Some Challenges Facing Scientific Software Developers: the Case of Molecular Biology

Chris Morris
Computational Science and Engineering Department
Science and Technology Facilities Council
Daresbury Laboratory
Near Warrington
WA4 4AD
UK
Correspondence to: chris.morris@stfc.ac.uk

Judith Segal
Department of Computing
The Open University
Walton Hall
Milton Keynes
MK7 6AA
UK
j.a.segal@open.ac.uk

Abstract

It is apparent that the challenges facing scientific software developers are quite different from those facing their commercial counterparts. Among these differences are the challenges posed by the complex and uncertain nature of the science. There is also the fact that many scientists have experience of developing their own software, albeit in a very restricted setting, leading them to have unrealistic expectations about software development in a different setting. In this paper, we explore the challenges facing scientific software developers focusing especially on molecular biology. We claim that the nature and practice of molecular biology is quite different from that of the physical sciences and pose different problems to software developers. We do not claim that this paper is the last word on the topic but hope that it serves as the inspiration for further debate.

1. Introduction

Advances in science are becoming ever more reliant on software, used for instance to simulate complex processes which are impossible to investigate in a lab (the effect of a nuclear explosion, for example) or to aggregate vast amounts of data from many different sources in order to search for patterns and generate hypotheses. However, there is a widespread feeling among scientific software developers that the challenges of developing such software remain largely invisible to the scientists who use or commission such software (Segal, 2007, Segal, forthcoming). In our conversations with such developers, we have heard many express regret that the software they deliver, despite their best efforts, does not optimally support their scientist users. They ascribe this in part to the fact that scientists, especially those who commission or fund the development, don't understand the challenges.

Two examples are frequently cited. The first is that, according to many scientific software developers, scientists don't realise that funding is necessary to maintain software after initial delivery for the following reasons: infrastructure software such as web browsers and operating systems change over time necessitating changes in application software; and the delivery of the software often opens up new avenues of scientific enquiry and hence new requirements emerge. The second example is that many scientific software developers feel that scientists don't realise that software developed at great expense in one project might be reused at little cost in others provided funding is made available to ensure that the initial software meets the criteria of robustness and generalisability.

The aim of this paper is to explore and articulate the challenges inherent in scientific software development and in so doing, to raise their visibility among scientific software developers, scientists in general and molecular biologists in particular.

To this end, in section 2 below, we discuss some challenges which we think are generic to all scientific software developers and in section 3, consider molecular biology. The practice of molecular biology has characteristics which set it apart from the physical sciences. For example, molecular biology has typically been studied by different sub-disciplines applying different tools, standards of evidence etcetera to the same biological entity. In addition, molecular biology

is heavily context based and thus does not lend itself easily to generalizations or abstractions (Knorr Cetina, 1999). Such characteristics have a significant impact on developing molecular biology software as we shall discuss in section 3. We should emphasise here that section 3 is, we think, highly novel: we have not seen the challenges articulated herein discussed elsewhere. Thus this section should not be regarded as the last word on the topic but rather as an invitation for molecular biologists and developers of molecular biology software to explore the topic further.

In section 4, we discuss where we go from here. We have begun to articulate the challenges: how do we go about addressing them? We discuss a current initiative, but again, intend this section to act as a jumping-off point for debate and an invitation to others to get involved.

2. Generic challenges

In this section we describe the challenges that are common to most, if not all, scientific contexts.

2.1. Some challenges associated with the traditional model of scientific software development

Traditionally, scientific software has been developed by a scientist (or scientists) in order to address the specific current problems of either the developer him/herself or that homogeneous group of scientists in which the developer is embedded. In her field studies of a variety of scientists developing software in this context, Segal (Segal 2005, 2007, 2009) identified a widespread model of software development, which, while it is very successful within the traditional context in which it is usually deployed, does not generally result in software which may be used safely outside of such a context. That is, the developers of such software are not generally concerned with how it might be deployed to address problems other than those for which it was originally intended and so:

- they pay little attention to the comprehensibility of the code, thus making it difficult to extend and maintain.
- they test the code only with respect to the specific problem they are addressing, and hence the reliability of the code in different contexts is not assured.
- they are not aware of the challenges associated with the issue of negotiating requirements in a diverse community.

Providing the software is only used in the context for which it is developed, the developer(s) lack of attention to these concerns poses no problems. Problems emerge, however, when the software is deployed outside this context, for example, when the software 'escapes', as it were. In this (not uncommon) situation, a piece of software developed according to the traditional model is observed to be successful at addressing a particular problem and is appropriated and possibly modified in an ad hoc manner by other scientists in the same group to address other problems, and then by scientists outside that group, and so on, while all the time such robustness and efficacy that it had in its unmodified state is only assured within the original context of use. Related to this is the problem of legacy software. Legacy software is a means by which an older generation of scientists might pass down (at least some of) their encapsulated knowledge to a younger. However, such software is generally written according to the traditional model, so its use outside the context for which it was developed is, as described above, inherently risky.

The challenge here is raise the awareness of the scientist-developer both of the risks involved in traditional development and of the tools, methods and technologies which might mitigate such risks. This challenge is made more difficult by identifying which of the myriad available tools, methods and technologies are helpful in the specific scientific context in which a particular scientist-developer works. It is hoped that the proposed network, ISS-net, see section 4 below, will address this challenge by exploring the fit between methods, tools and technologies on the one hand and scientific contexts of use on the other.

2.2. Some challenges associated with multi-disciplinary development teams

By 'multi-disciplinary teams' in this context, we mean teams consisting of scientists and scientist-developers, and also of professional software developers to whom the science is just another application domain. Such multi-disciplinary teams are essential when the software is intended to support a community of scientists with a wide range of problems, or where the complexity of the software needed is beyond the development capabilities of a scientist.

As with any multi-disciplinary team, there are challenges of communication across the disciplines, but there are additional challenges specific to the fact

that some members of the team are scientists. It is very likely that such scientists have experience of the traditional model of scientific software development, as discussed in 2.1. above. Field studies (Segal 2008a, 2008b, 2009) have indicated that scientist-developers frequently do not recognize that the success of this model is dependent on a specific context in which the developer is a potential user and where the scientific problem to be addressed is of the moment, and try to apply the model, consciously or unconsciously, in other contexts.

Often, and in our view, rightly, a scientist is in overall charge of a scientific software development project. Segal records several instances of scientists in such a role not being aware of the issues of negotiating requirements from a heterogeneous user community or of testing or of developing the software so as to be maintainable. As discussed in 2.1. above, these are not salient issues in the traditional software development context. This lack of awareness results in clashes of expectations between scientists and professional software engineers as to the amount of resources necessary and the nature of the skills required to enable a successful scientific software development.

Typically, the scientist, used as he/she is to the traditional context of scientific development, expects any such development to require less time and fewer specialised skills than does the professional software developer. One author has the experience of making time estimates for development tasks that are typically three times greater than the estimates made by a senior scientist with extensive experience of the traditional development model.

Additional challenges occur when the developers include computer scientists, as is frequently the case for research-council funded e-science projects. Here there is a potential clash between the aims of the computer scientists – to explore novel uses of computers – and the end-user scientists for whom the software is merely a black box and is judged solely in relation as to how well it supports the science.

2.3. Some challenges of project management in multi-disciplinary scientific software developments

In 2.2. above, we noted that multi-disciplinary teams of scientists, scientist-developers and professional software developers, are frequently under the management of a scientist. We also discussed the potential clash of expectations between the scientists and professional software developers regarding the resources to be deployed. Other challenges may arise from the incongruence of certain concepts, such as 'project success', 'project management' and 'authority'.

- A scientist will deem a scientific software development project a success if it demonstrably advances the science irrespective of whether it delivers the software exactly as in the project proposal. To a professional software developer, however, a scientific software development project is a success precisely if it does deliver the software exactly – or almost exactly - as in the project proposal.
- To a scientist, project management for such a project is focused on advancing the science; to the professional software developer, it is focused on advancing the software development.
- The scientist uses his authority to advance the science; the professional software developer expects authority to be deployed in the interests of advancing the software development.

Such incongruences in the understanding of roles and concepts can lead to complete breakdowns in working relations (Segal, forthcoming)

2.4. The challenges of retaining scientific software developers

Successful scientific software developers are a rare breed. Whether they come from a scientific or a professional software development background, in order to be successful they have had to learn hard lessons about communication and collaboration across the disciplinary boundaries of science and software engineering. Given the increasing reliance of scientific advances on scientific software, one might expect that such rare and valuable beasts would be fostered and nurtured so as to encourage them to stay in post. Such does not appear to be the case.

The scientific community values scientific advances (of course) and the publications which report such advances, whereas the effort, knowledge and skill which goes into the production of the software enabling such advances remains, to a large extent, invisible. Segal (2007) noted in her field studies the pervasiveness among some scientific communities of the assumption that 'anyone can develop software'. A couple of dangers associated with this assumption are, firstly, that scientists may be appointed to software development teams solely on the basis that they are good scientists in need of funding without any reference to their software experience and skills, and secondly, that funding agencies may provide only enough funding to employ professional software

developers at the beginning of their careers (Segal, forthcoming).

As to software developers who have a background in the specific science, if they wish to pursue a career in scientific research, they would currently be ill-advised to focus on software development. One might get a single publication from a software development describing the functionality of the software, but in this publication-led era, a single publication in the quite lengthy time period needed to develop the software is not likely to enhance one's scientific career.

This problem of the lack of visibility of software development among scientists was recognized by Star and Ruhleder almost 15 years ago (Star and Ruhleder, 1996) in their comprehensive study of the (lack of) adoption of an infrastructure package to support the community of scientists studying the model organism c. elegans. The following comment probably still holds true today:

> "The difficulty is that there are no clear rewards for this kind of work [scientific software development], except for the contributions the tool makes to one's own work. The biologist working with the computer scientist doesn't get any "credit" for this within his own discipline…" [Star and Ruhleder]

3. Challenges specific to molecular biology

Almost since the very inception of computers, molecular biologists and the crystallographers with whom they work closely have availed themselves of this powerful tool. For example, Dorothy Crowfoot (later Dorothy Crowfoot Hodgkins), used a computer in her work on the structure of B12 in the late 1940s (Wolpert and Richards 1988).

The current software tools used successfully by molecular biologists have been developed both by scientist-developers, where challenges include those described in 2.1. above, and by multi-disciplinary teams of scientists and software developers, as in 2.2. An example of the former is the CCP4 suite of programs for protein crystallography which encapsulate much current crystallography knowledge, see http://www.ccp4.ac.uk and (CCP4, 1994) An example of the latter is the social web site myExperiment, http://www.myExperiment.org,which provides a virtual area where scientists with the same interests can safely share workflows and other objects.

In what follows, we firstly discuss the challenge of choosing between the traditional and the multi-disciplinary models of software development. We then explore some characteristics of the discipline of molecular biology and its associated practices, and the impact of these on software development.

3.1. Choosing between a traditional and a multi-disciplinary model of software development.

One challenge facing molecular biologists when initiating a project in which software development forms a part, is to determine whether to opt for a multi-disciplinary development or for the more traditional model of 2.1. At first sight, it might appear that this choice is simple, depending only on whether the software is to support "big" science, e.g. work involving large amounts of data such as high throughput work in protein science, or "small" science, such as hypothesis-driven investigations. In the former case, the obvious choice is to deploy multi-disciplinary software development teams; in the latter, it's to use the more traditional, cottage industry, model.

We argue here that what appears to be simple and obvious is, in fact, far from being so. Instead, we argue that although the debate about the relative contributions of big and small approaches to science is sometimes intense (Steitz, 2007), these approaches are essentially complementary, and so are the multi-disciplinary and traditional models of development needed to support them.

Let us firstly consider the complementary natures of big and small science. Big science, especially in the field of molecular biology, is typically associated with accruing and analysing large data sets. Small science is typically associated with hypothesis investigation in some sense. From the big science, for example from visualisations of large data sets, can arise the hypotheses for more detailed investigation in the small. In addition, data mining methods on large data sets can serve to identify methods which are more likely to be successful than not when applied to (for example) the expression of proteins.

As to small science, there will always be the scientist who is motivated to pursue individual or small group research by the challenge of standing apart from the common herd (as it were). For example, James Sumner, who isolated urease, said in his Nobel Lecture

> "A number of people advised me that my attempt to isolate an enzyme was foolish, but this advice made me feel all the more certain that if successful the quest would be worthwhile" (James Sumner, quoted in Tanford and Reynolds, 2004)

From small science, there often come results which have profound effects on the advance of big science. For example, the Human Genome Project, the epitome of a big science project, would not have been possible without the small science of Watson and Crick.

We claim that, as with big and small molecular biology, the multi-disciplinary and traditional models of software development are likewise complementary. In big science, the collection and storage of big data sets requires the reliability provided by the tried and trusted tools and methods of software engineering and hence necessitates the deployment of a multi-disciplinary team involving both scientists and professional software developers. However, the deployment of such data, its analysis with respect to a certain research question, might well be done better using software developed according to the traditional cottage industry approach. On the other hand, the molecular biologist working in the small using software she/he has developed using the traditional model, is very likely to make use of GenBank, the NIH genetic sequencing database, http://www.ncbi.nlm.nih.gov/Genbank/index.html, which is, no doubt, maintained in part by professional software developers using software engineering methods.

Thus, what looked like a simple equation (big science projects = multi-disciplinary software development teams; small projects = traditional cottage industry development) is, we feel, more subtle than first appears, and challenges the principal investigators of molecular biology projects to identify the right mix of developers.

3.2. Molecular biology as a set of sub-disciplines

Traditionally, molecular biology has been practised as a set of sub-disciplines, each with a different perspective on the same biological object and each centered on different techniques. For example, Tanford and Reynolds (2004) describe protein scientists in the 1930s as being a very diverse group of people, with assorted scientific backgrounds and totally different ways of working, but note the acceptance of this diversity:

> "A common bond was a tacitly agreed permissiveness - *carte blanche* for whatever your vision to future progress might be."

Until very recently, molecular biologists were, as in the quote above, content on the whole to focus on one sub-discipline, using the associated tools and software of that sub-discipline, for example, of protein crystallography, nuclear magnetic resonance or electron microscopy. There has traditionally been little collaboration between practitioners of the different sub-disciplines and hence little need for data sharing or for software which spans sub-disciplines.

There are problems with this separation as the following anecdote illustrates. It was told to one of the authors at a conference of structural biologists working for pharmaceutical companies. The two participants in the story were a crystallographer and a medicinal chemist, and they were discussing the binding of a small molecule to a protein. The crystallographer said that the electron density shows that the small molecule is folded twice, in a chair shape. The medicinal chemist said that the energetics require that the small molecule is planar. Since neither understood the evidence on which the other made his judgment, no agreement could be reached.

There is some evidence that this state of affairs is changing, see, for example, the discussion about extending structural approaches to cellular scale in Harrison (2004) and the emerging focus on systems biology. There is thus the challenge to develop software which joins up the software associated with separate sub-disciplines in order to support a more joined-up view of biology. Such software should at least enable seamless data transfer and thus support, for example, the input of results from structural into systems biology.

A recent initiative to address this challenge is INSTRUCT, see http://www.instruct-fp7.eu, a pan-European project aiming to provide infrastructure for integrating sub-disciplines in structural biology. At the time of writing, INSTRUCT is at an early stage and it remains to be seen how far it succeeds in its aim.

3.3. The heavy dependence of molecular biology on context

One characteristic of biology as compared with physical sciences is its heavy dependence on context. As a simple example, the same protein has completely different behaviour under different circumstances, e.g. in different organs of the human body. One implication of this is that biology, unlike for example physics, does not lend itself easily to abstractions or generalisations. For example, Howard Temin, who received a Nobel Prize for the discovery of reverse transcriptase, said:

> "Intellectually I felt that the central dogma was true, but that it didn't explain my results ... Since this is biology, I didn't have any

philosophical problems with my results being an exception – biology doesn't have the force of physics." (Temin, quoted in Kevles, 2008)

Traditional software engineering, however, focuses on abstractions and generalisations. The successful development of the Taverna workbench and myExperiment web-site (Goble and De Roure, 2007) is somewhat at odds with this tradition. De Roure and Goble, both computer scientists, describe how they first of all implemented software to meet the specific needs of specific groups of biologists, and then reflected on how these needs and hence the software might be generalised (De Roure and Goble, 2009). This bottom-up approach, as opposed to the more traditional top-down approach of software engineering, led to the production of software which proved very successful in supporting biologists.

3.4. Concepts and terms in molecular biology

Knowledge gained and shared between research scientists is largely liminal – at the threshold of the science – and tends to be unstable, difficult to represent and codify. This presents a particular challenge in the case of molecular biology. Given both the separation of the discipline into sub-disciplines and the difficulty of generalisation as described above, molecular biologists are content to accept that the same term might refer to different concepts in different contexts, and conversely, that the same concept might be described by different terms.

By way of illustration, here are a couple of examples. The first is that one of the authors of this paper, an algebraist by background, was shocked to find that a particular protein name did not uniquely identify a protein. The second is that the term 'construct' is used differently even by protein laboratories doing very similar wet lab work.

This lack of a one-to-one mapping between concepts and terms poses clear challenges to the software developer. For example, in the latter case, a field labelled 'construct' in a database might contain data with quite different meanings and import according to the lab in which the data depositor works.

4. Discussion

In section 2 above, we discussed some challenges facing scientific software developers in general, and in section 3, considered some challenges specific to molecular biology. The obvious question which now arises is: where do we go from here?

Let us consider the generic challenges first. It is clear that the context in which scientific software is developed is, in general, different from the context of commercial software development, see, Carver et al (2007), Segal (forthcoming). For example, the identification of requirements is entirely different. In the scientific context, requirements are largely emergent whereas in the commercial context, they are largely pre-specified. Also different is the practice of software testing, heavily dependent on the gut instincts of the scientist as opposed to being methodical and systematic.

Those methods, techniques etcetera of software engineering which are commonly referred to as best practice have largely been identified in the context of commercial development, and applying them willy-nilly to scientific developments can be very detrimental (Segal, 2005, Segal and Morris, 2008).

In the authors' opinion, the following is highly desirable:-

- a taxonomy of different scientific development contexts (for example, the heavily computational simulation models used in climatology; the heavily data intensive software used in molecular biology)
- a taxonomy of the tools, techniques and methods identified in software engineering
- a mapping between the two, so that each development context is associated with the most appropriate tools, techniques, methods.

This is the main aim of a network, ISS-net, (ISS = Improving Scientific Software) being proposed by the authors of this paper and others. Should any readers of this paper be interested in contributing to such a network, the authors warmly invite them to get in touch.

As to the challenges specific to molecular biology, so far as the authors are aware this paper is the first to address them explicitly. Given this fact, we recognise that section 3 is probably far from comprehensive and possibly quite contentious. Our intention in putting forward our ideas is not to argue that they are the last word on the topic but rather to foster debate, and to raise awareness of the challenges of software development for molecular biology among scientific software developers and molecular biologists. It is to be hoped that this paper will provide a useful jumping-off point for such debate.

5. References

"The CCP4 Suite: Programs for Protein Crystallography". *Acta Cryst*. D50, (1994) 760-763

Carver, J., Kendall, R., Squires, S. and Post, D., 2007, "Software Development Environments for Scientific and Engineering Software: A Series of Case Studies." *Proceedings of the 29th International Conference on Software Engineering*. Minneapolis, USA. May 23-25, 2007. p. 550-559.

De Roure, D., Goble, C., 2009, 'Software design for empowering scientists', *IEEE Software*, 26(1) pp 88-95.

Goble, C., De Roure, D. (2007) 'myExperiment: social networking for workflow-using e-scientists'. In: *Proceedings of the 2nd workshop on Workflows in support of large-scale science*, Monterey, California, USA.
http://eprints.ecs.soton.ac.uk/15095/

Harrison, S. C, 2004, Whither structural biology? *Nat Struct Mol Biol*, 11(1):12-15.

Kevles, D.J., 2008, 'Howard Temin: Rebel of Evidence and Reason', in *Rebels, Mavericks and Heretics in Biology*. Harman O.S., Dietrich M.R. (eds), Yale University

Knorr Cetina K, 1999. *Epistemic Cultures: How the Sciences Make Knowledge.* Harvard University Press, 1999

Segal, J, and Morris, C., 2008, 'Developing scientific software', *IEEE Software*, 25(4), 18-20

Segal J., 2005, 'When software engineers met research scientists: a case study', *Empirical Software Engineering,* 10(4), 517-536.

Segal J, 2007, 'Some problems of professional end user developers', VLHCC, IEEE Symposium on Visual Languages and Human-Centric Computing, 2007, pp111-118

Segal J, 2008a, 'Models of scientific software development', *SECSE 08*, Workshop on Software Engineering in Computational Science and Engineering, workshop co-located with ICSE 08, Leipzig, Germany. http://www.cse.msstate.edu/~SECSE08/Papers/Segal.pdf

Segal, J, 2008b, 'Scientists and Software Engineers: A Tale of Two Cultures', *Proceedings of the Psychology of Programming Interest Group, PPIG 08*, Lancaster UK, pp 44-51, ISBN 978-1-86220-215-3

Segal, J. 2009, 'Some challenges facing software engineers developing software for scientists', 2nd International Software Engineering for Computational Scientists and Engineers Workshop (*SECSE '09*), ICSE 2009 Workshop, pp 9-14, doi: 10.1109/SECSE.2009.5069156.

Segal, J, forthcoming, 'Software development cultures and cooperation problems: a field study of the early stages of development of software for a scientific community' , *Computer Supported Collaborative Work*

Star S. and Ruhleder K, 1996, 'Steps towards an ecology of infrastructure design and access for large information spaces', *Information Systems Research*, 7(1), 111-134

Steitz, Thomas, 2007, 'Collecting butterflies and the protein structure initiative: The right questions?, *Structure*, 15(12):1523–1524

Tanford, C, Reynolds, J, 2004,. *Nature's Robots: A History of Proteins* Oxford University Press.

Wolpert, L, Richards, A., 1988, *A Passion for Science*, Oxford University Press.