The Future of E-infrastructures

Judith Segal, Marian Petre, and Helen Sharp | The Open University

Public funding bodies in the UK and the US have published documents laying out a broadly similar vision for funding the development of e-infrastructures to support scientific collaborations. Analyzing these documents reveals much to commend in them as well as some weaknesses that could lead to failure to achieve their aims. One such weakness is lack of adequate recognition of the effect of the academic reward structure on the development and deployment of e-infrastructures.

The search for the Higgs boson, the huge advances in genetics, the improving understanding of climate change, and many other recent and ongoing high-profile scientific research endeavors depend on collaboration among scientists. These collaborations in turn depend on hardware and software infrastructure, referred to as e-infrastructures in the UK and as cyberinfrastructure in the US (we use the terms interchangeably in this article).

Large amounts of public money have been invested in e-infrastructures—in the UK alone, the e-Science program funded projects worth more than £213 million between 2001 and 2006 (www.epsrc.ac.uk/news/events/pubs/rcuk-review-of-e-science-2009-building-a-uk-foundation-for-the-transformative-enhancement-of-research-and-innovation/). After the program’s completion, the UK Engineering and Physical Sciences Research Council (EPSRC) consulted with the relevant community to reflect on how e-infrastructures might best be developed and deployed in the future. These deliberations resulted in three documents published in 2012: a strategy document and an action plan (both available from www.epsrc.ac.uk/research/ourportfolio/themes/researchinfrastructure/subthemes/einfrastructure/software/), and a report on research software engineers (www.software.ac.uk/resources/get-speed). These documents were synthesized and expanded into an e-infrastructure roadmap (www.epsrc.ac.uk/research/ourportfolio/themes/researchinfrastructure/subthemes/einfrastructure/strategy/roadmap/). This roadmap is intended to be used by both scientists and peer reviewers to check that projects being proposed for funding are consistent with the national strategy.

Recent research has revealed that an e-collaboration’s success doesn’t depend solely on its hardware and software underpinning. Rather, the people involved and the cultural, social, and organizational contexts in which they work—the human infrastructure—play a significant role.1, 2 Here, we focus on the human infrastructure involved in developing the underpinning software rather than on the scientific collaborations per se. We analyzed the funders’ documents from the perspective of this focus and in the light of recent empirical studies on e-infrastructure development.

Throughout this article, we include illustrative examples from a four-year-long field study (2006 to 2010) of the development of a relatively small e-infrastructure software package, a laboratory information management system for the use of protein scientists that we call FLIMS (field study laboratory information management system).

Commitments Made in the Funders’ Documents

In our opinion, the most important contributions made in the documents are the declarations of the funding councils’ commitments to funding infrastructure development and to enhancing the value of scientist developers.
Funding was secured only after a further aim was added: to promote data standards so as to enable interoperability between different structural biology software programs and between these programs and instruments used in structural biology.

Funding Infrastructure Software Development

The commitment to fund infrastructure development by both the EPSRC and the NSF is already being made concrete in calls for proposals. Examples in the UK include proposals for the development of software for use in computational science and engineering, and for “novel software tools” for the biosciences (www.epsrc.ac.uk/funding/calls/softwarefuture and www.bbsrc.ac.uk/funding/opportunities/2012/2012-tools-resources-development-call2.aspx, respectively). In the US, a specific NSF program—SF1, Software Infrastructure for Sustained Innovation (www.nsf.gov/funding/pgm_summ.jsp?pims_id=503489)—funds the development of scientific software elements and infrastructure, and the establishment of scientific software innovation institutes.

Before the commitment, Rob Baxter and his colleagues describe the situation in the UK prior to 2012 as follows: “Until recently … Research Councils have had no funding policy for software development and sustainability which has led to the need to disguise such development in grant proposals.” The consequences of disguise could be very harmful.

In the case of the UK e-Science program, the stated aim was to facilitate collaborations between scientists. However, rather than focusing on the scientists, establishing their requirements, and then developing a robust infrastructure, the thrust of the e-Science program was on computer science research. The first call was explicit in this regard (http://web.archive.org/web/20010628022656/http://www.epsrc.ac.uk/EPSRCWEB/MAIN/SUPPRES/CURESOP/INTRO/INTRO.asp?Return_CFP.asp): “It is expected that the majority of funding will support [sic] people developing and using advanced informatics techniques applied to specific text beds.”

During the program’s implementation, it became clear that the aim of supporting scientific collaborations was being compromised by the clashing interests of the computer scientists and the scientists whom the software was intended to support. This conflict was recognized in the program review (http://www.epsrc.ac.uk/newssevents/pubs/?pageNumber=6&resultsPerPage=25&filterSortBy=releaseDate&filterSortOrder=asc), which quoted the following “excellent comment” from the University of Reading, UK: “Scientists often looked to computer scientists to perform the jobs of software engineers and develop robust tools to order; conversely, computer scientists often—quite reasonably—wish to focus on the latest techniques as part of their own research agenda.”

This situation isn’t unique to the UK. In their analysis of four scientific collaborations in the US, David Ribes and Thomas Finholt noted that an emphasis on computing science research leads to the production of novel software that might be unsuitable for supporting the work practices of scientists: “Novel platforms often do not match the extent needs of users, do not offer the functional stability that daily use demands, or lack the human resources to upgrade and maintain existing technology.”

In the case of FLIMS, the need for a laboratory information management system for protein scientists became apparent in the early 2000s with the massive amounts of data generated by new experimental techniques for forming protein crystals in a lab. Early attempts to secure funding for the FLIMS development from UK and EU sources were unsuccessful. These early bids focused on data management, and funding was secured only after a further aim was added: to promote data standards so as to enable interoperability between different structural biology software programs and between these programs and instruments used in structural biology. The data standards were reified by a protein production data model. The lead scientists behind FLIMS described this model as an “almost complete representation of requirements,” thus revealing that they saw requirements primarily in terms of the identification of the data items that should be captured in the database.

The protein production data model was initially made central to FLIMS. According to the project manager/lead developer, the decision to do this was somewhat controversial, and several reasons were put forward to justify it. The primary reason arguably was that it furthered the cause of data standardization. In the first release, both the relational database schema and the user interface were generated from, and were a reflection of, this model.

The model’s centrality adversely impacted FLIMS’s overall development. The original plan was for FLIMS to be developed in an iterative, incremental fashion, with subsequent releases addressing issues raised by users and introducing further functionality. But the bench scientists (the intended primary users of FLIMS, who were largely PhD students and postdoctoral researchers) hated the first implementation. What was important to them wasn’t the identification of the data items that needed entering into the
database but rather the ease of entering and retrieving data, the ability to copy relevant information in FLIMS to a paper or thesis, and the general usability of FLIMS.

The first release paid no attention to these requirements, with the consequence being that the bench scientists refused to use FLIMS, stymieing the overall plan of an iterative, incremental development. It was clear that the FLIMS development model needed to focus less on the protein production data model and more on meeting the needs of the bench scientists. Eventually, the development grew independent of the model, and the aim of supporting data standardization was quietly jettisoned.

The impact of initially basing FLIMS development on this model lasted a long time. Not only did it take some years for the software to be made independent of the model, but it also took a long time to regain the interest and engagement of potential users whose perception of FLIMS, based on the first release, was that it was totally unusable. Had there been a straightforward path to obtaining funding for the development of data management software for protein scientists and thus no necessity to satisfy aims other than the production of an effective piece of software, this situation might not have arisen.

Enhancing the Value of Scientist Developer

We define a scientist developer as a person with high educational attainment and research experience in a particular science, very possibly with a PhD, who develops software to support the practice of that science. Such a person might be a scientist end-user developer (someone who develops, or contributes to the development of, software for his or her own scientific use), whose self-identity is very firmly that of a scientist, or alternatively someone whose career has progressed from a focus on scientific research to spending all, or most, of his or her time on developing and maintaining software for his or her speciality. People in the latter category, whom we shall call career scientist developers, might have attended courses in software engineering and work either in academia or in industry. This definition of scientist developer differs somewhat from that of research software engineer, namely, “These individuals combine a professional attitude to the exercise of software engineering with a deep understanding of research topics.” We believe that in the case of most scientist end-user developers, the professional attitude they espouse is to the science and not to software engineering.

The Value of Scientist Developers. We’ve observed that much valuable software supporting scientific research, especially in the mathematically based disciplines, is written by scientist end-user developers. Perhaps the most important reason for this is the issue of requirements and needs. Scientist end-user developers are essentially writing software for themselves; they have a deep understanding of the problem the software is intended to address, and if a first coding attempt isn’t satisfactory, they can modify and extend the code at will without having to explain their needs to someone else.

However, one of the significant characteristics of scientist end-user development is its localization: the software is intended (initially) to address one particular problem for (at most) a small group of scientists at a particular point in time. Such software is in some sense the exact opposite of e-infrastructure software, which is intended to support a wide variety of users doing a wide variety of tasks over a long timeframe.

Nonetheless, scientist developers can provide a great deal of value to an e-infrastructure development. Baxter and his colleagues, focusing on career science developers, emphasize how their understanding of research culture gives them an edge as scientific software developers over generalist contract programmers. Judith Segal and Chris Morris describe how a scientist end-user developer, being respected both as a scientist and a scientific software developer, acted as an effective bridge between scientists and developers and, by visiting prospective user labs and discussing his experience of using FLIMS, increased the community of users. The authors also describe how the same scientist end-user developer and two career scientist developers made suggestions based on their knowledge of the science and their experience of using scientific software, which increased both the usability and flexibility of FLIMS. The project manager/lead developer is of the opinion that these suggestions were crucial to the eventual success of FLIMS.

Scientists’ Perceptions of Scientist End-User Developers. Segal conducted a series of interviews with scientist end-user developers in 2007. One emergent theme in those interviews was the lack of recognition ascribed by scientists to the value of software development knowledge and skills, and to the time required for software development. Interviewees talked of the perceptions of many scientists that “everyone” knows how to develop software as part of their skills base and that development can be “spun off” outside formal work hours, for example, “in half an hour over your lunch time.”

The lack of recognition of the particular skills and value of scientific end-user developers has been acknowledged since the early days of scientific infrastructure software. More recently, writers have argued that science is a reputation economy, with reputation being based on scientific publications, and that scientists’ endeavors in developing—and even more so, in maintaining—software to support their community in its practice of science count for very little in strengthening that reputation.

Our experience is that, in some cases, a publication is associated with a piece of software: the developer publishes a description of the software in a journal dedicated to techniques and tools in that particular science, and use
of the software is permitted only on condition that the user cites the paper when publishing his or her scientific results. However, this isn’t a panacea to the problem of recognizing the importance of scientist end-user developers. One issue is that, in a culture that appears not to recognize the knowledge and skill involved in software development, journals dedicated to techniques and tools might not have the same status among scientists as journals that focus on the science. Another problem is that of recognizing maintenance and further development—that can’t be a separate paper for each new version of the software. And yet, without maintenance, software can swiftly fall into disuse.

The Funders’ View of Scientist Developers. Both the NSF and the EPSRC are clear that scientist developers are undervalued. According to the NSF (www.nsf.gov/pubs/2014/nsf14059/nsf14059.jsp), “Scientists and engineers who develop software … for their research spend significant time in the initial development of software … Despite the growing importance of software products, the effort required for their production is neither recognized nor rewarded.” The EPSRC roadmap states “There is a … growing need to recognise the role and value of the research software engineer” (www.epsrc.ac.uk/research/ourportfolio/themes/researchinfrastructure/subthemes/einfastructure/strategy/roadmap/).

There are several ways in which the perceived value of scientist developers might be enhanced. One way is to refer to them specifically in funding calls.³ The FLIMS field study demonstrates that this hasn’t always been the case in the past. The FLIMS principal investigator said in an interview that people are often recruited to develop scientific software, not on the basis of their software development skills, but because they’re perceived as being useful people within their research groups, and their current funding is in jeopardy.⁵ This attitude toward developer appointments is wholly consistent with the view expressed above that “everyone” knows how to develop software.

One consequence of undervaluing scientist developers is the lack of a clear career path. With respect to scientist end-user developers, who are often PhD students or post-doctoral researchers, the traditional career path is PhD → postdoctoral position → tenure (although in recent years, it has become increasingly difficult to achieve the last step of this path).⁶ We noted earlier the primacy of publications to a scientist’s reputation and hence to his or her progression on an academic career path and the difficulty of reflecting the value of software development in publications. The implication of this is that ambitious scientists might be wary of spending too much time on software development.

Implicitly acknowledging this, the EPSRC roadmap explores ways in which recognition of software’s value within academic communities, and hence recognition of its developers, might be achieved independently of publications. Currently, the system for assessing the excellence of research in UK higher education institutions is the REF (Research Excellence Framework; www.ref.ac.uk), which published its results in December 2014 (http://results.ref.ac.uk/). The REF’s findings affect the allocation of government funding to particular institutions for academic research. As Baxter and his colleagues explain,³ the REF allows for the recognition of software deliverables through its system for impact measurement. However, the concept of evaluating software impact on scientific research is relatively new, and it isn’t obvious how such impact might be recognized or measured. This problem is explicitly recognized in the NSF’s “Dear colleague” letter (www.nsf.gov/pubs/2014/nsf14059/nsf14059.jsp), which invites proposals for workshops and exploratory research to investigate the use of metrics for software usage and impact together with novel methods of citation and attribution. The goal here is that the technologists involved in cyberinfrastructure, including the software developers, might be credited for their contribution.

The situation is complicated by the fact that development of scientific software isn’t enough; software has to be maintained to continue to be useful. And software maintenance, even more so than software development, isn’t an activity that’s particularly respected in an academic environment. As Ribes and Finholt say,⁴ “the work of maintenance (while crucial) is often thankless, of low status and difficult to track.”

The vision documents suggest at least two ways in which scientist end-user developers might maintain and extend software without it encroaching too much on the time spent directly on science and building up their reputation in the traditional way. The first is that scientist end-user developers create software with a view from the start of making it robust and maintainable. The Software Sustainability Institute (www.software.ac.uk) is funded by the EPSRC to offer advice and support in achieving this. For its part, the NSF aims to establish scientific software innovation institutes—which it calls “long-term community-wide hubs of excellence”—to provide, among other things, expertise and education in scientific software development. The second way is for scientist end-user developers to make their software open source. The vision documents emphasize the software sharing aspect of the open source model, but shared maintenance and extended development is also supported.

Career scientist developers working in academia are often on fixed-term, project-related contracts. This poses the risk that, having worked hard to understand how the project software might best support the scientists and the science, the developer then leaves, taking away his or her understanding. The EPSRC implicitly acknowledges that solutions to this problem aren’t immediately forthcoming by stating an intention of working with the Software Sustainability Institute to produce a report on “research software engineers.”
Finally, we applaud the commitment stated in both the EPSRC and the NSF documents to enhance the value of scientist developers by supporting training in software engineering and in providing new tools and techniques throughout their careers.

Some Weaknesses in the Documents
In the light of the empirical studies, we identified some weaknesses in the documents which could be detrimental to the choice of projects to be funded, to the execution of such projects, and to the training/education of scientist developers in software engineering.

Using Terms without Elaboration
Two caveats became apparent to us on reading the funders’ documents both concerning the use of terms without elaboration. The danger is that these unelaborated terms might be misinterpreted so that resources for both project funding and training might be wasted.

The documents abound with the context-free use of the first term, “best practice.” Examples include “software engineering professionals are trained in best practices,”13 “all researchers must be exposed to best practices in software development,”14 and “the NSF will invest in activities that … provide new curricula and methods to teach students at all levels and across all disciplines best practices for software engineering” (www.nsf.gov/publications/pub_summ.jsp?ods_key=nsf12113). Nowhere in the documents is it acknowledged that no single set of “best practices” exists in software engineering. In fact, such practices are strongly dependent on the context of development: what works well in one type of software development (such as commercial projects with well-understood needs) might be the wrong choice for another (such as scientific research projects with emergent needs). As Vic Basili and his colleagues noted in their study of the development practices in the high-performance computing (HPC) community, “Several software engineering practices generally considered good ideas in other development environments are quite mismatched to the needs of the HPC community.”15 This quote is especially apposite as people who develop and use HPC form a significant part of the community of cyberinfrastructure users and developers.

The second term is “innovation” and its companion “novelty,” both of which are used frequently in the documents without elaboration. Examples include “supporting innovation” stated as an objective in the EPSRC software strategy and “innovation in software provision” in the EPSRC roadmap. Although “innovation” or “innovative” are generally used in the NSF documents to refer to the science enabled by cyberinfrastructure, they also appear in relation to software, as in “innovative new software elements.” These usages beg the question: innovation of what?

We feel it’s important that the term “innovation” be interpreted broadly. What is vital for e-infrastructure software is that it supports the scientist in the most effective manner, where the exact meaning of the term “effective” depends on the context. The innovation might well lie in the application of well-established methods and techniques of computer science and software engineering to the development of e-infrastructure software rather than in the development of new methods and techniques. If the interpretation of innovation or novelty relates only to the development of new methods and techniques this becomes an important criterion for funding, the pitfalls experienced by the e-Science program might well recur.

Requirements of Scientists as Well as of Science
It’s gratifying to note that both the EPSRC and the NSF see their e-infrastructure programs as being driven by the needs of science. For example, the EPSRC roadmap explicitly states that “the UK will have a computing ecosystem that is … responsive to meet the needs of internationally competitive science,” and the NSF states that “CIF21 is driven by science, engineering and education needs” (www.nsf.gov/cise/ac2/cif2/CIF21Vision2012current.pdf). The problem is that the needs of science aren’t necessarily the needs of scientists, yet both sets of needs must be met insofar as is possible.

Given that it’s widely accepted that scientific software requirements emerge largely in use12, a common overall development plan for scientific software is to release a first version that both provides hands-on users with something of immediate value and is designed with usability in mind, so that users are naturally inclined to deploy it and can then comment meaningfully on its features and design. The FLIMS field study provides an example of how this development plan can be stymied if the two criteria of value and usability are not met, even when use of the software is mandated. In one lab where the use of FLIMS was compulsory, rather than enter the details of an experimental procedure in the expected structured manner, bench scientists instead used a function intended for images and uploaded the relevant Word documents in which they had described their experiments. This action met their obligation to use FLIMS, rather than enter the details of an experimental procedure in the expected structured manner, bench scientists instead used a function intended for images and uploaded the relevant Word documents in which they had described their experiments. This action met their obligation to use FLIMS, but the unstructured data contained in this document was, of course, impossible to aggregate with the structured data already in the database and very difficult to include in searches. These bench scientists clearly didn’t regard FLIMS as useful software in the process of evolution and themselves as active agents of that evolution—rather, they saw FLIMS as a nuisance to be worked around.

Our concern is this: if this point isn’t emphasized in the funders’ documents, proposers and reviewers might not appreciate the importance of meeting the needs of scientists as well as the needs of science. Although the NSF
document does refer to scientists’ requirements as being one of the drivers of cyberinfrastructure development, it never mentions usability. The EPSRC roadmap does, but only peripherally. This lack of emphasis might lead to challenges in executing projects, such as was the case with FLIMS.

A Fundamental Challenge to Collaborations

The raison d’être of e-infrastructure is to support collaborations often, though not always, of scientists. Although “collaboration” is an oft-used term in the funders’ documents, nothing is said about the challenges thereof: they are many, varied, and merit a whole discipline to unravel, understand, and address. The challenge that we focus on here is that posed by the academic reward structure.

The current scientific polemic stresses the benefits of interorganizational, interdisciplinary, and international collaboration—witness the activity centered on CERN and the Large Hadron Collider, the UK e-Science program, and the various cyberinfrastructure-enabled research programs in the US.13 The current reward structure doesn’t match the polemic but is instead centered on competition between individuals—for example, the UK’s REF focuses on individual institutions, researchers, and units of discipline.

This, of course, profoundly affects collaborations. For example, scientists typically face a quandary when it comes to sharing data: Should they share it freely or hide it from others, at least for a time, to give themselves a competitive advantage? This quandary might lead to difficulties in developing the underlying e-infrastructure. This was the case in FLIMS, where the project manager found it hard to elicit the access requirements that the lead scientists desired for their data.7 In some cases, the reward structure goes so far as to deter scientists from forming collaborations. For example, collaborations frequently lead to multi-authored papers. Ruth Müller describes the perception of postdoctoral researchers in the life sciences that first- or single-authored papers are essential to their finding permanent positions.14 According to Muller, this leads researchers to believe that the smart way of working is “alone if possible and collaborating only if necessary.”

We would have welcomed some sort of acknowledgment of this anomalous situation, encouraging collaboration while rewarding individual effort, in the funders’ documents.

Given our analysis of the funders’ documents, one suggestion for future work is to encourage more empirical studies of scientific software developments and of scientists using software. Such studies might illuminate how requirements for both scientists and the science can best be established, how scientists might best be supported by software, and how the various established methods of software engineering might be matched with the various contexts of scientific software development. We’re heartened by the move toward interdisciplinary empirical studies to investigate further the impacts of the cultural, social, and organizational context of software infrastructure development on that development. An example of this is the establishment of the NSF’s Science of Science and Innovation Policy (SCISIP; www.nsf.gov/funding/pgm_summ.jsp?pims_id=501084), which has the overall aim of improving the science policy decision process based on evidence from empirical studies.

We believe there remains a major obstacle to enabling collaborations of scientists and programmers to develop effective e-infrastructures and to fulfilling the potential of such e-infrastructures in facilitating profound scientific innovations. This obstacle lies within the current culture in which scientists work (in the Western world at least). Although both funding bodies recognize the need to address one aspect of this culture, the undervaluing of scientist developers, neither grasps the nettle of another aspect, that of the reward structure being based on individual achievement in a single discipline rather than collaborative achievement in multidisciplinary settings as encouraged by current scientific rhetoric. We strongly suggest that the funding bodies should seize all available opportunities to discuss and address this issue with other interested parties.

References


Judith Segal is a visiting research fellow at The Open University. Her research interests include the organizational and contextual aspects of scientific software development. Contact her at Judith.Segal@open.ac.uk.

Marian Petre is a professor of computing at The Open University. Her research interests include expert behavior and reasoning in the design of complex systems. Petre has a PhD in computer science from University College London. She held a Royal Society Wolfson Research Merit Award, in recognition of her research on the nature of expertise in software design, and her empirical studies on reasoning and representation in software development. Contact her at m.petre@open.ac.uk.

Helen Sharp is a professor of software engineering at The Open University. Her research interests include the study of professional software practice with a focus on human and social aspects of software development. Sharp is the associate editor for Transactions on Software Engineering; a member of the IEEE Software advisory board; a member of ACM, IEEE Computer Society, and the British Computer Society; and a Higher Education Academy (HEA) Fellow. Contact her at h.c.sharp@open.ac.uk.

Selected articles and columns from IEEE Computer Society publications are also available for free at http://Computing-Now.computer.org.