Scientific Life

Switching Software in Science: Motivations, Challenges, and Solutions

Jan R. Wessel,1,5,*
Krystof J. Gorgolewski,2,5,*
and Pierre Bellec3,4,5,*

A laboratory’s programming language has wide-ranging implications. As demands towards scientific programming change and languages evolve, investigators may look to change their existing software stack. Following up on a recent online debate, we discuss key considerations and challenges in choosing and changing languages and suggest solutions for investigators looking to switch.

Recently, J.R.W. was faced with a dilemma: he was considering switching his laboratory’s existing software infrastructure from MATLAB, a proprietary scientific programming language, to another, cost-free ecosystem. Looking for advice on some anticipated challenges, he publicly solicited input from the scientific community on a popular social media platform (https://twitter.com/Wessel_Lab/status/1031208646263418880). An intense conversation ensued, illustrating many of the challenges and complexities associated with switching between scientific software. After being approached by the editor of this journal to summarize that conversation, J.R.W. solicited help from K.J.G., the lead developer of several open-source scientific software packages, and P.B., an open-source contributor who recently transitioned his laboratory from MATLAB to Python. Together, they assembled a series of concrete questions and recommendations around three key considerations: how do you decide on the most appropriate programming language, what challenges are associated with switching, and what solutions exist to ease the transition (Figure 1)?

**Motivations**

What are the motivations that contribute to the decision on which software stack is most appropriate for your purposes? How do you evaluate whether your existing stack is still appropriate or whether you should consider switching to another language? Several factors play a role.

**Monetary Costs**

The first consideration is money. Are the tools you are planning to use free or require fees? Are those fees one time or subscription? Is your institution subsidizing those fees? The cost of the software you are using influences not only you, but also other scientists that might want to reuse your code. For example, if code works only in a commercial programming language (e.g., MATLAB, Mathematica), anyone who wants to reproduce a specific analysis will need to purchase a license.

**Scientific Fit and Community**

Specific languages have specific strengths that have to be considered. Are there libraries written specifically to solve problems related to your research? Is there a pool of open-source developers that can contribute to your project or help with technical questions? You may want to ask colleagues about the most useful libraries and the most active communities, or you may look to quantitative metrics, such as the frequency of recent commits or the number of active contributors on github.com projects, the number of active discussions on general-purpose sites like StackOverflow, and the amount of activity in community-specific help sites (e.g., Neurostars.org for neuroscience).

**Equipment Your Laboratory with Flexible Tools**

You may want to consider libraries beyond those directly relevant to your current research. Multidisciplinary science in particular calls for diverse and flexible tools. For example, you may have been focusing on memory meta-analysis research for most of your career, yet a new project may call for an analysis of sleep diaries using natural language processing tools. Does the library ecosystem of the programming language you have chosen support a broad range of potential new applications?

**Equipment Students with Transferable Skills**

The majority of students entering a PhD program will not end up running their own laboratories, or even working in academia [1], as the supply outpaces the demand [2]. Hence, the more useful the skills they learn in graduate school, the more appealing they will be on the wider job market. Familiarity with particular programming languages and libraries can lead to drastically increased employment opportunities. For example, many cognitive and neuroscience graduates transition into data scientist positions, where knowledge of Python, R, and SQL is more sought after than experience in MATLAB. Similarly, another popular alternative career – software engineering – is easier to switch to for students knowing JavaScript or Python. By contrast, MATLAB remains a popular choice for classical engineering disciplines. When choosing a software stack, think what transferable skills your trainees will gain from using it – after all, they may not end up in academia.

**Paradigm Shift**

Even an initially appropriate choice of software can be rendered inappropriate by the development of other languages or by a shift of procedures within one’s field. For example, with the advance of...
open-science practices, researchers who use commercial programming languages or software products may want to enable other scientists to use their code. This may be a strong incentive to switch to free and open-source software in place of commercial solutions.

**Challenges**

What if your current language is not (or no longer) the ideal one? Switching software is not a trivial endeavor, and it is marred by several concrete challenges even for principal investigators, who, unlike trainees, have full discretion to change the workflow in their laboratory.

**The Weight of the Past**

Often, the choice of one’s initial programming language is not made consciously or deliberately. Indeed, a scientists’ first language is often decided for them – either by existing procedures in their first laboratory or by the formal training offered in their training department. Furthermore, certain methodological implementations in a scientist’s field may be available only in a specific language. Finally, some scientists may wish to capitalize on programming expertise that predates their scientific career.

**Learning the New Language**

For principal investigators (PIs) and trainees alike, the challenges begin with the obvious: learning the new language. However, unlike trainees, who can often devote significant time towards skill development, PIs’ discretionary time is limited. Beyond that, PIs are faced with specific challenges relating to the training of students and the changing of an existing code base in their laboratory.

**Instructing Your Trainees**

PIs whose proficiency in their original language allows them to comprehend and scrutinize their trainees’ code and provide hands-on instruction will likely want to achieve similar proficiency in the new language. If a PI is able to reach that level, the next question is when and how to instruct their trainees to switch to the new language. Trainees may have spent several months learning the original language, often before performing any actual research. Should these trainees be tasked with learning a new language, further delaying their primary research? Moreover, should they learn the new language at the same time as the PI; that is, during a period in which PIs themselves may lack proficiency and can provide only limited feedback?

**Why Change a Running System?**

Changing languages flies in the face of the old adage ‘never change a running system’. The existence of streamlined laboratory workflow in a specific language provides a strong disincentive against switching. Since programming is a means to an end (performing science), once the end can be comfortably achieved, changing existing means is attractive only given very strong incentives.

**Solutions**

Even after one has committed to switching software, the change may be seriously delayed or even prevented, notably because of the challenges summarized above. Several strategies can be implemented to lead a smooth and successful transition.

**Let the Students Be the Change**

Many students already have strong technical skills and may already be committed to the new software stack. For example, some individuals understand the power of open-source software and are motivated to use it over commercial options. By hiring such students and encouraging them to develop coding skills and lead projects using the new stack, these students can act as a catalyst and precipitate the change within the laboratory environment.

**Get on a Project**

Pick one new project to work on using the new language. Solving concrete problems will force students and PI alike to
learn skills directly relevant to their research. Ideally, this project will build on a real strength of the new language that demonstrates its added utility compared with the prior language. Python, for example, has unparalleled offerings for machine learning (scikit-learn, tensorflow, pytorch and keras, among others). Advanced statistical modeling and data visualization both have remarkable solutions implemented in R. The idea is to start getting the reward for the change as early as possible, allowing you to stay motivated to move forwards.

Divide and Conquer
If the laboratory has an extensive code base written with the old scientific stack, a transition period is unavoidable. Gradually start using the new language for small projects and slowly transition some functionalities from the legacy code base. You may be surprised to discover that large portions of the legacy code base have drop-in replacements in the new language, which may even be of better quality and wider functionality.

Contribute to Open-Source Software
Open-source communities offer a unique learning experience for coding, such as the development teams behind nilearn (https://nilearn.github.io/), nipype (https://nipype.readthedocs.io), MNE (https://martinos.org/mne/), and DIPY (http://nipy.org/dipy/). These communities welcome new contributors and have established guidelines to help them get onboard. There are also initiatives such as ‘hacktoberfest’ and dedicated scientific workshops such as brainhack (www.brainhack.org; for neuroscientists) and the NCBI hackathons (https://biohackathons.github.io/; for bioinformatics) where new contributors can often get in-person support to get started. Even if new contributors simply work on improving the documentation or adding new tests, the contributor may receive extensive feedback on their code, and will also have to look into an established code base adhering to some of the best development practices. For these reasons, the learning benefits of contributing to open-source software cannot be overstated.

With the advent of ever-more-complex analyses and ever-growing datasets, staying on top of one’s software stack is a core challenge for every scientist. We hope that this article can provide a helpful resource for researchers at any career stage who are looking to switch their primary programming language or scientific software.

Acknowledgments
We thank everyone who participated in the discussion on the initial Twitter thread. Figure 1 uses several openly available vector graphics downloaded from freepix.com.

*Department of Psychological and Brain Sciences and Department of Neurology, University of Iowa, Iowa City, IA, USA
1Department of Psychology, Stanford University, Stanford, CA, USA
2Centre de Recherche, Institut Universitaire de Gériatrie de Montreal, Université de Montréal, Montreal, QC, Canada
3Psychology Department, University of Montreal, Montreal, Canada
4All authors contributed equally to this article

*Correspondence: Jan-wessel@uiowa.edu (J.R. R), chrisgor@stanford.edu (K.K. Gorgolewski), pierre.bellec@crugm.qc.ca (Pierre Bellec).
https://doi.org/10.1016/j.tics.2019.01.004

© 2019 Elsevier Ltd. All rights reserved.

References