An Ecosystem Perspective of Scientific Software Developer Productivity

C. Fan Du and James Howison[1]

An ecosystem perspective of scientific software developer productivity is to consider the contribution of software developers within the scientific software ecosystem. Researchers and practitioners of scientific software development are familiar with the fact that software components depend on each other in a layered architecture [2, 3, 7]. What further complicates the technological structure is a set of actors who engage in scientific software work [5]. While common examples of software ecosystem in the corporate space such as the Google Android ecosystem and Apple iOS application ecosystem are centered around a company-hosted platform of open innovation, involving both internal and external contributors in product development, other open source software ecosystems like Linux and Apache software ecosystem, leaning more towards their contributor communities, along with more bottom-up structure for coordinating development work [6]. In parallel, scientific software ecosystem is a distinct class, and perhaps a more complex class. One key reason lies in the complexity of scientific software ecosystem is that it spans across commercial space, communities of scientific researcher-developers, and institutions of science, including universities, research centers/laboratories, funding agencies, journals, professional societies, and science policy organizations and advocacy groups, etc [4]. The sophisticated networks of actors within scientific software ecosystem have multiplex implications, such as hybrid resourcing models and heterogeneous forms of organizing for scientific software projects [1]. These constitute the ***organizing complexity*** of scientific software ecosystem.

Another layer of complexity within the scientific software ecosystem is its ***technological complexity***. Open source scientific software components build on top of each other, resulting in software stacks with heavy dependencies often without *ex ante* consideration for a cost-minimizing dependency structure design. Such consideration requires a holistic view of open source scientific software dependencies, which is absent in science. As an outcome, the dependency risks of scientific software accumulate over time and threatens the whole ecosystem with the possibility of "software collapse" [2] and high ongoing maintenance costs. Moreover, other than the lack of understanding in the overall dependency structure of scientific software ecosystem, there is a need for more insights into the use, recombination, and complementarity of scientific software components, especially for the developers of these components [3]. Without these insights, the needed user-developer support and improvement of the software components, as well as the needed coordination work between scientific software projects often fall out of the sight of scientific software contributors. This is the complexity of software component use contexts and complementarity within scientific software ecosystem. The complexity of software dependencies, use contexts, and

[1] The University of Texas at Austin

complementarity largely constitutes the ***technological complexity*** of scientific software ecosystem. The ***technological complexity*** also involves technological changes, for example, the advances in particular software technologies or hardware could induce the need for updating existing software routines. In such cases software projects need to be aware of the external technological changes to remain well-functioning and up to date.

In fact, not only technology advancement affects the development trajectory of scientific software components, scientific progress also raises the requirement for software to stay in sync with novel datasets, approaches, methodological treatments, and techniques, etc. This is the ***science complexity*** of the scientific software ecosystem, as scientific software development and scientific progress go hand in hand.

Taken together, an ecosystem perspective of scientific software is revealing as it points to the organizing, technology, and science complexity within the ecosystem. These complexities are rarely attended to, ending up with a large portion of needed work unseen and thus undone within the ecosystem. In the following part, we aim to unpack the work needed to ensure the up and running of the scientific software ecosystem.

First, the science complexity and technological complexity require software projects to stay in sync, on one hand keep up with the external technology changes and scientific progress, on the other hand mitigate dependency risks and coordinate with each other for their complementarities. Sometimes, due to the organizing complexity of the ecosystem, scientific software projects need to react to the requirements of various users and stakeholders within the ecosystem. Altogether there is a demand for scientific software projects to keep abreast of their environment. We refer to this demand as the ***sensing work*** of scientific software projects.

Second, in reaction to all sorts of complexity and requirements ***sensed*** from the ecosystem environment, scientific software projects need to take action to adjust themselves and their software products. The due actions include fixing bugs, improving the design and architecture of software, implementing new methods, models, or procedures, providing user support such as updating documentation and responding to questions and bug reports from end-users and peer software producers, and sometimes connecting to and synergizing with other software projects such as those producing the integrated components. What we list here perhaps still does not exhaust all the ***adaptation work*** scientific software projects take on within the ecosystem.

Third, all the local adjustments and updates of one scientific software project as the result of ***sensing*** need to be channeled through the interconnected software projects, user and relevant stakeholder groups. Otherwise, if concerted efforts cannot be achieved among interconnected software projects and actors within the ecosystem, local adjustments will fail to serve its intended beneficiaries, and existing issues and risks remain unsolved. For local adjustments to achieve its due effect, ***synchronization work*** within the ecosystem needs to be accomplished. Synchronization means scientific software projects need to collect their adjustments, release them in an orderly fashion reaching out to all the related projects, stakeholders, and potential users/adopters. This is also coordination work that needs to be done at the ecosystem level, for a collection of interdepended projects to work together.

In summary, an ecosystem perspective of scientific software development is to examine software projects in relation to other projects and relevant stakeholders. Distinctively, scientific software ecosystem bears the complexity of organizing, technology, and science. These complexities that scientific software projects commonly face give rise to the work needed to be done at scale. Thus, an ecosystem perspective of scientific software developer productivity sheds light on the work developers need to engage in. If ***synchronization*** among interrelated scientific software projects and actors can be achieved at the ecosystem level, dependency risks will be effectively reduced, and scientific software will perform in an efficacious manner. The sustainability of scientific software will be consequently improved. Theoretically speaking, better software work will be accordingly achieved, and developers will be more motivated in their work.

But fronting the ideal of a synchronized scientific software ecosystem is the question of how can we motivate the needed work to be done? To answer this question needs not just the engagement of scientific software researchers, but also the instincts and experience of scientific software practitioners. Here we raise the question in hopes of opening a lively conversation.

Another note of our discussion is that while we primarily consider the scientific software ecosystem at its full scale, some scientific software projects lead its own ecosystem of interrelated sub-projects. However, such large software projects still run within the full-scale scientific software space. It will be a very interesting question, too, to consider how these organizational ecosystems of scientific software manage the relationship with their internal software projects and external projects.

Reference

1. Johanna Cohoon and James Howison. 2018. Routes to Sustainable Software: Transitioning to Peer Production. Academy of Management Proceedings 2018, 1: 12182.

2. Konrad Hinsen. 2019. Dealing with Software Collapse. Computing in Science & Engineering 21, 3: 104–108.

3. James Howison, Ewa Deelman, Michael J. McLennan, Rafael Ferreira da Silva, and James D. Herbsleb. 2015. Understanding the Scientific Software Ecosystem and Its Impact: Current and Future Measures. Research Evaluation 24, 4: 454–470.

4. James Howison and James D. Herbsleb. 2011. Scientific Software Production: Incentives and Collaboration. Proceedings of the ACM 2011 conference on Computer supported cooperative work - CSCW '11, ACM Press, 513.

5. Konstantinos Manikas and Klaus Marius Hansen. 2013. Software Ecosystems – A Systematic Literature Review. Journal of Systems and Software 86, 5: 1294–1306.

6. Maha Shaikh and Ola Henfridsson. 2017. Governing Open Source Software through Coordination Processes. Information and Organization 27, 2: 116–135.

7. James Howison and Kevin Crowston. 2014. Collaboration Through Open Superposition: A Theory of the Open Source Way. MIS Quarterly 38, 1: 29–50.