e-Science Application Development: Analyst and User Perspectives

Sarah Thew[1], Rob Procter[2], Alistair Sutcliffe[1], Colin Venters[1], Iain Buchan[3]

[1] Manchester Business School, University of Manchester, Manchester M15 6PB, UK

[2]National Centre for e-Social Science, University of Manchester

[3] Northwest Institute for Bio-Health Informatics (NIHBI)

Sarah.Thew@manchester.ac.uk

Abstract Potential users of e-science software often have strong technical skills, ranging from the use of scripting languages to the development of substantial pieces of software used by other members of their communities. In this paper we reflect on the experience of designing and developing software with a technically expert end user, and consider how the software development process might be structured to take advantage of the expertise of such users.

Introduction

A number of requirements challenges associated with the UK e-Science programme have been identified (Beckles, 2005; Zimmerman & Nardi, 2006). In particular, much has been made of the challenges of managing the relationship between domain scientists and computer scientists, with respect to enabling developers to understand users' scientific practices; and users to grasp the possibilities offered by new technologies (Voss, Proctor, Jirotka, & Rodden, 2007). Achieving this interdisciplinary understanding can be complicated by tensions (real or perceived) between the domain scientists and computer scientists, as each tries to ensure that the project provides the necessary space to pursue their respective research objectives. In this paper, we reflect on the experience of participating in an e-Science project from the perspectives of the software analyst; and an expert end user. Contrary to expectations, we find that, although the problems encountered can be attributed to a lack of shared understanding, they do not always fit the usual stereotypes.

There has been much discussion about the practices of end user developers (Taylor, Moynihan & Wood-Harper, 1998) and the design of software environments to support such users (Fischer & Giaccardi, 2004), but relatively little research into the involvement of such technical experts as 'users' within software development projects. The boundary between technical experts and early adopters is often blurred, particularly in fields such as bio- and health-informatics, arguably, it is this technical sophistication which enables early adopters to grasp the opportunities that e-Science offers more quickly. This technical experience ranges from the use of scripting languages to tailor applications to individual needs to the development of substantial software used by other members of their domain. This sophistication is welcome in that it might be expected to accelerate the formation of the all-

important ‘shared understanding’ referred to above. Our experience, however, is that it presents its own challenges, of which collaborators should be aware.

A previous case study (Segal, 2005b) reflected on the problems of scientific users who have their own software development techniques, trying to adapt to a different approach used by a software company. In this paper, we reflect on experiences of working on a project involving early adopters who are used to developing their own software in their own right, and consider the implications of our findings for software development practices in e-Science. We focus, in particular, on requirements gathering and user engagement. We discuss the experience of the requirements elicitation process with an emphasis on the experiences of working with a technically aware, early adopter. The findings provide an interesting counter point to the somewhat stereotypical assumption that ‘being understood’ is a more of a challenge with respect to users than designers (cf. Bowers & Pycock, 1994).

The case study

The ADVISES (ADaptive VISualization of E-Science) project is developing tools to support the use of geographic visualization in epidemiology and public health decision-making. The project team includes a requirements analyst (one of the authors) with a background in molecular biology and health informatics and a software developer, and involves two epidemiologists, both of whom regularly write statistical scripts. One of these users (the lead user) has extensive software development experience, having written a statistical software package for biomedical researchers.

It was quickly established that the other epidemiologist, while happy to contribute when asked, was more difficult to engage in the requirements elicitation process. When discussing epidemiological terms or calculations he was happy to argue his case but was inclined to defer to the lead user to make software design decisions. Hence, more often than not, the requirements gathering process was driven by the ideas of a single visionary user.

Methodology

The ADVISES approach to requirements analysis is based on scenario-based design (Carroll, 2000) and user-centred requirements engineering (Sommerville and Sawyer, 1997; Sutcliffe, 2003), both of which advocate the use of scenarios, storyboards and prototypes in an iterative cycle of requirements elicitation, design exploration and user feedback. Scenario-based design (SBD) was chosen in view of the often volatile and complex requirements of e-Science applications. As research practices often change as an investigation evolves, requirements can become a moving target, which is particularly true in the rapidly developing field of bio-health informatics. SBD is well suited to such circumstances because of its iterative approach which facilitates user-developer dialogue. This rapid iterative approach to development is shared by rapid application development methods (e.g., DSDM, 1995) and, agile software development methods (Beck, 1999).

Meetings between project team members were recorded and we draw on this material as a resource for reflecting on the conduct of the project and user engagement in particular.

Findings

It quickly became apparent that having a shared language and background enabled the quick development of a rapport between ADVISES project team members. The analyst was able to

use paradigms of data-driven hypothesis generation drawn from one domain (gene expression research), to facilitate discussions of similar issues within epidemiology. Similarly, the users involved were able to exploit their understanding of the language of computing and practical skills to be precise about their needs and to converse directly with developers using software terminology. However, development of a rapport was also facilitated by a shared outlook: a willingness to innovate and an interest in each others' domains, which transcends background and which we would argue is equally as important as shared domain knowledge.

There were other benefits associated with working with technically skilled end users. The lead user has a long term vision for the software and clear ideas of what he needs. It is clear that his technical knowledge and awareness of developments in computing influence this vision, for example, he was able to recognize the potential ways an ontology of epidemiology could be reused on other projects and to consider how the application might be integrated with other tools and web services he uses. Working with a user with a strong technical background can benefit the requirements process in other ways – the concept of decomposing processes and data came very naturally to the lead user, whilst his technical awareness was of use to the development team, for example, in directing us to useful web services and suggesting alternative GIS platforms to investigate.

Perhaps the most significant feature of working with such a user when developing new software is a willingness to innovate and take risks. The lead user frequently expressed both a willingness to cope with a less than perfect implementation in return for functionality, for example, suggesting that the software developer need only give him a command line interface, and made suggestions about unusual ways the software developer could get the application up and running:

> ".. like a Facebook plug in that actually does mapping or leverages something else that is so obviously 80% built and the 20% doesn't matter as much as getting 80% of one thing interacting with 80% of another thing."

However, it is apparent that the lead user does recognise the value of the requirements engineering activity and user engagement:

> "The best software I've ever seen is written by people who really understand what's trying to be accomplished by the user"
>
> "If they're (the developer) in the System Administrator mindset where users are an evil enemy to be controlled and kept at bay with strict rules, never actually seen face to face, that doesn't work, you've got to want people to enjoy your software and use it."

The lead user admits to being very impatient and 'wanting things yesterday'. This is understandable given that in other circumstances, he can just write the code he needs to solve a problem himself. Consequently, he expresses a strong preference for engagement via prototyping and rapid application development, and in other projects has adopted some of the practices of 'Agile' development (Beck, 1998) and 'extreme pair programming'. In contrast, the ADVISES project has (due to illness) been slow to produce prototypes, and this has proved frustrating to this user.

The engagement of a lead user with a strong vision provides excellent opportunities to support innovation and to develop novel software, and the lead user believes that this type of engagement is the best way to innovate:

> "I do not think the requirements activity should be weighed down by focussing too much on luddite users, or trying to do everything for everybody. I think we should pick those who are running fast and want to innovate and get into their heads."

However, the process requires a substantial time investment from a single lead user, and this can be problematic when the user is very busy. Furthermore, the lead user is clearly an 'early adopter' who is technically at ease and is keen to push the software to its limits to further his own research goals. Consequently, there is a risk of developing something which is not acceptable to more conservative or less technically confident members of the community. Within this project, for example, it was apparent that the lead user had different attitudes around collaboration and sharing and storage of data to other users interviewed, and it is the software development team's responsibility to find a balance between innovation and acceptability to other users.

There are also risks associated with our lead user's adoption of 'developer' behaviours. Characteristic of requirements meetings was a tendency to jump straight to solutions without reviewing the problem. This made it harder for the design team to develop a clear understanding of the domain and the use to which the software would be put. It was apparent also that the most technically proficient of the users has been uncomfortable with some of the ethnographic methods adopted by the requirements analyst. He commented that he found being observed whilst working difficult and was concerned about showing the requirements analyst the "wrong" things.

Perhaps the most concerning aspect of working with users with software development experience is the danger of the user modifying their requirements because of assumptions about technical limitations. In this case study, the lead user articulated clear requirements which, when examined in detail, were sometimes guided by concerns about potential technical difficulties, rather than being their ideal solution to the problem. In the example below we see the lead user making a design decision – to tab between two displays, based on his understanding of technical constraints. He states this as a requirement and only when pressed as to *why* the tabbing is necessary does it become apparent that this is because he is concerned about the difficulties of updating two yoked panels concurrently. His real interest was in understanding how the changes in one set of parameters (displayed in one panel) affected the overall data distribution (displayed as a graph in the second panel).

> **Lead User:** This is the same mind process really.. maybe flip or toggle between the histogram or the box and whisker. And the box and whisker might be drawn the other way, I've just done it.
>
> **Project Leader:** Sure, so it's there.. so really the question is, how do you get 4 or 5 of those to appear? Now from what I was just hearing you wanted to have something underneath here, a little box and you enter some number which gives you some percent of the range.
>
> **Lead User:** Err yeah.. it's usually done by the number of categories. And these then go away, you've selected 5, it calls the R process to get the quintile cut points and starts to draw on there where the quintile cut points are. It then goes and calculates and produces a sub array of this bit of the distribution, and goes and calculates the mean, standard deviation, and confidence intervals for there. You could actually overlay the box and whiskers over here couldn't you?
>
> **Project Leader:** Ok, yep, I'm with you.
>
> **Lead User:** But that's a frequency.. no, no it's conflating the scales at the point, you really have to toggle between them.

Requirements Analyst: So do you actually want to toggle between the two, or do you want to see the two alongside each other? Is it helpful to be flicking backwards and forward, because there's no reason we can't display them alongside?

Lead User: [The Project Leader's] windows let the user put them where they want. Yeah that's fine, but if you change the number of categories here, then there needs to be a back end connection to update this.

Project Leader: Yeah, so these are both closely coupled or yoked, so basically there's a little control here.

Requirements Analyst: Yes.

Lead User: Oh ok.

When asked about this incident at a later date, the lead user agreed that it was his technical knowledge that had led him to suggest a particular design, rather than a focus on the problem:

Lead User: "Where you have different windows overlaid on one another and they're part of the same application. Getting them in the right order so that one doesn't suddenly come onto of another one that's interacting with user, that's getting the z-order right. You'd think it would all automatically be done but I've seen so many programs where rearranging a window suddenly the z order goes wrong! I just didn't want to go down that path."

While it is sometimes the case that a user's ideal solution cannot be implemented as it was first conceived because of technical limitations, the exploration of possibilities will certainly lead to a better understanding of the problem and may result in alternative and better designs.

A previous study of a software development for technically proficient users (Segal, 2005b) discovered a mismatch in expectations of the development process between users and developers, and indeed investigations of end-user software developments (Taylor et al., 1998) reveal that such projects generally proceed in a very different manner to traditional software development projects. There is a strong emphasis on evolutionary development with the application emerging through trial and error. Formal software engineering approaches are rarely followed, and indeed there is some evidence to suggest that many users don't particularly value the skills of software development (Segal, 2005a). The lead user expressed a preference for developing via pair programming, whilst his description of his own software development approach recognises that he strongly favours a 'trial and error' approach in his own software developments:

"I'm a typical hacker, I need to play, I need lots of play and lots of input, so I will type something quickly at the command, and if it goes wrong I'll do it again rather than hold back and ponder and wait until it's perfect and then click 'enter'. It's an iterative process."

When being a 'user' rather than a developer, the lead user made use of his software expertise and requirements discussions tended to be very technical and focused on solutions. He often makes implementation suggestions and uses computing jargon, for example, the use of an R server below and the use of 'pointers'.

Lead User: It's a very simple rule base to find it. If you've got 5 categories, if you line everything up in order where are the cutpoints?

Requirements Analyst: Ok so it's just working it out based on the range?

Lead User: On the values. Because if there's a repeated value, if there are 10 1s.. So it's not as trivial as you think. Now I've got that algorithm written in VB of all things. It's very commonly available but if you've got something that's web serviced, why not have

R, just load it onto the system, it's just a command in R, if you've got an R processor running.

Requirements Analyst: That's what I'm hoping we can get set up, we've used that successfully on past projects, it seems very reliable.

Lead User: Well..!

Requirements Analyst: Well, depends on the box you've got it on! But it seemed to behave.

Lead User: No, no.. it's got better, there used to be a lot of bugs in R. It behaves itself, it doesn't fall over very often, but it does make mistakes occasionally, but not with simple things like this… So we're finding those cut points and then your.. erm.. [long pause] you create for example 5 sub arrays, probably pass a pointer to the cut points in a sorted array, and you get the mean, standard deviation and 95% confidence interval, for the mean. You might want to make that [the confidence interval] user selectable.

Similarly, when asked to explain the data he would need to create a map he was naturally inclined to draw out this explanation as a class diagram. He is keen to provide technical advice and direction to the project:

"You saw I gave quite a detailed response at the weekend to [the software developer] on where to go to find the functions. What I was trying to do was to send him a message that this has already been done."

However, the project structure made it difficult for the lead user to engage in direct discussion with the developer – most user requests and suggestions came via the requirements analyst, and opportunities for the developer and users to meet were rare. We would suggest this indirect communication is a source of potential problems when working with technically expert users. If developers have the expectation that users should provide problems and the developers should make implementation decisions, they may not be comfortable receiving technical direction from users. Furthermore, if developers are trying to respond to user requests it is possible that developers may feel somewhat dictated to over the choice of technology.

Lessons learned

It is apparent that working with a technically able user can be very beneficial to a software project; there is less distance between users and developers in terms of language and approach than is generally the case. Such users have an understanding of both technical possibilities and limitations, and are willing to innovate and tolerate less than perfect developments as long as they provide some value or novelty.

However, it was also clear that there are potential pitfalls. As previously reported (Segal, 2005b), differing expectations of the requirements engineering process were an issue, the lead user's own understanding of the domain and his clear conception of the desired solution meant he was keen to jump to solutions, whilst the software team needed domain and problem understanding in order to keep up. Spending time reviewing the requirements elicitation process with the lead user, and especially reinforcing the value of ethnographic methods went some way to addressing this problem. The use of abstract scenarios (Carroll, 2000) focussing on the problems the users wanted to solve rather than the interface and functionality also proved a good way to encourage a technically focussed user to step back and think about the bigger picture. During requirements meetings, the analyst learnt to watch for requirements which were founded on assumptions of technical limitations, and took time to explore the lead user's motivations and goals.

The traditional project structure of software development, where users' requirements are filtered to the software development via a requirements analyst, limits dialogue between technically expert users and software developers. During the ADVISES project, occasions when the full project team have been able to meet have been rare and discussions have tended to focus on strategic, high level issues. We believe it would be desirable to reduce the distance between technically expert users and developers, for example, finding opportunities for technically expert users to work with developers, either in design discussions or via pair programming meetings.

Co-realisation

The case for strong user engagement in software development has been made many times before (Robertson & Robertson, 1999; Sutcliffe, 2002) and we will not rehearse the arguments. More pertinent to the issues we have raised in this paper is how this engagement may be best realised. Indeed, the experience of the ADVISES project leads us to consider a specific issue in relation to user engagement which is what roles are users best suited to. In the literature, one aspect of user engagement which tends to remain unquestioned is the division of labour in projects between users and technical team members (i.e., requirements analysts, software developers) (Voss et al., forthcoming).

In the case of the ADVISES project, we find examples of users who are capable of crossing that divide and making a contribution to projects of an altogether more technical nature. That is, we should challenge the existing and taken-for-granted divisions of labour in the development of ICTs and the separation that currently prevails between the contexts for their development and subsequent use. Our experiences in the ADVISES project suggest that such opportunities should be capitalised on where feasible, that we should promote dialogue and involvement between users who want to have technical input into the project. The traditional roles, and the expectations developers have of users are not helpful because they place the problem with the user and the technical decision making with the developer. We argue that, where appropriate, it would help to redefine the user as a user/developer, involve them in technical decision making meetings, and let them spend time working directly with developers (e.g., through pair programming, like collaborations). This would give developers the opportunity to find out about users' technical expertise and start to trust their abilities. It is otherwise all too easy for developers to dismiss users who program as 'hackers' and it is also useful for users to see the software engineering processes a developer follows.

The co-realisation approach, for example, seeks do away with rigid divisions of labour in software projects, to make users part of the development team, and validate the users' involvement (Hartswood et al., 2007). As Robinson et al. observe, "Good communication can only occur when all inputs are considered to be equally valid." (Robinson et al., 1998) At the same time, we also recognise that there are obstacles (for example, users may resist being diverted from their principle objectives, or, as in the case of our lead user, have no shortage of interest or enthusiasm, but simply lack time) to achieving this.

There are a number of ways in which this more fluid negotiation of the role of users might be facilitated, reflecting the fact that user engagement in software projects must be sensitive to prevailing circumstances and capable of adaptation should these change (as they invariably do). We suggest a number of approaches should be considered as facilitating mechanisms:

- Re-use of user developed code – software projects incorporate user developed code either as a component of a larger software development, e.g. inclusion of user developed statistical routines, or by treating small user developed applications as a spring-board for a larger development, e.g. adding features to support multiple users.

- Cycles of user/developer programming – for example, via the 'Seeding, Evolutionary grown and reseeding process model' (Fischer & Ostwald, 2002). This approach begins with the software developer creating a set of basic routines and functionality (seeds), which are made available to expert end users. These expert users work with the seeds, adapting them to their own ends (evolutionary growth). Following a period of evolutionary growth, the software developers review the evolved code and 'reseed' by merging and generalising from the user developed code.

- Indirect programming – Users do not contribute to the development of the core environment but are able to manipulate it via a secondary control layer, e.g. the use of visual languages, system configuration, workflow creation (Letondal & Mackay, 2004; Fischer & Giaccardi, 2004).

- Pair programming – users and developers sit together to write core software code. The time demands of such a method would make a wholesale adoption impractical for many projects, however it is clear that even a small percentage of overall development time was devoted to pair programming, there are likely to be benefits in terms of developer and end user relations.

- Users as developers – users act as part of the development team, independently writing core software code. Again, this approach places heavy demands on the end user's time. Furthermore, depending on the user experience, additional training in software engineering practices may be necessary.

Conclusions

The importance of early adopter involvement in innovative software projects is undeniable. However, this case study shows that making the most effective use of what an early adopter can bring to a project is not necessarily straightforward, and in some ways, can be as challenging as working with less visionary users. Reviewing the roles within a project is a potential way to reframe expert 'users' to recognise and develop their contributions to the development team. Any user brings expectations and assumptions to a project and the development team must strive to understand and manage them.

In summary, our findings echo those of Segal (Segal, 2005b), where a gap in user expectations of the process and developer expectations caused delays in the development. Segal suggests that the development would have been more successful if they had understood the scientists' methods of software development and tried to incorporate them, or acknowledge them, in the imposed external process. We recommend that when working with early adopters, technical collaborators should spend time explaining the development process, which may feel very slow to someone who just writes some code when he needs to do something. In addition, the technical team should explain what they hope to achieve from the requirements gathering activities.

We would also suggest that, when working with technically expert end users, the overall structure of the software development is reviewed, and that the project team consider alternative ways to involve such users and benefit from their expertise.

References

Beck, K. (1998). *Extreme Programming: A Humanistic Discipline of Software Development.* Paper presented at the Fundamental approaches to software engineering, Lisbon.

Beckles, B. (2005). *User requirements for UK e-Science grid environments.* Paper presented at the Proceedings of UK All Hands E-Science Meeting, Nottingham.

Bowers, J. & Pycock, J. (1994). Talking Through Design: Requirements and Resistance in Cooperative Prototyping. *CHI*, 299.

Carroll, J. (2000). *Making Use - scenario based design of human-computer interactions*: MIT Press.

Fischer, G. & Giaccardi, E. (2004). Meta-Design: A Framework for the Future of End-User Development. In H. Lieberman, F. Paterno & V. Wulf (Eds.), *End User Development - Empowering People to Flexibly Employ Advanced Information and Communication Technology*. Dordrecht: Kluwer Academic Publishers.

Fischer, G. & Ostwald, J. (2002). *Seeding, Evolutionary Growth and Reseeding: Enriching Participatory Design with Informed Participation.* Paper presented at the Participatory Design Conference 2002, Malmo University, Sweden.

Hartswood, M., Procter, R., Rouncefield, M., Slack, R, Voss, A., Buscher, M. and Rouchy, P. (2007). Co-realisation: Towards a Principled Synthesis of Ethnomethodology and Participatory Design, in Resources, Co-evolution and Artefacts, M. Ackerman, T. Erickson, C. Halverson and Kellog, W, Editors. Springer.

Letondal, C. & Mackay, W. E. (2004). Participatory programming and the scope of mutual responsibility: balancing scientific, design and software commitment, *Proceedings of the eighth conference on Participatory design: Artful integration: interweaving media, materials and practices - Volume 1*. Toronto, Ontario, Canada: ACM.

Robertson, S. & Robertson, J. (1999). *Mastering the Requirements Process*: Addison Wesley.

Robinson, H., Hall, P., Hovenden, F. & Rachel, J. (1998). Postmodern Software Development (Vol. 41, pp. 363-375).

Segal, J. (2005a). *Two Principles of End-User Software Engineering Research.* Paper presented at the First Workshop on End-User Software Engineering, Saint Louis, Missouri, May.

Segal, J. (2005b). When Software Engineers Met Research Scientists: A Case Study. *Empirical Software Engineering, 10*(4), 517-536.

Sutcliffe, A. (2002). *User-Centred Requirements Engineering - Theory and Practice* (1 ed.): Springer.

Taylor, M. J., Moynihan, E. P. & Wood-Harper, A. T. (1998). End-user computing and information systems methodologies (Vol. 8, pp. 85-96).

Voss, A., Procter, R., Jirotka, M., & Rodden, T. (2007). *Managing User-Designer Relations in e-Research Projects. Position Statement.* Paper presented at the Workshop on realising e-Research endeavours, e-Science Institute, Edinburgh, March.

Voss, A., Procter, R., Slack, R., Hartswood, M. and Rouncefield, M. (forthcoming). Design as and for Collaboration: Making Sense of and Supporting Practical Action. In Voss, A., Hartswood, M., Procter, R., Slack, R., Buscher, M. and Rouncefield, M. (eds) Configuring User-Designer Relations: Interdisciplinary Perspectives. Springer.

Zimmerman, A. & Nardi, B. (2006). *Whither or Whether HCI: Requirements Analysis for Multi-Sited, Multi-User Cyberinfrastructures.* Paper presented at the Workshop on Usability Research Challenges for Cyberinfrastructure and Tools, ACM CHI Conference, April.