Research Evaluation Advance Access published July 27, 2015

*Research Evaluation* (2015) pp. 1–17 doi:10.1093/reseval/rvv014

# Understanding the scientific software ecosystem and its impact: Current and future measures

James Howison[1,*], Ewa Deelman[2], Michael J. McLennan[3], Rafael Ferreira da Silva[2] and James D. Herbsleb[4]

*[1]School of Information Studies, University of Texas at Austin, Austin, TX 78701, USA, [2]USC Information Sciences Institute, Marina del Rey, CA 90292, USA, [3]Research Computing, Purdue University, West Lafayette, IN 47906, USA and [4]School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA*

**Corresponding author. Email: jhowison@ischool.utexas.edu*

Software is increasingly important to the scientific enterprise, and science-funding agencies are increasingly funding software work. Accordingly, many different participants need insight into how to understand the relationship between software, its development, its use, and its scientific impact. In this article, we draw on interviews and participant observation to describe the information needs of domain scientists, software component producers, infrastructure providers, and ecosystem stewards, including science funders. We provide a framework by which to categorize different types of measures and their relationships as they reach around from funding, development, scientific use, and through to scientific impact. We use this framework to organize a presentation of existing measures and techniques, and to identify areas in which techniques are either not widespread, or are entirely missing. We conclude with policy recommendations designed to improve insight into the scientific software ecosystem, make it more understandable, and thereby contribute to the progress of science.

*Keywords: science; software; measurement; policy.*

## 1. Introduction

Modern science depends on software. Software analyzes data, simulates the physical world, and visualizes the results; just about every step of scientific work is affected by software. Although the use of commercial software is important in science, scientists, software developers, and students themselves are developing a significant amount of software used today in an academic setting. This software is funded primarily by the national funding institutions such as the National Science Foundation (NSF), National Institutes of Health, the Department of Energy (DoE), the Defense Advanced Research Projects Agency and their analogs globally.

Scientists, software and infrastructure providers, and funders have a wide range of information needs about scientific software, as we will discuss in detail below. Yet there is little data available to meet these needs. In this area, as in other areas in research evaluation, data availability is one of the 'critical issues for the evaluation of R&D programs and policies today' (Rogers 2013: 2).

Part of the complexity in measuring the scientific software ecosystem comes from the way that different pieces of software are brought together and recombined into workflows and assemblies. In this way, software runs 'on top of' other software components in a layered architecture, effectively hiding the components from users but gaining services and benefiting from those components. Each component is produced separately, providing services to each other but also starting, changing, and sometimes ending on different schedules. For this reason, we speak of a scientific software ecosystem. Adding further complexity, scientific software interacts with the scientific reputation economy, especially the publication system. To measure the scientific value of software and to guide its

© The Author 2015. Published by Oxford University Press.
This is an Open Access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted reuse, distribution, and reproduction in any medium, provided the original work is properly cited.

production and use, measurement must occur in many different places throughout a scientific software ecosystem.

Accordingly we address four questions:

(1) What do people in different roles in the scientific software ecosystem want to know?
(2) What measures are available and in use?
(3) What is not yet measured, why is it not measured, and how might it be measured?
(4) What policy options are available to improve the situation?

We draw on literature review and qualitative fieldwork, consisting of semi-structured interviews and participant observation. We have described our interviews and the interview protocols elsewhere Howison & Herbsleb (2011); Howison & Herbsleb (2013); in total we conducted 35-h-long interviews. While material from these interviews has formed the empirical basis for previous papers, we make use here of the considerable material that came out in the interviews on the needs expressed by the interviewees for information about how the software they developed, maintained, or administered is actually used. Indeed, it was the capture of this material that did not fit the themes of the previous papers that inspired us to do the work reported here. Our participant observation consisted of organizing four workshops over 4 years, and participating in three others. In addition, the second and third authors each have over 15 years experience in the scientific software ecosystem. Our literature review consisted of both the formal and the gray literature. In particular, we drew on reports from workshops focusing on the scientific software ecosystem, often sponsored the US NSF (e.g., Carver 2009; Stewart, Almes and Wheeler 2010; Stodden et al. 2010; Berente, Howison and King 2013).

Our method was guided by the work of Weick (1989). Weick describes theory development as ‘disciplined imagination’ directed to ‘sensemaking’ and subject to iterative selection criteria, especially the characteristic of ‘plausibility and interestingness’ that develops through this selection process. In practice, we recorded each interview, then discussed the interview among ourselves, producing memos that described our evolving understandings. Those memos then form a guide to returning to the interviews to re-listen and confirm interpretations. Our participant observation allowed us to bring our evolving understandings to the communities, engaging in ‘member checking’. In this way, our interpretations were often challenged, prompting us to seek further discussion (and sometimes interviews); we also shared drafts of this article with key informants, incorporating their insights. Sometimes, though, members found our interpretations useful, helping to recapitulate their own experience, and this helps confirm our interpretations as ‘plausible’ and ‘interesting’.

Wherever possible we highlight real-world examples of existing measurement to illustrate the measure we are discussing, pointing to Web sites and publications to assist the reader in pursuing further detail about a measurement technique and its successful implementation. It is important to note that our interviews were limited to participants in the USA, and our participant observation primarily occurred in the USA, although some authors have spent many years working on projects with European and other international partners.

The article is organized as follows. First, we examine the information needs of different roles in the scientific software ecosystem. We then abstract a framework by which to categorize different types of measures and their relationships. We use this framework to organize a presentation of existing techniques and to identify areas in which techniques are either not widespread or are entirely missing. We conclude with policy recommendations designed to improve insight into the scientific software ecosystem, make it more understandable, and thereby contribute to the progress of science.

## 2. Roles and information needs in the scientific software ecosystem

Our qualitative fieldwork and participant experience led us to identify four different roles that our informants occupied in the overall scientific software ecosystem, each linked to a need for different kinds of insight and thus measurement. The four roles are: (1) scientist end-user, using software to undertake domain science, (2) producing and distributing scientific software components, (3) administering scientific software installations, including HPC centers and software distributions, and (4) having concern over the overall functioning of the scientific software ecosystem, including senior scientists and science policy-makers such as funding agencies and their personnel. Any individual might well play more than one role at a time, or the role might be played by a group of individuals.

### 2.1 Scientific end-user

Scientists engaged in domain science are key players in the scientific software ecosystem. The scientists in this role we spoke with were driving forward their scientific investigations, and in the process drew together software artifacts to collect, manage, format, analyze, model, and visualize data, with the ultimate goal of publishing their results in the scientific literature. Scientific end-users are, from time to time, also writing software code, especially ‘glue code’ that links together software components obtained from elsewhere, as well as custom components that are intended for local use and not for redistribution (although they may be shared on request) (e.g., Brown et al. 2007).

From the perspective of the scientific end-user role, the scientific software ecosystem is a provider of software that facilitates and supports particular studies. In the words of one informant, the key thing is to ‘get the plots out’.

Toward this goal, end-user assemblers are concerned with the availability, quality, and usability of outside components. In addition, they are concerned with the likelihood that chosen components will continue to be scientifically useful, and that the components will continue to work with the other software components that the end-user has assembled to accomplish their scientific work.

Scientific end-users are interested in understanding what software and configurations were used in the production of their scientific results. In larger labs or collaborations, this means understanding and recording what software others used and how they used it. Yet it also means being able to accurately recall what they themselves used in the production of a specific result. Interestingly, while many informants called this an interest in reproducibility, their emphasis was on understanding the past to revisit it as a basis for future extension; end-user scientists want to record and recover what was used to *work* with the code, not simply to *replay* it.

Those in the scientific end-user role are interested in what other scientific end-users are using. When we asked participants why they used particular components, they indicated that they had an easier time keeping the focus on the scientific contribution of a paper if the software used was well understood by reviewers. And this meant using software used by many other scientists in their domain. A second reason that scientists are interested in what other end-users are using is that seeing others choosing to change the software used could indicate a shift in their methods, always of concern to scientists interested in the scientific frontier. Finally, widely used software was both a signal of its quality and an assurance that staff and graduate students with appropriate skills could be found and would find sufficient experience among their peers to resolve software questions quickly and maintain a focus on their science.

Scientists, as we will discuss below, think beyond their particular labs, but the key concerns expressed in the end-user role involved answers to these questions: What is the cost of the software (monetary, amount of resources needed to install, maintain, amount of time needed to learn the new software and integrate it into my methods)? What other software do I need to install, configure, or develop to use this software? If I make the investment of time and resources in deploying and using this software, who will support it, and how helpful they will be if I run into problems? How often is the software updated, and will I have to upgrade? Will this software still be around next year? What responsibilities (such as licenses or citation requests) does my use of this software bring to me? Who else is using this software (and the methods it enables)?

### 2.2 Software component production

A second ecosystem role is that of producing scientific software components and releasing them so that others might use them. Our informants here ranged from individuals to teams (both called projects for simplicity) producing artifacts that were used by only a few others to those used by entire disciplines. The projects we spoke with were often funded by grants, such as through the US NSF or DoE, but others were undertaking component production as part of a parallel software practice, rewarded solely by the reputation benefits of publishing papers about their software and the reputation derived from users' appreciation of their software. Some software that is eventually shared begins as software designed purely for the personal use described above. Sometimes this software stays within the confines of a single laboratory or group, but sometimes, it is shared with others and potentially broadly adopted, thus making the scientist part of the software ecosystem (e.g., Van de Geijn 1997).

In some cases, the software is developed in close collaboration between domain scientists and computer scientists, each conducting research. Computer scientists are using real-world problems to further understand the behavior of applications and the computer systems they run on. They develop new algorithms and incorporate them into software that is then used by the domain scientists for their research. A challenge for computer scientists is to abstract the problem as much as possible so that their solutions and software can be broadly used, potentially in a number of domains. Such is the case with the Pegasus Workflow Management System (Deelman et al. 2005), which develops workflow technologies that enable the execution of complex workflows in distributed environments. Pegasus started out as part of a project that targeted the gravitational-wave and high-energy physics community (e.g., Deelman et al. 2004). However, the solution was generic enough that it was adopted by astronomers, earthquake scientists, bioinformaticians, and others.

The measurement concerns of software component producers centered on what happens to their software outside the project. In effect, component producers were attending to their place in the overall software supply chain. This included an interest in understanding who is using their software component and what those users are doing with the code, including how it is configured. These insights are important to component producers for two key reasons.

First, understanding what happens to the software the project produces is closely linked to how and why scientific software projects obtain resources. Projects care about their scientific impact, both in terms of the number and type of users their software reaches, and in how their software contributes to the science that others are undertaking. For this reason, component producers are concerned that they had users they either did not know about or that their users used their software in ways that were not evident in the scientific literature. One informant laughed in frustration when discussing how he tells his funders how many citations his software papers have received, laughing because he believes that these citations represent ‘less than 10%’ of those who actually used his software for scientific work. Software component producers often (but not universally) use highlighted locations on their Web sites to appeal to their users to provide citations and reach out directly for letters of support when applying for grants.

Second, insight into what happens to the software a project produces helps the project to understand what work to undertake, what should be added or changed about their components. For some projects (typically grant-funded rather than open source), this is framed almost entirely as understanding ‘user requirements’ for their software. This can be quite broad in nature because it includes knowing when changes in other scientific components ought to affect the project. For example, a change in a component produced by others that feeds data to a project’s component might spark a set of bug reports and require the project to alter code that had, until the ecosystem around it changed, been working adequately. Bietz, Baumer and Lee (2010) describe this important kind of work as ‘synergizing’, as a project reacts to its surrounding ecosystem, in a manner that the ecological idea of adapting to a changing ‘niche’ helps to think about.

From an ecosystem perspective, then, software component producers are interested in ecosystem insight to answer these questions: Who uses our components and how, and what are their current challenges and upcoming computational needs? How have we contributed to the science of others? Is our work visible in the scientific literature? With what other components is our software used? How are those other components changing?

Finally, some projects are managed open source style, and are successful in attracting contributions from many scientists, including a long tail of contributors who make small but substantial contributions. Visibility of individuals’ contributions is often important for reputation and for justifying time and effort to the various funders of those making contributions. This raises the relevant measurement question of which scientists have contributed what functionality to an open project.

### 2.3 Software distribution and execution managers

A third role in the scientific software ecosystem is those who provide collections of software that are made available for end-users to use in their scientific work. Such software might be made available as a collection for download to the user’s own computers, or users may bring their computation to a collection of software hosted at a software execution environment, including supercomputers and cloud computing, and the science gateways that provide access to them (Wilkins-Diehr et al. 2008). Our informants included participants in two main examples: SBGrid and supercomputing centers providing national computational resources.

SBGrid provides a distribution of software for structural biologists, supported by membership fees (Morin et al. 2013). Member labs receive access to a software distribution and a set of shared licenses for those packages that require them. The software is gathered by SBGrid, packaged appropriately and distributed via periodic updates. Lab members assemble and run analyses on their own computing resources on which the SBGrid software has been installed. SBGrid staff also work as liaisons accepting bug-reports and, if the issue requires the attention of the software’s creators, ‘pushing them upstream’ to the appropriate software component producer. The work in SBGrid’s software distribution consisted primarily of monitoring projects for new versions, adopting those into the distribution, including building them for the target platform, and resolving their dependencies within the distribution. In addition, the work involved managing distribution configuration software designed, so that users can select the particular versions of packages they wish to use while undertaking particular analyses.

Supercomputing centers, like cloud computing providers, distribute software in conjunction with execution environments. While the software stays local to the center, typically running on the hardware they provide, it is distributed in the sense that it is made available to users. The software provided includes compilers specialized to particular hardware, middleware libraries that facilitate multiprocessor computing (e.g., Message Passing Interface or MPI stacks) and analysis packages specially tuned to perform well in the particular computing environment provided (e.g., a special compiled version of CHARMM, i.e., Chemistry at Harvard Macromolecular Mechanics software). A core concern of supercomputing centers is the utilization of their computing resources, which are shared among users; the center seeks to maximize the ability of their resources to support science, creating a focus on both attracting additional users and ensuring that their analyses are computationally efficient.

From an ecosystem perspective, both kinds of software distributors are interested in ecosystem insight to answer these questions: Who uses, or doesn’t use, what components? Which versions do they use? How frequently do they update? What new packages are available that should be included in a distribution or made available via the grid or cloud environment? These questions are particularly relevant to understanding when particular components can be retired, reducing the size of the distribution that must be managed, and freeing up resources for other work (McLay and Cazes 2012). Those who also manage computing resources that provide execution environments are interested in additional questions: Which code consumes the most computing resources; is it a candidate for optimization work? Which code leads to more execution errors? (Hadri et al. 2012). In addition, since both types of software distribution ultimately seek to contribute to scientific impact, either as part of professional identity or to argue for renewed grants, both are interested in understanding what science they have facilitated, in much the same manner as software component producers.

### 2.4 Ecosystem stewards

Many of our participants were also concerned with the overall functioning of the ecosystem and its contribution to science. These concerns were particularly salient for project officers at funding agencies, such as the NSF or DoE, but interest was also evident among other parties, including senior domain scientists reflecting beyond their individual work and thinking of their fields as a whole.

Concerns expressed here focus on questions about the operation of the ecosystem as a system that takes resources (time, money, and attention) and affects the conduct and output of science, both as a whole and in individual fields, complemented by interest in how the behavior of the system can be influenced. These are more traditional questions of research evaluation directed to understanding the impact of funding and programmatic decisions.

The cyberinfrastructure vision, expressed in the 'Atkins Report' and instantiated for the scientific software ecosystem in the NSF call for Software Infrastructure for Sustained Innovation (NSF SI2), envisions software not only advancing science but doing so with growing efficiency over time (Atkins 2003). Key to this is a belief that software ought to evolve toward a shared platform, with components that are reused as widely as possible as both end-users and component producers coalesce around particular pieces of software. The literature on software platforms outside science has called this 'coring' and 'tipping' (Gawer and Cusumano 2008) by which a community discovers its shared functionality and coalesces around packages which provide it, leading to efficient use of resources through economies of scale. Coring also results in increasing overlapping usage that facilitates more transparency in science, leading to greater quality and correctness, as more eyes and effort are directed to the same pieces of code that are sustained and evolve over long periods of scientific usefulness.

Coring toward platforms can be contrasted with its opposite, often perceived by informants: dysfunctional chaotic churn, with many projects with few users, each having short lives ending with their initial grant funding, disconnected and parallel user communities, stubbornly unchanging incompatibilities, and periodic and seemingly uncoordinated attempts at 're-boots'. Underlying this is a concern that opportunities are missed and that the progress of science is slowed (e.g., Stewart, Almes and Wheeler 2010).

These general concerns suggest a set of specific questions, focusing on overall patterns and emergent patterns within the ecosystem, including: What funding has gone toward software production? How many users or user communities do projects have? What are the scientific impacts of that use? Are user numbers growing? Do projects have sufficient resources and skills to handle their growth? Which projects have overlapping functionality? How long do pieces of software and projects persist? Do we have disconnected user and developer communities? Are particular components, or layers of components, missing? Which code is often used together; are the projects and people producing these components communicating appropriately? How can we sustain critical software?

Along with these questions are questions of how to influence the ecosystem, including questions of tipping points leading to coalescing use as well as direct policy interventions encouraging the use of particular components. Here there is a clear tension between a desire for flexibility and freedom, linked to expectations of scientific innovation and desires for authority structures and coordinating control. Questions of influence include those like: Which funding programs, and which requirements in their calls, have resulted in widely used software and substantial scientific impact? What are the features of fields that have achieved greater coalescence? Which journals and conferences have exemplary policies? How is software work viewed within hiring and evaluation practices, such as tenure cases?

## 3. Current and missing measures for the scientific software ecosystem

The interests and information needs discussed above are many and varied; they extend throughout the lifecycle of the creation, distribution, use, and evaluation of software components. To organize our discussion below, we present a simple process model framework of software in science, shown in Fig. 1.

Resources are devoted to the production of scientific software. End-user scientists (directly or indirectly) use software components to undertake science, resulting in science impact. Science impact then justifies resources, either prospectively (anticipated impact justifies initial resources) or retrospectively (scientific impact justifies ongoing resources).

Many of the questions discussed above touch on multiple parts of the framework, making it a useful way to organize our discussion of measurement techniques. For

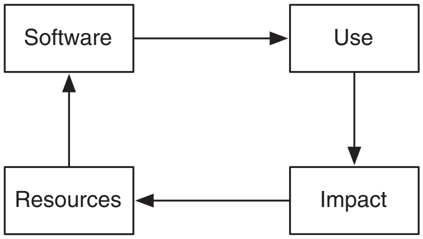


**Figure 1.** A process model of software in science.

example, ecosystem stewards (such as funding agency program officers) may be primarily responsible for allocating resources, but to do so effectively, they require insight into each of the areas, from the software produced (or proposed) to its scientific impact. Component producers are focused on understanding the work needed and arguing for the provision of resources to do that work, but to do so, they need to illustrate the scientific impact of their software, via its use (or prospective use) in science. Scientific end-users are interested primarily in documenting their internal use of software (and in the scientific impact that use might have) and seeing what others are using, but are also interested in how the software is produced and resourced, for reasons of quality and the sustainability of the software components they select.

As we will see, many of the metrics currently employed focus only on one element in our process model, limiting their broader usefulness. Improving the ability of metrics to assess the overall process (extending 'around' the elements in the model, following the process) is therefore key to our conclusions and policy recommendations. In the discussion below, we examine each of the four connections in the framework above. Yet some of these connections are easier to measure at present; thus, our discussion is unbalanced, substantially longer for some than others. We return to this issue in the conclusion, drawing on it to highlight the need for measure development in and across each part of our framework.

### 3.1 Resources to software

Understanding how scientific software has come to be built is a crucial first step in measuring the scientific software ecosystem. We see two starting points for this inquiry: one could start with scientific software packages and has who has made contributions over time and how those participants were motivated (and in some cases, funded). Or, one could start with funding sources and ask how each has resulted in software used by scientists. Both approaches share a need for data about the work of building software: who did what, when.

The first strategy was employed in Howison and Herbsleb (2011) which examined the software used to construct three focal papers, collecting data on motivations and funding through interviews. They identified six models of software production in science, primarily distinguished by the key motivations: (1) direct monetary rewards (commercial software, employed software developers), (2) academic reputation (incidental software, driven by direct scientific need), (3) parallel software practice (scientific need enhanced by publishing 'software papers' alongside domain research), (4) a software subfield (direct reputation for software work), and (5) hybrids such as dual-licensing and software work within large collaborations (software as direct scientific contribution). The resources used in the production of software work, therefore, did include direct grants for software work, but also included a diverse range of resource origins, including money allocated from domain grants, collaboration service work, and the ostensibly 'free-time' of scientists pursuing their direct research, and thus funded by the diverse range of funding sources that underlie any scientific career, from instructional budgets, grant overhead, the willingness of scientists (and graduate students) to work 'overtime', as well as domain science grants and software-specific grants.

Interview approaches are useful for qualitative insight but do not scale to collect data at the scale required to understand most of the questions posed above. A promising mid-level approach is possible when the history of work on a software product is available in source code control systems. Systems such as CVS Subversion, and git (which underlies github) store each change to the code over time ('committing') and keep track of who added that change to the repository ('committer'). After accounting for the specifics of a project's workflow, the identity and institutional location of the committer can stand as a proxy for the source of resources that has supported the production of that software. In research outside science, this approach has been successful in understanding the involvement of different companies in providing resources for open source software development, using the organizational domain of the email address making a commit (e.g., Wagstrom et al. 2010).

One advantage of these repositories is the potential to identify episodes of contribution and to relate them to sources of support. By starting with the full list of 'commits', one can seek background on the work that lead to those contributions, one can avoid the over-generalization that is commonly a feature of memory, as done for studying open source development outside science in Howison and Crowston (2014). In this way, effort contributed to projects from various sources, including what might be called 'dark effort' that cannot be allocated to any particular source of funding.[1]

Other approaches start with research funding, specifically the administrative data about grant making (Rogers 2013), and attempt to track the software products that have resulted. More specifically the US NSF recently adjusted its reporting to specifically include software products along with other research outputs such as publications, Web sites, and data sets. Accordingly, agencies are able to build systems that connect their grant funding to specific software products. Efforts in this area include catalogs produced by the NSF's Advanced Cyberinfrastructure program and the OpenCatalog run by DARPA. These efforts have tended to be limited to programs that specifically fund software but could, in theory, be extended to all grant funding organizations.

Tracing the connection between research funding and software produced could draw on existing techniques used to connect funding and other contributions, especially publications. By examining publications, researchers can draw on the inclusion of funding acknowledgment statements, typically required by funding agencies and/or conflict of interest disclosures, to connect publications with funding. Recently some bibliographic databases have begun to systematically extract funding acknowledgment information and present it more directly (Rigby 2011).

Collecting and analyzing funding acknowledgment for publications is much easier than doing so for software. First, funding agencies have insisted on funding acknowledgments in publications for many years; it is an established and expected practice. Second, bibliographic databases provide centralized collections of publications for analysis. With software, neither of these conditions are in place: scientific software packages are often not collected into large collections, but distributed individually, and it is not clear whether a funding agency acknowledgment in an application or its source code is required and where it ought to be located. For example, some projects make an acknowledgment of funding on their project Web sites, rather than embedding that acknowledgment in the source code itself.

### 3.2 Software to use

The distribution and use of software in science is important to measure if we are to address the information needs describe above; it is also surprisingly difficult. One-time surveys can play a role but they are costly and impose a significant burden of response on already busy ecosystem participants; even then as one-shot data collection surveys do not produce ongoing insight. Accordingly, we emphasize on measures which draw on side effects of software activity. There are, as we see it, four opportunities to observe software use, or reasonable proxies of use: (1) when the software is obtained by users, (2) when users seek support in its use, (3) when the software actually executes, and (4) when software workflows and assemblies are stored.

### 3.3 Measuring distribution

When software is distributed commercially, the requirement to pay money for a software license has two implications: the first is that the software producer learns the identity of its users, while the second is that the value of money makes it less likely that a user will pay for software and then not use it (especially when it comes to updates).

When software is distributed for free download, or simply installed at supercomputing centers, there is no automatic record of the user obtaining the software. This is in contrast to software sold commercially, and even with open source software obtained by downloads from repositories, or through package management systems. The lack of automatically accessible data means that projects (or others interested) must make specific efforts to understand who has obtained their software. There are two key techniques used to accomplish this: the first is counting downloads and the second is registering those seeking downloads.

Download measurement involves analyzing the log files created when the software is downloaded, either directly from a common location or through a package management system. A first issue arises when software is distributed in multiple channels: such as download direct from a project's Web site, inclusion with another software package, inclusion in a distribution, installation directly at an execution site, or copying from friends. In large projects, one channel may dominate sufficiently to stand in for a minimum metric, but many scientific software projects have relatively small potential market sizes, and so missing entire distribution channels may disproportionately understate total distribution.

More problematically, distribution is a poor proxy for use. Many downloads may not result in use at all: the downloader may try the software and find it wanting for some reason. Moreover, many downloads may result from nonscience or even nonhuman distribution, as when a bot downloads the software, perhaps as a result of web spidering.

Finally, when software is updated and new versions released, actual users may re-download the software. Thus, if downloads are not adjusted for new releases, download figures will overstate use by orders of magnitude. This is both problematic and an opportunity. In these cases, downloads spike immediately after a download as regular users update the software. Techniques exist to use this understanding to convert time series of raw download numbers into estimates of installed base and estimates of the experimentation (and thus the conversion of downloaders into regular users). Fig. 2, from Wiggins, Howison and Crowston (2009), shows the logic.

Passive, distribution-focused techniques such as those above may, with appropriate adjustments, provide aggregate figures of the number of users. For other purposes, however, including extending around the framework above, there is a need to understand the identities of users, both to reach out and understand their usage environment and to seek evidence of scientific impact (see below). Registration of users asks those seeking the software to provide contact details before a registration link is provided, usually requesting basic information such as a name, email address, institution, and perhaps domain science field (e.g., the VMD project from UIUC, <http://www.ks.uiuc.edu/Development/Download/download.cgi?PackageName=VMD>). A similar function is provided when code is described in a publication but made available 'by request to the author'; this technique may chill some exploration of the software, but at least the producer has a clear idea of those that are really interested. As with downloads, however, registration

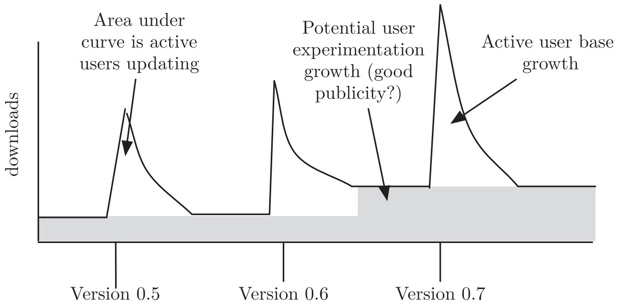

Area under curve is active users updating
Potential user experimentation growth (good publicity?)
Active user base growth
downloads
Version 0.5
Version 0.6
Version 0.7

**Figure 2.** Converting download time-series into estimates of the installed base (Wiggins, Howison and Crowston 2009).

in this way counts only distribution and not use directly. Moreover, the request can be viewed as annoying by potential downloaders, even invasive of a user's privacy, especially in highly competitive fields. Nonetheless, it may be reasonable to prioritize creating a list of likely users, even at the cost of putting off a few potential users, for the project or for those who have funded it; the alternative is either little real understanding of use or costly efforts to survey a user base that quickly goes out of date.

### 3.4 Measuring support

A question asked on a project mailing list can be a useful source of data about software use. It reveals the identity of the user, and the content of a question can reveal information about the nature of the use, even extending to understanding which surrounding components are complementary to a software component. Accordingly, measuring contacts from users is a proxy of use employed by scientific software projects. Such contacts may come through a variety of formats, from mailing lists, forums, bug-trackers, and direct email. In some cases, software providers assign tracking numbers to their support questions, thus potentially making it easier to track the support provided for the software. An example of such a software project is HTCondor (Litzkow, Livny and Mutka 1988). Projects may also count users who are not posting but are listening, such as when projects tout the number of subscribers to their project mailing lists.

As a proxy for use, measuring activity has both pros and cons. It is reasonable to assume that those subscribing, updating, or asking questions are more likely than anonymous downloaders to actually be running the software, as some downloaders don't continue to use the software (Wiggins, Howison and Crowston 2009). Similarly, while it is easy to interpret a high number of support requests or bug-reports as indicating software that is difficult to use (perhaps due to poor documentation), the effort required to post a question indicates a sincere desire to use (or even improve) the software and thus that the asker values it. Conversely, however, only software that is user-facing is likely to generate active support discussions; infrastructural software or development libraries that are drawn on by developers of other software may be very widely used without generating any widespread support discussion (consider how many users of email clients are aware of the library that actually interacts with a mail server, or how many developers use a library for string manipulation without encountering any issues.). In fact high-quality software may disappear by these measures because it 'simply works'.

In addition, counting raw numbers of user messages to a project is subject to the same sorts of double counting that counting downloads without adjusting for versions is: a single user may generate many messages. Thus counts of messages used as a proxy for use ought to be adjusted by normalizing by identifiers, such as email addresses or real names. Subscription-based numbers can be normalized by employing mailing-list probes that automatically unsubscribe those with bouncing emails or those that fail to respond to a prompt.

### 3.5 Measuring software execution

In theory, software can report a great deal about its own use, beginning with the simple fact that it ran and extending outward to describe the circumstances under which it ran. This is because a program can, in the course of its execution, send a message to a central server. That message might simply report that the software ran, as described for the Globus software by Thain, Tannenbaum and Livny (2006). That message can, however, be more complicated. For example, desktop programs, including Microsoft Word, often ask users if the program can generate and communicate logs of memory and execution state when a program crashes, providing very useful data to developers (albeit at some risk of overload). Programs may also have useful information about the environment in which they execute, including the operating system and version, or other 'nearby' software such as any virtual machines that a component is interacting with or other runtime software dependencies.

An excellent example originating outside science is the package 'popularity contest' available for the Debian and Ubuntu Linux distribution (http://popcon.debian.org/). This package examines the user's file system and local package database to determine which packages have been installed and, by observing the last date particular files were accessed, how often the package was run. Results are reported via an anonymous UUID, stored temporarily and publicized only in aggregate. The project README explains how these data are used by the maintainers of the distribution to make decisions about which packages to prioritize.

Scientific end-users may see data collection efforts as overly invasive, arguing that they affect performance and communicate sensitive data (Thain, Tannenbaum and Livny 2006). In Structural Biology, for example, the mere fact that a lab is investigating a particular molecule communicates that they think it is tractable, revealing the results of months of offline work and providing a competitive leg up for competitors (Howison and Herbsleb 2011). On the other hand, projects may see concerns where there are, in fact, few. Thain, Tannenbaum and Livny (2006) report that while their survey revealed broad concerns about privacy, actual users were not concerned about revealing their own use; in fact users asked to be added to illustrations of global Globus use. This intriguing result suggests that privacy concerns are more abstract than concrete, at least for this particular user community; nonetheless, community support is paramount.

A second class of runtime observation is instrumenting locations such as supercomputing or cloud computing environments, where the code (and data) is brought together to execute. Unlike personal or lab infrastructure, these shared computing resources are not owned by the scientists undertaking the computation. Moreover, these resources are made available under particular conditions of use; providers have legitimate reasons to understand what is executing on their platforms. Supercomputing centers have traditionally focused on reporting resource availability and utilization rates, but have not sought to examine in detail nor report on the software used by scientists 'inside' computational jobs. For example, the OSG metrics and the XSEDE Metrics on Demand (XDMoD) project can record and analyze usage via a variety of parameters, including which machines, which accounts, and even which scientific field used computational cycles, but don't record data about the software run inside computational jobs (e.g., Barbera et al. 2012; Furlani et al. 2013).

Recently, however, a number of execution environments have moved toward understanding what software is being run within the computational jobs executing on their systems. These inquiries are driven primarily by the desire to understand how efficiently computational resources are being used. The Texas Advanced Computing Center, for example, has begun to collect and report data on how the specialized and tuned software libraries (known as 'modules') are used in user's computational jobs (McLay and Cazes 2012), helping the center prioritize the maintenance and optimization of most used components. Similar efforts have been undertaken with Cray supercomputers (Hadri et al. 2012).

Science gateways provide customized access to computational resources for particular scientific domains, making it easier to find tools that others have created and reproduce results. This mode of service-oriented computing is increasingly successful, including examples such as HUBzero (McLennan and Kennell 2010), Galaxy (Goecks et al. 2010), and MyExperiment (Roure et al. 2009). These environments provide both execution and archiving services, assisting users in the aims discussed above by offering a repository of tools and workflows, and an easy way of running them without having to download, compile, or install any code. Along the way, the services can collect data about what tools and workflows users have accessed, and even what specific users have run and in what configurations. Currently, however, the usage agreements and understandings with users preclude particular uses of these data. Various sites based on HUBzero, for example, will report an aggregate number of simulation runs for all tools over time, an aggregate number of runs for a particular tool over time, and even a breakdown of users by country throughout the world, but sites will not report information about individual users or even about the input parameter space that has been explored by a tool. Services are appropriately concerned with the trust their users hold in them and hold users' data confidential.

Nonetheless, the aggregate data that these services report is extremely useful. HUBzero uses those data as a factor in an automatically calculated 'ranking' for a tool, which is a measure of quality on a scale of 0–10. The higher the usage, along with user reviews, citations, and support activity associated with a tool, the higher the ranking. Tools with high ranking tend to bubble up in searching and browsing activities, while those with low ranking are harder to find, a characteristic likely to lead to others preferentially discovering the tool and thus to community coalescence around tools. Furthermore, such aggregate impact metrics are a strong incentive for researchers to contribute new tools to the repository. On HUBzero sites, each contributor has an area on his profile showing the impact metrics for all of his contributions, including the number of users that have run a tool, the number of runs that were performed, and the number of citations for the tool in academic literature. MyExperiment has similar profile pages showing a user's contributions, along with interests, friends, groups, and credits from other users who have reused their assets. Researchers use these data as part of their story of scientific impact, such as in tenure and promotion cases and to argue for funding for continued research (https://nanohub.org/members/3482/usage).

Some services provide detailed information about a user's own activities, including the data files accessed, the tools used, and the way that computations were chained together. For example, Galaxy provides this sort of detailed provenance information as a way of helping users to build new workflows. Users can experiment with their data, see what works, annotate steps as they move along, and create new workflows automatically based on the steps recorded in detailed histories. Users may choose to share their data sets, histories, and workflows within Galaxy to other specific users or to the world as published entities. Galaxy 'pages' provide a complete description of each computational experiment, including notes and diagrams, and embedded data sets, workflows, and histories. MyExperiment supports a similar concept through 'packs' of materials that include the workflow, results from running the workflow, presentation materials describing the project, and publications that may have resulted from the effort.

Full access to the extensive user data gathered by these services can be studied to provide insight across the scientific ecosystem. For example, nanoHUB.org, the Web site that spawned the HUBzero platform, has been active since 2002 and has grown to more than 300,000 users each year; therefore, it has a wealth of user interaction data. These data have been shared with social scientists under controls satisfying the user privacy policy. One study has shown that tools originally created for research tend to migrate into educational settings, and that the median time for adoption is less than 6 months (Madhavan, Zentner and Klimeck 2013). Another study has shown different principles of team formation in the nanoHUB user community and has extracted the characteristics of successful teams (Contractor 2013).

A further possibility is assessing use by mining archives of scientific workflows, such as those required for publication in some fields (e.g., McCullough, McGeary and Harrison 2006). The emphasis for these archives has been on reproducibility rather than measurement per se (Stodden et al. 2010); typically, the expectation is that a finalized, published piece of research is stored or associated with the workflow that underlies the results. Some efforts in this area attempt to automate the entire process, from data analysis, generating figures, integrating them with the paper text, and producing the PDF, an approach known as an executable paper (http://www.executablepapers.com). Once an archive contains many records of research stored in this manner, however, mining its archives to identify particular packages that have been used would be possible, with the added advantage of directly associating that use with the associated publication, thus reaching around the cycle described above, linking use and scientific impact.

Finally, surveys of users (and potential users) are possible, seeking an indication from a body of scientists as to who has used which software and for what purpose. One challenge is establishing an appropriate sample frame. For example, it would be problematic to simply survey a user community, since that would not represent the input of nonusers or ex-users. Similarly, open calls for respondents would seem far more likely to attract responses from those already using the software. A second, related challenge is coping with motivation of participants to respond. Survey responses are time-consuming and hold little intrinsic interest for those being surveyed. Systematic bias stemming from enthusiastic users (or unenthusiastic ex-users) is likely to be more motivated than nonusers to respond to a survey request from a project. One possibility might be arranging a survey around a convenient but widespread sample frame, such as all submissions to a journal or a conference. Nonetheless, surveys are difficult vehicles through which to obtain detailed understanding of usage, given the limits of respondent's specific recall and the trade-off between response rate and requested detail.

### 3.6 Use to impact

While impact in research evaluation means both impact on science and broader social and economic impact (Penfield et al. 2014), we focus on science impact because broader impacts of research infrastructure are mediated by their ability to support scientific research.

A common approach is for providers of funding, software, and execution resources to ask their users to help them report impact, typically by requesting that users tell the providers which publications have been facilitated. Scientists and others are familiar with requests like this from the grant funding process; grantors request periodic and final reports that summarize how resources have been turned into impact, typically by requesting that funded projects provide lists of publications and other contributions that have been supported.

Analogous requests from providers of software packages to their users suffer from two problems, however: motivation and specificity.

Motivation for acknowledgment is problematic because unlike providers of funding, providers of software have little leverage; requests for acknowledgment are dependent on appealing to the better natures of users and intimating that the continuance or improvement of the project is at stake. Current practice in science does not take advantage of the possibility offered by software licensing, whereby the use of software could be made contingent on users providing appropriate acknowledgment, just as is done with Creative Commons Attribution licenses and the Apache Foundation's NOTICE source code licensing policy (http://www.apache.org/licenses/example-NOTICE.txt).

More problematic still, however, is the question of specificity: simply providing a citation of an article does not say much about *how* that research benefited from the particular piece of software, nor does it suggest that the research might not have been possible without that piece of software (a case much more easily made for the provision of research funding). Finally questions of 'fractional credit' arise (Katz 2014): if it is useful information that a research paper was facilitated by a piece of software used directly, to what extent is it also useful to imply a contribution by those who provided software dependencies drawn on by the software directly used? If so, how might one reallocate the credit accruing among the dependencies? These questions, however, are likely dwarfed by the difficulty of simply having users provide this information in the first place, especially as a project grows beyond a small group of users. Moreover this system would suffer from gaps in data: if any project did not ask its users then the chain of credit would be broken for lower dependencies.

An alternative approach is to view publications as a data set that can be mined to identify software used in the preparation of the publication and thus likely to have contributed to the science in some way. Scientific publications themselves do contain significant data on software use. These come in three forms: citations, mentions, and artifacts inserted in the paper (such as graphs). Using citations to identify software use and thus argue for impact means analyzing citation lists for publications that refer to particular software packages. This is possible when projects have written 'software publications' or 'methods papers' about software packages and their users have come to understand that they ought to cite these papers. When this holds, it ought to be possible to use automated bibliometric systems to extract such citations and connect them to the software packages they indicate. Unfortunately, the practices of citation to software vary considerably from field to field Howison and Bullard (in press) and appear to miss significant software. Howison and Herbsleb (2011), for example, found that projects did not formally cite entire classes of software, including commercial software (even when papers are published) and software developed in-house, either by the authors or by software developers in a collaboration. Moreover, some venues actively discourage citations to software packages, and when a paper has used many packages, attempts to mention them all could lead to unacceptably long citation lists.

A second form of evidence in the publication record is mentions. By this we simply mean the use of package names in the free text and footnotes of publications rather than formal citations to software publications. In some fields, such as bioinformatics, it is a common practice to list programs and their versions in the 'materials and methods' sections of papers in a manner similar to chemical reagents or instruments. These mentions may be appropriate to measure use and thus impact in the literature, particularly when the evaluation research question begins with a list of packages to look for.

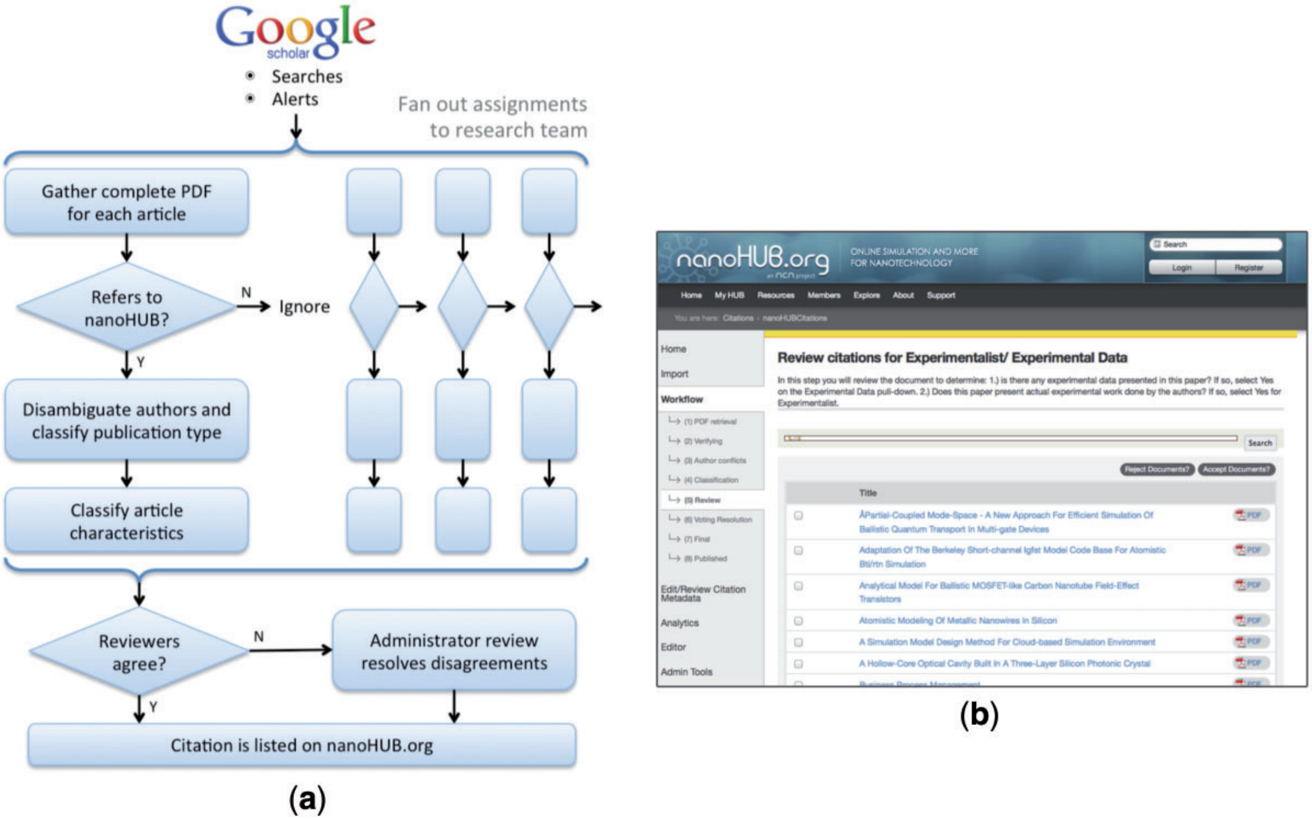
The third form of evidence in publications is through the inclusion of characteristic artifacts that the software creates. By this we mean figures and tables generated by particular software packages, especially graphics and statistical packages. Just as one can recognize default templates from, say, Microsoft office, it should be possible to train machine learning algorithms to identify plots produced by particular software. In its limited area of application, this technique could be quite powerful, allowing us to chart the use of particular software through the history of the published literature. A number of research projects are investigating this, including the Impact Story project (Piwowar 2013) and some of the current authors Howison and Bullard (in press).

The nanoHUB project has developed an approach to identifying impact in publications that mix automated analysis with manual inspection by experts to judge the extent to which a publication truly drew on the software or services provided by nanoHUB and thus ought to be claimed as a contribution. This process starts by gathering potential citations, primarily through Google Scholar searches and automated alerts for specific keywords (such as 'nanoHUB' and unique tool names). The resulting citations are fed into a web-based system created on nanoHUB to manage the remaining workflow, illustrated in Fig. 3.

Various staff and graduate researchers each tackle a subset of the citations. First, they track down a PDF document containing the full text of the article, and perform a cursory reading to determine whether or not it actually pertains to nanoHUB. Any false positives are eliminated from consideration. Records for the remaining articles are classified according to publication type (journal article, conference proceedings, thesis, technical report, etc.), and author names are linked to existing nanoHUB user profiles. Publications are further scrutinized to see if the authors are theorists or experimentalists, if the publication includes data from experiments, and if it references a particular tool or some other resource on nanoHUB. Results from multiple reviewers are combined, and if they disagree, the nanoHUB managing director steps in for final review and resolves the dispute. Once fully processed and approved, citations become visible on the 'citations' page on nanoHUB.org, which also includes diagrams of coauthorship networks, numbers of secondary citations, and nanoHUB's own h-index calculation. So far, this process has located more than 1,000 citations pertaining to nanoHUB, of which 80% are related to nanotechnology research and 30% include data from experiments. Citations are automatically listed on the page for each simulation tool that they reference. This helps the users of a tool, as well as funders, understand the academic literature relevant to the use and impact of the tool.

Two additional data sources are available, at least in some cases, each of which can extend beyond publications: user reviews and qualitative stores of impact. Some software repositories, including nanoHUB, permit users to 'favorite' software packages and to provide reviews. Other reviews are available periodically in software survey papers published in some fields (e.g., Renfro 2004), including those which compare the performance of different software packages (e.g., Tasker et al. 2008). Opportunities exist to make this practice more widespread, but the question of what motivation potential reviewers have to take the time to provide reviews (outside publications) would require further inquiry.

A final, but important, source of information about the connection between use and impact is the collection of qualitative stories of impact. For example, the SBGrid project (a software distribution for Structural Biologists) publishes periodic 'Member Tales', discussing how software has been used in the conduct of the science in member labs. Other publications include these stories as well, such as Scientific Computing World and International



**Review citations for Experimentalist/ Experimental Data**

In this step you will review the document to determine: 1.) is there any experimental data presented in this paper? If so, select Yes on the Experimental Data pull-down. 2.) Does this paper present actual experimental work done by the authors? If so, select Yes for Experimentalist.

| | Title | |
|---|---|---|
| □ | ÂPartial-Coupled Mode-Space - A New Approach For Efficient Simulation Of Ballistic Quantum Transport In Multi-gate Devices | PDF |
| □ | Adaptation Of The Berkeley Short-channel Igfet Model Code Base For Atomistic Bti/rtn Simulation | PDF |
| □ | Analytical Model For Ballistic MOSFET-like Carbon Nanotube Field-Effect Transistors | PDF |
| □ | Atomistic Modeling Of Metallic Nanowires In Silicon | PDF |
| □ | A Simulation Model Design Method For Cloud-based Simulation Environment | PDF |
| □ | A Hollow-Core Optical Cavity Built In A Three-Layer Silicon Photonic Crystal | PDF |
| □ | Business Process Management | PDF |

**Figure 3.** (a) The process of gathering citations related to nanoHUB.org, and (b) the web infrastructure that supports that process.

Science Grid This Week (isgtw). Undoubtedly, letters of support or commitment from users that accompany grant applications include similar stories. Yet these tend to be one-offs and nonsystematic. Future studies could address this, creating systematic qualitative studies of the widespread impact of a small number of packages, or assessing the software usage over a systematic sample of scientists.

### 3.7 Impact to resources

The final aspect of our framework is from impact to resources, or the manner in which the scientific software system rewards the achievement of desired impact with further resources. This is perhaps the least well-understood link in our framework, yet is crucial as a driver of activity throughout the scientific software ecosystem.

Commercial software directly links impact and resources through sales; if a package is not being useful, it will not be purchased (or at least upgrades will not be purchased) and the project will receive fewer resources. Yet in science, at least for noncommercial software, this process is far more indirect.

Perhaps, the most direct route is when scientists develop software for themselves or their collaboration, thereby creating quite local scientific impact through improved quality and productivity of their own science. In this case, the route from impact to resources is indistinguishable from the general, nonsoftware routes. Improved academic reputation leads to a suite of resource rewards from improved employment contracts, potentially lucrative prizes, consulting/expert witness opportunities, and success in grant applications. Conceptually, perhaps, the contribution of the scientist's software work might be distinguishable from their domain science contribution, but it would be extremely difficult to separate in practice, a prerequisite to measurement.

For cases where the impact is on the science of others, the route to additional resources is more indirect still. In particular, it is widely claimed that toolmaking is underrewarded as a scientific contribution, both in terms of limited career paths and limited availability of grants/funds for software work directly (e.g., Stewart, Almes and Wheeler 2010). This is perhaps particularly true when compared with rewards available for software work in industry (Howison, Berente and King 2013).

Success in winning grants is a key source of resources. Yet we know little about how grant panels assess software contributions, including how they assess use and impact measures in considering awarding grants. Research could assess how grant applicants make use of different impact measures (such as citation rates, user numbers, and resource utilization claims) and, potentially, examine panel summaries or even discussion to understand which measures are considered credible and relevant and which are not. Another research question would ask what aspects of a project, its community, the use of the software, and the structure of its surrounding ecosystem do evaluators need to understand to make informed and successful decisions about resourcing for projects. Similarly, research could assess how potential scientific end-users of software understand the resourcing (and thus likely sustainability) of projects.

Other, more amorphous, sources of resources can be grouped together under the concept of career success. There are a number of anecdotal stories of career successes and failures associated with software contributions in different communities. Generally speaking, discussions at workshops have tended to focus on career failures rather than successes, but systematic assessments have not been attempted. A significant recent success is that the 2013 Nobel Prize in Chemistry included direct reference to models, which were implemented by the CHARMM software package that Martin Karplus, one of the laureates, originally developed to do his work with models. Research effort could assess career paths, particularly focusing on the ways in which claims of impact have been used within tenure cases and other evaluations within science (such as in national labs and supercomputing centers). The altmetrics community, particularly the Impact Story project, has worked to assist scientists to tell the story of their impact beyond individual publications and to include these on their academic CVs (Piwowar and Priem 2013).

## 4. Conclusion and policy recommendations

In the discourse on scientific infrastructure, there is an often-held goal of providing infrastructure, including software, that is so easy-to-use and seamless that the infrastructure becomes invisible; it ‘gets out of the way’ so much that users don’t perceive its existence. This understanding of the nature of successful infrastructure is shared by work in science and technology studies, describing the invisibility of infrastructure and the work that maintains it (Star and Strauss 1999). Yet descriptions of successful infrastructures are descriptions of an endpoint, not a strategy for achieving that endpoint. While it is true and useful to say that electricity, as an infrastructure, is largely invisible to its users, it is far from true to think that those building that infrastructure, especially in the early days, ever wanted it to be that way. In fact, as Marvin (1988) and others have shown, particularly when discussing the Chicago World’s Fair, electricity in its early days was a gaudy, showy affair, built to impress and explicitly demonstrate. The route to invisibility was anything but invisible.

The measurement techniques in this article, used together, would increase the visibility of the use, the role, and the consequentiality of scientific software. Nonetheless, the interests of the various players in the scientific software ecosystem that we have identified will be served differently by different techniques. Further, the techniques themselves often require trade-offs that impose responsibilities or perceived inconveniences on some players (such as scientific end-users or system administrators) to provide insight that is of most interest to other players (such as science funders or component-producing projects). This is particularly true of techniques that extend around the process model we describe above, linking different phases of the impact cycle. Accordingly, the implementation of more advanced techniques will require a collective effort of understanding and balancing interests in the scientific software ecosystem. Still, much can be done already, using techniques that closely align effort and interest in the information to be collected and analyzed.

Accordingly, we break our recommendations into two groups. First we make recommendations to ecosystem participants to help answer their questions revealed in our fieldwork, while producing data likely to be useful more broadly. These have the advantage of being well motivated and amenable to local action. For some, they require further sharing of data to be useful for broad-based research evaluation, although the recommendations to ecosystem stewards aim to create circumstances that will prompt that sharing. Second, we make recommendations as to goals for longer-term collective action that address the overall aims of research evaluation (while drawing on, or seeking to usefully alter, the local motivations of ecosystem participants). Table 1 shows recommendations for ecosystem participants; we turn to longer-term recommendations for collective goals in the text that follows.

The table above shows recommendations that can be acted on by individual players in different positions in the software ecosystem. These recommendations have the advantage of being well motivated since they are techniques to answer the specific questions of those in ecosystem roles, while simultaneously producing data that are of interest to other roles and to the overall practice of research evaluation.

However many of these are limited in their ability to ‘reach around’ the framework presented earlier in our article. For example, software component producers can attempt to collect data on impact through publications but that succeeds only to the extent that they can rely on users to mention software in publications. In turn, this relies on policies enacted by journals and funders to standardize and incentivize users. Similarly, ecosystem managers seeking to draw on usage data to plan priorities for funding efforts rely on actions to be taken by both component producers and users of scientific software. Finally, because much software use is indirect through software dependency (and therefore hidden from the user and any potential evaluator), there is a need to collect software dependency information broadly and to link it to many of the measures above (particularly impact measures).

**Table 1.** Questions and concerns of ecosystem participants and recommendations for addressing them that tends to produce broadly useful data

| | |
|---|---|
| | Scientific end-users |
| What software did those in my lab use and how did we configure it? | Use source code control systems and make all data available for analysis.<br>Use Lab Information Management Systems and workflow systems.<br>Use package management systems for dependencies wherever possible, and make dependency data available. |
| What software are my colleagues using? What software are reviewers comfortable with? | Encourage colleagues to make their software use visible in publications and presentations.<br>Publish in journals with software citation policies and follow them. |
| How should I acknowledge my use of scientific software? | Examine the project documentation for a request for citation and/or license conditions.<br>Publish in journals with software citation policies and follow them. |
| | Software component producers |
| Who uses our software and how? | Have an explicit strategy for user survey<br>Consider limiting distribution to known channels, record logs to count, and analyze downloads<br>Instrument software with reporting that balances user competitive concerns with need to demonstrate project impact |
| How have we contributed to the science of others? Is our work visible in the scientific literature? | Make obvious and explicit, standard requests for citation (don't rely on users knowing what to do); consider pushing this request into user workspace.<br>Implement regular literature searching (described above) to locate likely users and confirm with them. |
| | Software execution environment and distribution managers |
| How has our execution environment contributed to Science? | As with software component producers, make explicit request for citation and monitor literature. |
| Who uses, or doesn't use, what components? Which versions do they use? How frequently do they update? | Inspect code running within computational jobs, not merely the metadata of the job (user, time, utilization).<br>Distributions can inspect component use on end-user systems (as with Debian Popularity Contest)<br>Visualize component use over time.<br>Make aggregated visualizations available to users (to encourage coalescence and use of updated components) |
| | Ecosystem stewards |
| What funding has gone toward producing which software components? | Ensure that project reporting systems include software components, ideally with separate links to primary codebase and community support and distribution venues (provides a template for project organization).<br>Consider providing standard distribution practices (e.g., providing repositories or 'blessing' appropriate repositories). |
| Who is using which software? | Require projects to develop and report their plans for understanding the use of their code. Help projects (and their users) see importance of measuring use.<br>Ensure that reporting is regular (allowing midcourse corrections)<br>Incentivize other ecosystem participants to follow the recommendations above, and provide central locations into which their data can be shared. |
| Which areas need targeted funding? Are particular components, or layers of components, missing? | Analyze available workflows that link components together; are there locations across workflows with bespoke components?<br>Ask funded hosted environments (e.g., clouds/grids) to provide aggregated and appropriately anonymized data on code running on them as a condition of their funding.<br>Change usage agreements for hosted facilities to facilitate analysis of jobs while attending to competitive exposure (e.g., aggregation techniques).<br>Encourage and fund software distributions; require funded projects to make code available through distributions. |
| What are the scientific impacts of software use? | Work with publishing venues to ensure appropriate software visibility (esp. software citation policies).<br>Provide template processes for funded projects to measure impact.<br>Work directly with intended user-communities and social scholars of science to document software impact. |
| Do projects have sufficient resources and skills to handle their growth? | Monitor use metrics and watch for 'phase changes' (rapid growth in use), proactively assess project organization and funding levels. |
| Are projects on a path to sustainability? | Revise reporting requirements to assess community health of the project:<br>● Does the project have contributors outside those directly funded?<br>● Does the project actively encourage contribution and are offered contributions integrated?<br>● Are project venues open and transparent?<br>● Do projects make explicit and clear request for acknowledgment? |
| Are software-contributing scientists achieving appropriate credit and successful career progression? | Conduct scientific workforce studies: Where do contributors come from? Where do they go to?<br>Work with publication venues for appropriate software citation policies. |

Thus, there is a need for coordinated action across different roles in the scientific software ecosystem toward improved research evaluation and scientific practice. We frame these as goals, rather than immediately actionable recommendations, because they require community input and development, and most likely, implementation strategies specific to each scientific community. We believe that the goals of coordinated action should focus on solidifying each step around our framework, making visible the links:

(1) develop improved reporting requirements for grants to make clear the connection between research funding and software products, including requirements for how software is made available;
(2) establish the norm that that software component producers have a responsibility to collect data about the usage and impact of their software;
(3) build an expectation that execution environments should write their usage agreements in such a way that makes aggregated and anonymized usage data available and the norm that the user's confidentiality requirements balance against the need to collect useful data;
(4) establish and collectively enforce clear policies for mentioning software that contributed to scientific results in publications;
(5) build a cultural understanding among users of the importance of appropriate acknowledgment of software contributions; and
(6) Work to improve the rewards available to those making key software contributions (perhaps including establishing well-funded prizes to reward nongrant funded software work).

The appropriate route to realize these goals collectively must involve widespread community consultation appropriate to individual scientific fields; accordingly, traditional venues such as agency-funded workshops will play a role, as might high-level 'blue-ribbon' panels the UK's Scientific Software Sustainability Institute shows a way forward here. Bottom-up approaches will play a role as well, particularly connecting with the movement toward 'open science', including movements such as Mozilla's Science Lab (https://wiki.mozilla.org/ScienceLab) and executable papers, drawing on its energy and ability to shape norms for younger scientific practitioners. Each scientific field, together with its funding agencies, will need to find appropriate ways forward. In any case, though, the recommendations for immediate action in the table above should tend toward demonstrating the potential in creating the data and then in sharing it for ecosystem-level insights.

Accordingly, policy-makers can encourage new research that draws on these emerging information sources to improve the understanding of ecosystem functioning. This may include additional funding programs targeted at researchers from fields such as social network analysis and management and strategy, as well as experienced survey researchers. Other innovative approaches might include creating a machine learning challenge toward recognizing software in publications, drawing in new constituencies interested in advancing understanding of the scientific software ecosystem.

Similarly, there is a need for research in the field of research evaluation on the relative costs and benefits of different measures, in different locations around the cycle and the software ecosystem. For example, the efforts by nanoHub to locate papers that drew on their project, discussed above, require the time of many students to assess papers for nanoHUB impact. While this time with the literature is of ancillary benefit to the students, it is not clear what the cost trade-off is in improving the measurement of impact. Similarly some of the metrics discussed are more invasive than others, raising questions about privacy or confidentiality (e.g., automated insight into 'backstage' software use and examination of consideration of impact in peer review of software focused grants). The trade-off here is not in financial terms, but in terms of the potential controversy and dissatisfaction of scientists, compared with the value of data collected; it may be that communities establish new norms of transparency, but their imposition from outside is unlikely to be well received, undermining trust and perhaps other efforts. Financial trade-offs are, of course, also relevant: some of the metrics proposed in this article are likely to be more costly than others, while producing information of different quality. There is a need to investigate this further, examining the cost, context, and information quality from different measures enable those interested in research evaluation to prioritize the further development of particular measures.

In conclusion, this article has argued that the central and still growing importance of software to the collective achievement of science requires a response from those interested in understanding scientific practice, among scientists themselves, those building infrastructure for scientists, and those seeking insight at the policy level, especially for research evaluation. Albeit primarily drawing on the context of the USA, we have provided empirical insights into what information needs to exist and a framework for understanding those needs and how they interact. Finally, we have provided a practical overview of existing measurement techniques and highlighted areas of policy and collective-action challenge for the research evaluation of scientific software. Realizing the potential for software to improve science demands no less.

## Funding

This material is based in part upon work supported by the US National Science Foundation under Grant Nos. 09-43168, 10-64209, 11-48515, 02-28390, and 06-34750.

## Note

1. We are indebted to an anonymous reviewer for highlighting the potential importance of 'dark effort'.

## References

Atkins, D. (2003). *Report of the National Science Foundation Blue-Ribbon Advisory Panel on Cyberinfrastructure* <http://www.nsf.gov/od/oci/reports/toc.jsp>.

Barbera, R. *et al.* (2012). 'Gustav: CPU accounting for small-sized grid infrastructures', *International Journal of Grid Utility Computing*, 3/2-3: 89–96. DOI: 10.1504/IJGUC.2012.047759.

Berente, N., Howison, J. and King, J. L. (2013). *Report on Workshop on 'Managing Cyberinfrastructure Centers'*. University of Georgia <http://managingcenters.net/>.

Bietz, M. J., Baumer, E. P. and Lee, C. P. (2010). 'Synergizing in cyberinfrastructure development', *Computer Supported Cooperative Work*, 19/3-4: 245–81. DOI: 10.1007/s10606-010-9114-y.

Brown, D. A. *et al.* (2007). 'A case study on the use of workflow technologies for scientific analysis: gravitational wave data analysis', in Taylor I. J. *et al.* (eds) *Workflows for e-Science*, pp. 39–59. London: Springer.

Carver, J. C. (2009). 'First international workshop on software engineering for computational science and engineering', *Computing in Science and Engineering*, 11/2: 11.

Contractor, N. (2013). 'Moneyball for nanoHUB: theory-driven and data-driven approaches to understand the formation and success of software development teams', in Daniel F., Wang J. and Weber B. (eds) Business Process Management*, Lecture Notes in Computer Science*, pp. 1–3. Berlin Heidelberg: Springer.

Deelman, E. *et al.* (2004). 'Pegasus and the pulsar search: from metadata to execution on the grid', in Wyrzykowski, R. *et al.* (eds) *Parallel Processing and Applied Mathematics*, Lecture Notes in Computer Science, pp. 821–30. Berlin, Germany: Springer.

——. (2005). 'Pegasus: a framework for mapping complex scientific workflows onto distributed systems', *Scientific Programming*, 13/3: 219–37.

De Roure, D. *et al.* (2009). 'Towards open science: the myExperiment approach', *Concurrency and Computation: Practice and Experience*, 22/17: 2335–53.

Furlani, T. R. *et al.* (2013). 'Performance metrics and auditing framework using application kernels for high-performance computer systems', *Concurrency and Computation: Practice and Experience*, 25/7: 918–31. DOI: 10.1002/cpe.2871.

Gawer, A. and Cusumano, M. A. (2008). 'How companies become platform leaders', *MIT Sloan Management Review*, 49: 28–35.

Goecks, J. *et al.* (2010). 'Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences', *Genome Biology*, 11/8: R86. DOI: 10.1186/gb-2010-11-8-r86.

Hadri, B. *et al.* (2012). 'Software usage on Cray systems across three centers (NICS, ORNL and CSCS)', *Proceedings of the Cray User Group Conference (CUG 2012)*, Stuttgart, Germany.

Howison, J., Berente, N. and King, J. L. (2013). 'From loss to gain: exploiting diaspora in cyberinfrastructure enterprises', *Presented at the Atlanta Conference on Science and Innovation Policy*, September 26, Atlanta, GA.

Howison, J. and Crowston, K. (2014). 'Collaboration through open superposition: a theory of the open source way', *MIS Quarterly*, 38/1: 29–50.

Howison, J. and Herbsleb, J. D. (2011). 'Scientific software production: incentives and collaboration', *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pp. 513–22, Hangzhou, China. DOI: 10.1145/1958824.1958904.

Howison, J. & Herbsleb, J. D. (2013). 'Incentives and integration in scientific software production. In', *Proceedings of the ACM Conference on Computer Supported Cooperative Work* (pp. 459470). San Antonio, TX. http://doi.org/10.1145/2441776.2441828.

Howison, J., & Bullard, J. (in press). 'Software in the scientific literature: Problems with seeing, finding, and using software mentioned in the biology literature', *Journal of the Association for Information Science and Technology (JASIST)*. Retrieved from http://dx.doi.org/10.6084/m9.figshare.1146366.

Katz, D. S. (2014). 'Transitive credit as a means to address social and technological concerns stemming from citation and attribution of digital products', *Journal of Open Research Software*, 2/1: e20. DOI: 10.5334/jors.be.

Litzkow, M. J., Livny, M. and Mutka, M. W. (1988). 'Condor-a hunter of idle workstations', *Presented at the 8th International Conference on Distributed Computing Systems*, pp. 104–11. San Jose, CA, USA DOI: 10.1109/DCS.1988.12507.

Madhavan, K., Zentner, M. and Klimeck, G. (2013). 'Learning and research in the cloud', *Nature Nanotechnology*, 8/11: 786–9. DOI: 10.1038/nnano.2013.231.

Marvin, C. (1988). *When Old Technologies Were New*. Oxford: Oxford University Press.

McCullough, B. D., McGeary, K. A. and Harrison, T. D. (2006). 'Lessons from the JMCB archive', *Journal of Money, Credit, and Banking*, 38/4: 1093–107.

McLay, R. and Cazes, J. (2012). Characterizing the Workload on Production HPC Clusters *(Working Paper)*. Texas Advanced Computing Center.

McLennan, M. and Kennell, R. (2010). 'HUBzero: a platform for dissemination and collaboration in computational science and engineering', *Computing in Science and Engineering*, 12/2: 48–53. DOI: 10.1109/MCSE.2010.41.

Morin, A. *et al.* (2013). 'Collaboration gets the most out of software', *eLife*, 2. DOI: 10.7554/eLife.01456.

Penfield, T. *et al.* (2014). 'Assessment, evaluations, and definitions of research impact: a review', *Research Evaluation*, 23/1: 21–32. DOI: 10.1093/reseval/rvt021.

Piwowar, H. (2013). 'Altmetrics: value all research products', *Nature*, 493/7431: 159. DOI: 10.1038/493159a.

Piwowar, H. and Priem, J. (2013). 'The power of altmetrics on a CV', *Bulletin of the American Society for Information Science and Technology*, 39/4: 10–13. DOI: 10.1002/bult.2013.1720390405.

Renfro, C. G. (2004). 'A compendium of existing econometric software packages', *Journal of Economics and Social Measurement*, 29: 359–409.

Rigby, J. (2011). 'Systematic grant and funding body acknowledgement data for publications: new dimensions and new controversies for research policy and evaluation', *Research Evaluation*, 20/5: 365–75. DOI: 10.3152/095820211X13164389670392.

Rogers, J. D. (2013). 'Introducing the special section theme: recent developments in data sources and analysis for R&D evaluation', *Research Evaluation*, 22/5: 269–71. DOI: 10.1093/reseval/rvt027.

Star, S. L. and Strauss, A. (1999). 'Layers of silence, arenas of voice: the ecology of visible and invisible work', *Computer Supported Cooperative Work (CSCW)*, 8/1: 2–30.

Stewart, C. A., Almes, G. T. and Wheeler, B. C. (eds). (2010). *NSF Cyberinfrastructure Software Sustainability and Reusability Workshop Report*. This is hosted on Indiana University's institutional repository: https://scholarworks.iu.edu/dspace/handle/2022/6701

Stodden, V. *et al*. (2010). 'Reproducible research', *Computing in Science and Engineering*, 12/5: 8–13.

Tasker, E. J. *et al*. (2008). 'A test suite for quantitative comparison of hydrodynamic codes in astrophysics', *Monthly Notices of the Royal Astronomical Society*, 390: 1267–81. DOI: 10.1111/j.1365-2966.2008.13836.x.

Thain, D., Tannenbaum, T. and Livny, M. (2006). 'How to measure a large open-source distributed system', *Concurrency and Computation: Practice and Experience*, 18/15: 1989–2019.

Van de Geijn, R. A. (1997). *Using PLAPACK–Parallel Linear Algebra Package*. Cambridge, MA: MIT Press.

Wagstrom, P. *et al*. (2010). 'The impact of commercial organizations on volunteer participation in an online community', *Presented at the Academy of Management Conference (OCIS Division)*, August 6, Montréal, Canada.

Weick, K. E. (1989). 'Theory construction as disciplined imagination', *Academy of Management Review*, 14/4: 516–31.

Wiggins, A., Howison, J. and Crowston, K. (2009). 'Heartbeat: measuring active user base and potential user interest in floss projects', in Boldyreff C. *et al*. (eds) *Proceedings of 5th IFIP WG 2.13 International Conference on Open Source Systems (OSS 2009)*, pp. 94–104. Skövde, Sweden: Springer.

Wilkins-Diehr, N. *et al*. (2008). 'TeraGrid science gateways and their impact on science', *Computer*, 41/11: 32–41. DOI: 10.1109/MC.2008.470.