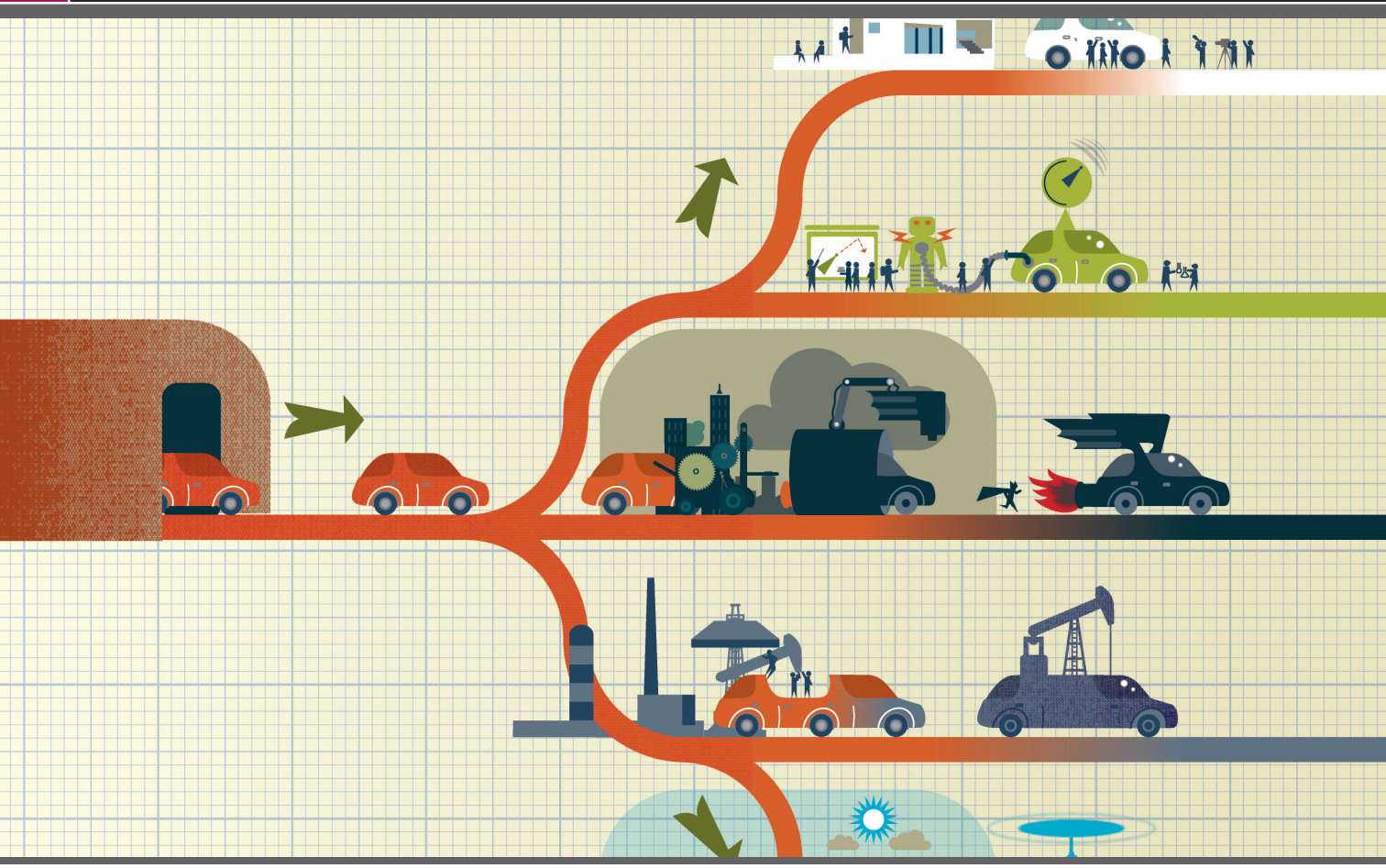

Scientific Software Communities

Andy Terrel | Continuum Analytics
Michael Tobis | Planet 3.0
George K. Thiruvathukal | Loyola University Chicago

If simulation is the third tier of science, then the communities that build the simulation software are the engine of innovation. Yet the scientific community as a whole tends to avoid issues surrounding the building of software. With a preference for more traditional scientific achievements, such as experimental results or theoretical derivations, the average scientist has attributed the writing, maintaining, and distribution of software as a tax that must be paid rather than a process that's rewarding in its own right. The scientific community as a whole, in its turn, neglects to reward producers of polished, shareable extensible software. The consequence is typically software that, while generally suitable for task, is brittle and problematic when viewed as an asset to the long-term needs of the scientific enterprise.

The importance of software to the modern world cannot be understated and software's importance to science is no different. Whereas successful software efforts lead to a fruitful, celebrated career in industry, the scientific software writer is often forgotten. To highlight this uncherished group, this issue of *CISE* has been devoted to presenting the challenges and the collective efforts of scientific

 1521-9615/15/$31.00 © 2015 IEEE Copublished by the IEEE CS and the AIP

software communities. The scientific software community has produced so many huge innovations in our society that it's vital that we make the process of building such communities well understood and well supported.

The Community's Role and Its Importance

In this issue, we hear two different approaches to reproducible software practices, an approach on maintaining documentation for important base libraries, and a discussion on ways of extending a software library's functionality to keep it relevant as a community evolves over time. These topics challenge the boundaries of what software can be for an individual and for a community. They expose weaknesses in our state-of-the-art practices with an eye towards a sustainable future. By using these techniques, we avoid numerous withdrawn results—a current crisis due to reliance on software without verification.

Our authors' insights into these problems give us an occasion to pause and ask what the fundamental role of the scientific software community should be. We posit that writing simulation software is transitioning from an activity that can be accomplished by a few coders who learned on the job to a full profession requiring years of study. This transition has prompted a few institutions to build centers to give a scientific software writer a place to have a career in academe, but this trend is new and is only appearing after a massive drain of talent to private enterprise. Certainly, building a community that supports and sustains the careers of our novice software developers is critical to the path of science.

The transition of scientific software from an occasionally used skill to a demanding career also requires additional policies, practices, and structures to motivate excellence in the field. One often-used analogy for scientific software is the comparison to a piece of experimental equipment. While the physics experimental instruments will have hundreds of authors, even though the software has a similar numbers of contributors, our papers usually only list the very few and very dedicated authors. Additionally, these large efforts directly pursue a grand challenge that unites a large portion of the field, while software communities tend to be the foundation that must be strong to allow further growth in all sciences.

Perhaps the analogy to the university library is more apt. As the center of an institution's knowledge, libraries are the base of knowledge for an institution to retain and teach its pupils. Software communities similarly build tools and libraries that enable future generations. Additionally, as a librarian's role is to help researchers answer difficult questions through previously collected and stored knowledge, the scientific software community's job is to enable researchers to answer difficult questions through knowledge collected and stored in code. In this regard, the scientific software community's role has become an extension of the librarians' role, as our scientific knowledge has become a product of the code we write.

As the librarian stands as the guide for knowledge throughout the university, the scientific software developer stands as the trailblazer for new, computationally intensive intellectual enterprises. The trail is one that leads to further science results and a healthy dialog feeding more software innovation as a result. This feedback loop of science result to new idea to software implementation to contribution to software library is vital to the continued success of scientific innovation. The need for recognition and maintenance of our software communities is critical.

In This Issue

Articles in this issue provide some examples of how a more considered focus on the software development process can feed the development of science.

Victoria Stodden and her colleagues write about the needed infrastructure to support reproducible science in "ResearchCompendia.org: Cyberinfrastructure for Reproducibility and Collaboration in Computational Science." They describe their ResearchCompendia effort, a Web portal for uploading and managing a compendium surrounding a scientific result. Addressing issues with the "ubiquity of errors" in science, including the specialized errors that occur in simulations, requires a complete pipeline that's documented, published, and managed. Such an effort requires a dedicated community to oversee the resource and help its adoption.

Next, in "Reproducible Research as a Community Effort: Lessons from the Madagascar Project," Sergey Fomel gives us a perspective on reproducibility from leading a scientific software community. Admirably, the Madagascar project holds reproducibility as its foundational goal. This goal requires a dedicated team maintaining the previous work and using tools that automate the reproduction of the work. Fomel argues very well that while this burden often falls out of possibility for a single

author, a community dedicated to lifting its members to their full potential benefits all of science.

In "Crowdsourcing Scientific Software Documentation: A Case Study of the NumPy Documentation Project," Aleksandra Pawlik and her colleagues turn our attention to documentation, a task dreaded by all software developers but vital for community projects. A potential solution to this gap in needs of the community and dedication of developer time is to crowdsource. The authors take us through the journey of creating infrastructure, maintaining the service, and engaging the user base to allow such a documentation procedure. By lowering the barriers of entry into contributing to the projects, we see the quality of the documentation grow and the community around the project grow as well.

Finally, Jed Brown and his colleagues take us on a thought experiment about what mass-market software would be like without run-time extensibility in "Run-Time Extensibility and Librarization of Simulation Software." The nightmare scenario that would drive end users away from their browsers is offered as an analogy to the standard operating practice of scientific computation. They propose adopting methods of run-time extensibility, allowing code methods to progress from the ad hoc methods of a small project toward the development of infrastructure for sustaining the innovation of an entire community.

In summary, the articles in this issue show us the benefits of applying professional software development standards to scientific software. Software projects which build in best practices such as extensibility, reproducibility, deployment, and testing, encourage further productivity among their users and subsequent developers. The road to making software that can be tested, understood, reused, and extended without undue hardship helps all of science, even though the cost of development will initially be higher.

Analogously, developing an academic environment which rewards domain specialists for attention to the methods of good software will have costs. An academic environment which provides reliable and rewarding career paths for developers who have the rare overlap of skillsets of numerical, statistical, or combinatorial algorithmics as well as systematic, testable, and extensible software development will have costs, too, especially in a world where a subset of those skills is in great commercial demand.

All of these approaches have been proven on various occasions to have enormous payoffs in various computing domains. The articles in this issue testify to the proposition that science is no exception in this regard.

Our sincerest hope is that this issue gives visibility to some of the challenges that software communities encounter and benefits they provide in supporting the science. Whether tenured professor, scientific staff, or new research assistant, community members' diligence in creating community-minded scientific software is critical to sustained innovation. Just as a river must have many feeding streams, scientific computation requires constant sources of ideas and implementations so that all of science reaps its benefit. ■

Andy Terrel is the Chief Science Officer at Continuum Analytics and President of the NumFOCUS Foundation, a foundation dedicated to sustaining scientific computing tools. His research includes utilizing supercomputers with Python and studying methods for speeding up computational fluid dynamics. Terrel has a PhD in computer science from the University of Chicago. He has contributed to numerous open source projects, notably the FEniCS Project and Sympy. Contact him at andy.terrel@gmail.com.

Michael Tobis is the editor-in-chief and cofounder of *Planet 3.0*, a site dedicated to scientifically informed conversations about sustainable technologies and cultures. His interests focus on the interface between science and public policy. Tobis has a PhD in atmospheric and oceanic sciences from the University of Wisconsin-Madison. Contact him at mtobis@gmail.com.

George K. Thiruvathukal is a full professor in the Computer Science Department at Loyola University Chicago, where he also directs the Center for Textual Studies and Digital Humanities. He is also a guest faculty member at Argonne National Laboratory in the Mathematics and Computer Science Division, working on runtime environments and software to support computational biology applications. His research interests span multiple areas of computer science and interactions with science and the humanities. Thiruvathukal has a PhD in computer science from the Illinois Institute of Technology. He is the editor in chief of *CiSE* and an associate editor for *Computing Now*. Contact him at gkt@cisemagazine.org.