software? An external replication. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 582–591, 2018. doi: 10.1109/SANER.2018.8330263. [p23]

R Core Team. *Writing R Extensions*. R Foundation for Statistical Computing, Vienna, Austria, 2021. URL https://cran.r-project.org/doc/manuals/r-release/R-exts.html. [p22]

StackOverflow. How to find the operating system details using JavaScript. URL https://stackoverflow.com/questions/9514179/how-to-find-the-operating-system-details-using-javascript. [p23]

J. W. Tukey. *Exploratory Data Analysis*. Number v. 2 in Addison-Wesley series in behavioral science. Addison-Wesley Publishing Company, 1977. ISBN 9780201076165. [p22]

S. Urbanek. *rJava: Low-Level R to Java Interface*, 2021. URL https://CRAN.R-project.org/package=rJava. R package version 1.0-6. [p22]

K. Ushey, J. Allaire, and Y. Tang. *reticulate: Interface to Python*, 2022. URL https://CRAN.R-project.org/package=reticulate. R package version 1.23. [p22]

Simon Urbanek
University of Auckland
Department of Statistics
Auckland, New Zealand
urbanek@R-project.org