12-2021

Rejoinder: Software Engineering and R Programming

Melina Vidoni

Rejoinder: Software Engineering and R Programming

by Melina Vidoni

Abstract It is a pleasure to take part in such fruitful discussion about the relationship between Software Engineering and R programming, and what could be gain by allowing each to look more closely at the other. Several discussants make valuable arguments that ought to be further discussed.

The Roles

It is worth arguing about the difference between **research software engineers** and **software engineering researchers**. While the former can be anyone developing scientific software for computation/data sciences (regardless of their technical background or "home" discipline), the latter are academics investigating software engineering in different domains.

Software engineering researchers aim to produce research that is translatable and usable by practitioners, and when investigating R programming (or any other type of scientific software) the "practitioners" are *research software engineers*. This distinction is relevant as one cannot work without the other. In other words, *software engineering researchers* ought to study *research software engineers* such like they study, e.g., a web developer, with the goal of uncovering their "pain points" and propose a solution to it. Likewise, *research software engineers* depend on *software engineering researchers* and expect them to produce the new knowledge they need.

However, what a *research software engineer* will vary by the programming language they use, and what they aim to achieve with it. In terms of R programming, as one discussant pointed, there can be a difference between an "R user" (which *uses* R to perform data analysis) and an "R developer" (which besides *using* the language, also *develops* it by creating publicly shared packages). However, to this extent, research has used both terms interchangeably, which leads to a possible avenue of work in terms of "human aspects of R programming".

The Software

This is where the next link appears–the **tools and packages** mentioned in the commentaries were developed with the intention of translating/migrating knowledge acquired/produced by *software engineering researchers* to the domain of R programming, and to be used by *research software engineers*. For example, the package `covr` streamlines the process of calculating the unit testing coverage of a package, and the original papers presenting such measures can be tracked down to the late "80s (Frankl and Weyuker, 1988; DeMillo, 1987). Albeit it is known coverage as a measure evolved and changed over time (and continues to do so), it is an excellent example of the outcome produced by *software engineering researchers* that successfully translated their findings to "practitioners" (in this case, *research software engineers*).

Therefore, a package is part of the "translation" of the knowledge acquired through software engineering research, into an accessible, usable framework. However, the tool itself is not enough–without the "environment" changing, growing, and learning, the tool may not be used to its full potential. Note that "environment" is used to refer (widely and loosely) to a person's programming habits, acceptance to change, past experiences (e.g., time/effort spent in solving a bug, or domains worked on), and even the people around them (e.g., doing/not doing something because of what others do/do not do) that influence their vision, attitude and expectations regarding programming.

Moreover tools and packages are not finite, static elements–because they are software, they evolve. And when the requirements of a community change, so must do so the tools. This act as a reminder to not assign a "silver bullet" status to a tool meant to solve a particular, static problem, when it has been known that software (and thus the practices to develop it) evolve, and may even become unmanageable, never to be fully solved (Brooks, 1987).

The Goal

Another related aspect is that "scientific software" has broader, different goals than "traditional" (namely, non-scientific) software development–it has been argued that "scientific software development" is concerned with knowledge acquisition rather than software production (Kelly, 2015); e.g. a "tool" can be an RMarkdown document that allows performing an analysis (hence, *using* the language).

Related to this, "scientific software" uses diverse paradigms, such as *literate programming* (which has been considered a programming paradigm for a few decades (Cordes and Brown, 1991)) and *scripting* (which in turn, continues to elicit mixed stances from *software engineering researchers* (Loui, 2008)) with goals different to "traditional software".

Thus, what "software engineering practices" mean for "scientific software" remains ambiguous, and some authors have argued that the "gap" between software engineering and scientific programming threatens the production of reliable scientific results (Storer, 2017). The following are some example questions meant to illustrate how these other aspects of "scientific software" may still be related to software engineering practices:

> *Could text in a literate programming file be considered documentation? Is scripting subjected to code-smell practices like incorrect naming or code reuse? Does self-admitted technical debt exists in literate/scripting programming? What is the usability of a literate programming document? Should analytical scripts be meant for reuse?*

The original article was intended to highlight some of the efforts made by *software engineering researchers* to bridge this gap of software engineering knowledge for "scientific programming". Nonetheless, *software engineering researchers* have perhaps focused more strongly on R packages because of their similarities to their current research (namely, "traditional software" development), thus making the translation of knowledge slightly more straightforward. Approaching other aspects, paradigms, tools and process of "scientific software" development still remains a gap on research that should be further studied.

The Community

The **community** is the next link in this chain–they motivate *software engineering researchers*" investigations, are the subjects, and the beneficiaries. Yet many times, they can also be the cause of their own "pain points". For example, research has shown that although StackOverflow is nowadays a staple for any programmer, many solutions derived from it can be outright insecure (Rahman et al., 2019; Fischer et al., 2017; Acar et al., 2016), have poor quality and code smells (Zhang et al., 2018; Meldrum et al., 2020), be outdated (Zhang, 2020; Zerouali et al., 2021), or have low performance (Toro, 2021), among others. This is but a facet of the concept of "there is no silver bullet" (Brooks, 1987), and the only way of solving such situation (partially, and temporarily) is to look at it from multiple points of views. This action is what the original paper aimed to highlight.

Final words

In the end, the differences between *software engineering researchers* and *research software engineers* are blurry, and the translation of concepts from "traditional software" development/research to "scientific software" development/research may not be as straightforward as both groups of stakeholders consider. However, for the R community to continue evolving, both can (and should) work together and learn from the other.

Bibliography

Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky. You get where you're looking for: The impact of information sources on code security. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 289–305, 2016. doi: 10.1109/SP.2016.25. [p26]

F. Brooks, Jr. No silver bullet essence and accidents of software engineering. *IEEE Computer*, 20:10–19, 04 1987. doi: 10.1109/MC.1987.1663532. [p25, 26]

D. Cordes and M. Brown. The literate-programming paradigm. *Computer*, 24(6):52–61, 1991. doi: 10.1109/2.86838. [p26]

R. A. DeMillo. *Software Testing and Evaluation*. Menlo Park, Calif Benjamin/Cummings Pub. Co, 1987. ISBN 978-0-8053-2535-5. [p25]

F. Fischer, K. Böttinger, H. Xiao, C. Stransky, Y. Acar, M. Backes, and S. Fahl. Stack overflow considered harmful? the impact of copy amp;paste on android application security. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 121–136, 2017. doi: 10.1109/SP.2017.31. [p26]

P. Frankl and E. Weyuker. An applicable family of data flow testing criteria. *IEEE Transactions on Software Engineering*, 14(10):1483–1498, 1988. doi: 10.1109/32.6194. [p25]

D. Kelly. Scientific software development viewed as knowledge acquisition. *Journal of Systems and Software*, 109(C): 50–61, nov 2015. ISSN 0164-1212. doi: 10.1016/j.jss.2015.07.027. URL https://doi.org/10.1016/j.jss.2015.07.027. [p25]

R. P. Loui. In praise of scripting: Real programming pragmatism. *Computer*, 41(7):22–26, 2008. doi: 10.1109/MC.2008.228. [p26]

S. Meldrum, S. A. Licorish, C. A. Owen, and B. T. R. Savarimuthu. Understanding stack overflow code quality: A recommendation of caution. *Science of Computer Programming*, 199:102516, 2020. ISSN 0167-6423. doi: https://doi.org/10.1016/j.scico.2020.102516. URL https://www.sciencedirect.com/science/article/pii/S0167642320301246. [p26]

A. Rahman, E. Farhana, and N. Imtiaz. Snakes in paradise?: Insecure python-related coding practices in stack overflow. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 200–204, 2019. doi: 10.1109/MSR.2019.00040. [p26]

T. Storer. Bridging the chasm: A survey of software engineering practice in scientific programming. *ACM Comput. Surv.*, 50(4), aug 2017. ISSN 0360-0300. doi: 10.1145/3084225. [p26]

M. L. Toro. Understanding the consistency of stack overflow code: A cautionary suggestion. *LC International Journal of STEM (ISSN: 2708-7123)*, 2(1):40–47, 2021. [p26]

A. Zerouali, C. Velázquez-Rodríguez, and C. De Roover. Identifying versions of libraries used in stack overflow code snippets. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pages 341–345, 2021. doi: 10.1109/MSR52588.2021.00046. [p26]

H. Zhang. *On the Maintenance of Crowdsourced Knowledge on Stack Overflow*. PhD thesis, Queen's University (Canada), 2020. [p26]

T. Zhang, G. Upadhyaya, A. Reinhardt, H. Rajan, and M. Kim. Are code examples on an online q amp;a forum reliable?: A study of api misuse on stack overflow. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 886–896, 2018. doi: 10.1145/3180155.3180260. [p26]

Melina Vidoni
Australian National University, School of Computing
Canberra, Australia
0000-0002-4099-1430
melina.vidoni@anu.edu.au