Some Problems of Professional End User Developers

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

Abstract

By the term 'professional end user developers' we mean people such as research scientists who work in highly technical, knowledge-rich domains and who develop software in order to further their professional goals. In common with other end user developers, professional end user developers do not describe themselves as software engineers and have no formal training in software engineering. They differ from most other end user developers, however, in that learning programming languages rarely presents them with any problem. In this paper, drawing on data from field studies of different groups of professional end users, we examine the problems that such people face in meeting the demands of software development given the culture in which they work and their normal development practice. Understanding these problems is an essential prerequisite to developing tools, techniques etcetera to support professional end user development.

1. Introduction

By the term 'professional end user developers' we mean people such as research scientists or financial mathematicians who work in highly technical knowledge-rich domains and who develop software to further their own professional goals. Like other end user developers, they do not think of themselves as software developers but as physicists, mathematicians, biologists, etcetera, and have had no formal training in software engineering (though some may have attended courses on particular programming languages). Unlike many other end user developers, they tend to have few problems with learning a general purpose programming language such as Java. But there is a distinction between knowing how to implement programming constructs in a particular language, and using that knowledge to code a particular piece of software in order to address a particular domain problem. And in any case, software development involves more than just coding a piece of software which appears to solve the problem at hand. Depending on the context in which the software is going to be used, issues such as code comprehensibility, software robustness and performance become important.

In this paper, we take it as axiomatic that an essential prerequisite to producing tools, technologies and methods for supporting such developers, is to analyse how they work and the nature of the problems that they face in developing their software. This view is consistent with other studies such as [1], which examined the development practices of software engineers with a view to constructing tools to support these practices, and [2], which did the same for computational scientists.

We have conducted field studies with financial mathematicians [3] and with planetary and space scientists, [4], [5], and are currently conducting a study with biologists.

The clichéd view of end user development is that it involves an individual developing a piece of rather inconsequential software for his/her own use. However, our studies reveal a variety of contexts of professional end user development including:

- constructing models in order to advise clients;
- developing code both for the developer's own use and for the use of colleagues in the local community of practice;
- working with software engineers in order to develop a component library, and

0-7695-2987-9/07 $25.00 © 2007 IEEE
DOI 10.1109/VLHCC.2007.17

- working with software engineers in order to develop laboratory infrastructure software.

Another context of professional end user development, that of high end computational software, is examined in [2]. This work differs from ours not only in context but also in that it does not consider the culture within which professional end user developers work. There are some similarities and also some differences between the problems and practices discussed in [2] and those revealed by our studies, as we shall see.

In section 2 below, we briefly describe the context of our field studies, and in section 3, discuss both the culture within which professional end user developers construct their software and their normal development practices. In section 4, we discuss the problems which face professional end user developers and the software engineers which whom they may collaborate. Although we do not claim that these problems are unique to professional end user developers, we argue that they are exacerbated by the culture in which they work. We conclude the paper in section 5 with a summary and discussion.

2. The field studies

Here we will briefly describe our field studies, which we summarise in Table 1. One characteristic shared by all the studies is the amount of specialized domain knowledge required to develop the software. Another is that all the professional end user developers are highly educated in their domain (most have PhDs), and have no formal training in software engineering. A third is that, in building software in order to deepen understanding of partly-understood domains, as is the case with the first three field studies below concerning market economics and planetary and space science, it can be very difficult to ascertain whether or not the software is correct. In the case of unexpected output, it is not clear to the user whether this signifies that his/her domain model is incorrect or incomplete, or whether there is some error in the software, see also [2].

Data was collected primarily by means of semi-structured interviews which were audio-taped and transcribed, backed-up by emails and phone conversations and any relevant documentation. In the latest ongoing study, that of the biologists, 2.4, we have introduced a more ethnographic observational element. In the mature studies, 2.1 – 2.3, analysis was done by iterative inductive coding of the tape transcripts and other texts. Our interpretation of the data was negotiated and agreed with the participants, as advocated in, for example, [6].

2.1 The financial mathematicians

This study focused on a commercial consultancy where the professional end user developers were mathematicians on the lower rungs of their career ladder working through a series of professional examinations. Their task was to use an internally developed package together with Excel and VBA to develop financial models. These models were then used by more senior people in the organization, the consultants, in order to advise clients. The situation being modeled was full of uncertainty, for example, as to how the market would move. Following an embarrassing failure of one of these models to accommodate a cosmetic change required by a client (a simple change of format for the output), the CEO of the organization wished to proceduralise the development and testing of models by means of a manual.

The consequences of software failure were potentially high. Not only were the organisation's reputation and any potential repeat business at risk, but it was also theoretically possible for an aggrieved client to sue the consultancy to the point of bankruptcy.

Further details are available in [3].

2.2 The planetary scientists

In this case, the professional end user developers were either doctoral students or post doctoral researchers working within a research laboratory, and a typical development task was to write software to drive some instrument (for example, a spectroscope) and to analyse the output data.

The culture within the laboratory was broadly collaborative. Scientists with a particular facility for software development were recognised informally as software gurus. These people were called upon to help their colleagues either with advice or by developing the colleague's software for them. Given the similarity of the instruments, there was the potential to share code, though this potential was not fully realized, as we shall discuss in section 4 below. These phenomena, of the informal recognition of software gurus and of code being initially constructed for individual use but then becoming a shared artifact, are not uncommon in the context of end user software development, see, for example, [7].

Table 1. A summary of our field studies

Study	Purpose of software development	Customers	Social context of development	Consequences of software failure	Study specific features/issues
Financial Mathematicians	To produce financial models in order to advise clients.	Consultants in the same organisation.	A group of developers working individually on independent tasks; much collaborative help.	Could be disastrous to the organisation: financial loss; loss of reputation; at worst could lead to being sued to the point of bankruptcy.	A manual was implemented with the intention of institutionalising software development procedures and testing, and of sharing knowledge.
Planetary scientists	To drive scientific instruments and analyse the output data.	The developers themselves or close colleagues.	As financial mathematicians	If undetected, could corrupt the science to the detriment of the scientific community	Creating/sharing knowledge of software development.
Space scientists	To provide a library of components to enable space scientists to drive instruments	Space scientists	Professional software engineers working at a remote location from the space scientists.	Large financial loss and loss of reputation for the research organisation; potential corruption of science to the detriment of the research community	The switch of roles from being a professional end user developer to being a customer in the context of a waterfall-like development.
Biologists	To provide laboratory infrastructure software.	The wider community of biologists.	An interdisciplinary and distributed team of biologists and professional software developers.	At worst, loss/corruption of scientific data.	Begun in June 2006 and ongoing.

If the software failed in such a way that the failure was not apparent but resulted in data being output which was not true to the science, then this could affect the whole scientific community.

Further details are available in [5].

2.3 The space scientists

This situation is somewhat different from those described above. The space scientists, like the planetary scientists described in 2.2, had experience of developing their own software to drive experimental instruments, but instruments for use in space are more complex and have to comply with more constraints than those in the laboratory. In this situation, the space scientists enlisted the help of professional software engineers who supplied them with a library of components which the scientists could compose so as to drive the instrument when it reached its destination in space. The whole enterprise is very risky. For example, the physical context in which the instrument is to be deployed (radiation levels, temperature etcetera) is not known with any certainty, and so the odds on the instrument surviving in this alien environment are likewise not known.

The development process by which the component library was constructed adhered to a waterfall-type staged development as advocated by the European Space Agency for small (in their terms) software projects. The role of the space scientists was that of customers expected to supply requirements upfront and to carry out extensive user testing. They found both these expectations difficult to meet. Supplying requirements upfront ran counter to their previous experience of developing their own software in the laboratory, see 3.2 below, and *extensive* user testing did not happen, for reasons discussed in 4.3.

Although the scientists in this study were no longer acting as professional end user developers, we feel that the study has its place in this paper, because the provision of a library of components is often mooted as a means by which software engineers might support professional end user developers, see, for example [8] and [9]. This study made it clear that the provision of such a library is not without problems, as we shall discuss briefly in section 4.

Further details are available in [4].

2.4 The biologists.

We shall not have much to say about this as our field study has just begun, but its inclusion in this paper is useful in illustrating the variety of contexts in which professional end user developers work.

The challenge here is for an inter-disciplinary team of biologists and software engineers to develop some laboratory infrastructure software (which we will abbreviate to LIS) to meet the needs of a variety of laboratories within the UK (and possibly beyond). Some of the biologists involved have considerable experience in developing, or modifying, software for their own laboratories. The challenges facing the development of the LIS are somewhat different from those facing the development of software for one's own laboratory, however. The LIS must both be broad enough to reflect the wide variety of working practices and yet flexible enough to appear to users at a particular laboratory as being specific for that laboratory. It also has to be of production quality in terms of robustness (there will be nobody in the individual laboratories to fix it should it break) and in terms of performance.

Having briefly described each field study, we will now discuss the culture within which the participants develop software and the means by which the software is developed.

3. The culture and development practices of professional end user developers

We begin by considering the culture within which professional end users develop software.

3.1 The culture of professional end user development

Despite the potential consequences of software failure as discussed above and in Table 1, the dominant characteristic of the culture within which professional end user developers work is that software development is regarded as being a very secondary activity to their main work.

This is illustrated by the following quotes taken verbatim from interview transcripts:

'[Developing software] is a secondary thing to being a scientist… it's just a tool. … just part of your skills base. The trouble is…it is seen as *so* secondary …' [planetary scientist, emphasizing the word 'so'].

'[the ability to develop software] is just a tool that you… pull out of the metaphorical cupboard when you need it' [space scientist]

The implications of this perception of software development as a secondary activity are far-reaching. One such is that the effort and knowledge required to develop software both tend to be under-estimated. The following quotes are from a planetary scientist and the line manager of the developers at the financial institution.

'I think the attitude towards computing .. [is] it's something you do in your spare time. I don't think people have any idea how long it actually takes to sit down and write a program. I think we quite happily imagine that you just … spin it off in half an hour over your lunch time.' [planetary scientist]

'everybody in theory knows how to do [software development]…. It's assumed that everybody knows what to do' [line manager in the financial institution]

'There is a knowledge management capturing exercise [which] goes on but that's more for intellectual capital rather than technique… what we don't try and capture, as far as I'm aware… is what we learnt about [software development] that we could pass on to everybody' [line manager]

The use of language in this last quote, where domain knowledge is 'intellectual capital' and software

development knowledge is 'technique', reveals, we think, a distinct value differential.

In addition to the quotes above, a planetary scientist told us that developing software to drive an instrument was not considered to be 'real work', whereas building the hardware of the instrument definitely was.

The perception of software development within professional end user organizations as a secondary activity and the lack of recognition of the skills and knowledge required to develop software, leads to a situation in which possession of such skills and knowledge is not formally recognized by the organisation's rewards or appointments structure. For example, in the space scientists' team, a man was appointed to a post entitled 'project programmer' with the brief of developing ground software. This man had no software development experience and no software engineering knowledge beyond a coding course at University. He made it clear that he saw the job as a means of gaining a toe-hold into space science rather than as a means of developing a career as a software developer.

Having considered the culture within which professional end user developers produce software, we now look at their development practices.

3.2 Development practices

The salient characteristic of professional end user development practice is its iterative quality. These iterations are not simply a matter of using trial-and-error to hack the code, but rather seem to reflect a growing understanding of the requirements as the code evolves. The following quote, typical of many, is from [4]:

'Generally what tends to happen with me is, I do a first attempt …start writing programs, start using it, realize it's not quite what I wanted, and then have a second attempt' [planetary scientist]

A space scientist contrasts the scientists' need for emergent requirements with what he perceives as the software engineer's need for an upfront requirements specification (again, this quote is from [4]):

'.. on the science side, we're not good enough at defining the specifications. But I suppose we have more of a view… that you write it and then test it and then improve it a little bit having looked at the output and so on. Whereas the software developer would rather write it and then view the thing as more or less finished. But it's really only the *start* of the development process once the software is running in conjunction with the experiment… '[space scientist, our emphasis].

This reliance on iterative development to enhance the scientist's understanding of the problem and to reveal new or modified requirements has been seen elsewhere. For example, the computational scientists described in [2] exhibited the same iterative and evolutionary behaviour.

Another characteristic of professional end user development practice in our studies is the lack of any disciplined testing procedure. This may be explained, in part at least, by the fact that when a scientist is developing software for his/her own use, and it becomes apparent to him/her that the software is incorrect at some level, then he/she can just amend it as needed.

4. Some problems facing professional end user developers

The first problem we consider arises from the culture within which professional end users work, as described in section 3.1 above. It concerns how professional end users acquire software development knowledge within a culture where it is assumed both to be trivial and part of everybody's armoury of tools. As we discussed in the introduction, the developers in our studies had little difficulty in learning how to implement programming constructs in a programming language (Java, LabView or Visual Basic in these studies): they used manuals or internet tutorials or memories of University coding courses. However there is a big gap between learning these basics and using the programming language to implement a software solution to a specific domain problem. Where the professional end user developer finds the knowledge to bridge that gap is the issue which we will address in 4.1 and 4.2, focusing on artifacts such as code and documentation in 4.1 and on the community of practice in 4.2. In 4.3, we shall discuss a problem associated with the normal development practices of professional end user developers (section 3.2). This is the problem of testing.

4.1 The problems of sharing knowledge through code and documents

One implication of professional end user developers' perception of software as discussed above, is that they are loath to expend effort on aspects of software development which do not immediately appear to improve the supporting of domain tasks. This makes difficult the sharing of software

development knowledge through artifacts such as code and documents, as we shall now discuss.

We consider firstly the sharing of knowledge through code (code reuse). The literature tells us that the formal institution of a reuse program in software engineering organizations is very difficult. A study of such organizations in [10] found that the successful introduction of a reuse program is dependent on major changes in organizational culture so as to embrace reuse, together with significant changes in work practices and roles. If a formal reuse program is difficult to implement for professional software engineers, then it must presumably be far more difficult to implement within the context of professional end user computing, where software development is seen as very much a secondary activity. We saw only one attempt at formal code reuse in our studies. The CEO of the financial mathematicians (see 2.1 above) intended the on-line manual not only to lay down development and testing procedures but also to act as a repository for code fragments which could be reused, possibly with some customization. As described in [3], this intention was not met. The costs in time of identifying a code fragment suitable for reuse, testing its robustness and modifying it so that it could be easily customized for new situations, were perceived as being too great.

As to informal code sharing, given the similarity in their instruments, it was theoretically possible for planetary scientists to share each other's code, perhaps with some degree of modification. Such informal sharing is facilitated by the code being easy to modify, and hence comprehensible. However, code comprehensibility does not appear to be a significant issue for professional end user developers. The expectation was that only the developer could understand his/her own code, and even for the developer, this understanding became very difficult over time, as attested by the following quote:

[space scientist] 'we would hope to use [the software] later on … [on] different projects. So we keep everything.. But.. if there is another problem [which] comes along… we try and start from scratch rather than trying to pull out [the saved software]. Because …. I would have forgotten what I'd done on that program'.

[investigator] 'You couldn't even remember for yourself?' [as the originator of the software]

[space scientist] 'that's right'

[investigator] 'So another person wouldn't?'

[space scientist] 'No no chance. Not a chance'

This lack of awareness of the potential importance of code comprehensibility is mirrored by the community of developers in the financial consultancy. The CEO described how developers would strive to produce the tersest possible code to fit a given situation. He would then ask them to rewrite this code so that comprehensibility was favoured over terseness.

Another artifact by which knowledge might be shared is documentation. Professional end user developers did not voluntarily produce documentation, apart from the occasional user guide, in either our studies or those reported in [2]. The space scientists, encouraged to provide and archive project documents by the European Space Agency, were skeptical about the use of such documents as knowledge artifacts:

> '[there are a lot of documents available but] going through them and exactly understanding them would be far more difficult than getting the guy who did it and asking him to go through it' [space scientist (not the same person as quoted above)]

This quote illustrates the general perception that the community of practice (those people loosely working together in close proximity and undertaking similar tasks) is more effective in sharing knowledge than documents. The following quote illustrates the importance of this community in facilitating code sharing:.

'I would expect anyone who wanted to use them [his routines] to get in touch with me .. rather than fumble around in the dark without help… [otherwise] it might take a fellow post-doc several weeks to sort it all out'. [space scientist who kept a repository of his own code]

We now turn our attention to a consideration of the problems of relying on a community of practice for the sharing of knowledge.

4.2 The problems of sharing and creating knowledge through the community of practice

The importance of a community of practice in creating and sharing knowledge is widely recognized in the literature, see, for example [11]. Certainly, the professional end users themselves appreciated this. Both the financial mathematicians and the planetary scientists described how, if they came across a problem, their first port of call would always be a colleague whom they knew had done similar coding before. The mathematicians described moving desks so as to facilitate such sharing. The two quotes above illustrate the space scientists' recognition of the importance of the community of practice.

There are problems with relying on a community of practice for knowledge creation and sharing, however. The problem where a community of practice does not exist in that there is only one end user developer at a location, though not unusual in end user development, did not hold in our mature field studies (of the financial mathematicians and the planetary and space sciences). Consistent with the fact that the reputation of professional end users in their organizations does not depend on the exclusivity of their knowledge of, and skill in, software development, we saw no problems associated with a reluctance to either ask for or share knowledge, cf. [12]. What was problematic was the inherent instability of the community. Professional end user developers in our mature field studies did not intend developing software for the rest of their careers. In the financial institution, when the students passed all their professional exams, they became consultants and instructed their students, in turn, to develop software for them. In the case of the planetary and space scientists, the people doing the software development were almost invariably either doing their PhDs or on short-term research contracts. Their goal was to become established enough in their domain so as to obtain a permanent post and, again, instruct their students or research fellows to develop their software. In both cases, as the mathematicians and scientists ascended their career ladder, they took their software development knowledge with them and, in any case, it fell out of date through lack of use.

Of course, this unstable community of practice is not unique to professional end user developers. Software engineers also frequently work on short term contracts. However, software engineers can draw on a wider network of practice to create and share software development knowledge. They can read magazines, attend practitioner meetings, read and publish blogs, subscribe to internet newsgroups etcetera; they are not dependent on a collocated community of practice. For professional end user developers, however, consistent with a culture in which the value of software is not fully recognized, this wider network of development practice does not exist: the magazines, newsgroups etcetera to which they subscribe, are focused on their domain. The collocated community of development practice is thus more important, we argue, to professional end user developers than to professional software engineers.

4.3 The problems of testing

We saw in section 2 that the consequences of software failure in our studies could be dire (and, in fact, this situation is not uncommon in end user development, see, for example, [13]). Given this, we were surprised to see that testing did not seem to be taken very seriously. This was partly due to pressures of time. In the financial consultancy, following the software failure described in 2.1, the main aim of the CEO in developing the manual was to institutionalize a disciplined testing regime. This wasn't wholly successful. In the context of a busy consultancy, testing often fell victim to time pressures. In these cases, the decision was taken, consciously or unconsciously, that it was riskier to alienate the client by not delivering on time than by delivering a model which had not been fully tested. In any case, given the uncertainty of the domain, as described in 2.1, it was impossible to verify the model completely.

The problem of time pressure is clearly not unique to professional end user developers, but we shall now argue that it is exacerbated by their method of requirements gathering. We saw in 3.2 that a cornerstone of normal professional end user development practice is the emergence of requirements and deepening of understanding of the problem domain through iterative and evolutionary development. One problem with this is knowing when to stop.

'.. being like a bunch of scientists, we [thought] we could change everything up until the last minute…[the software engineers] were just saying "Sort the requirements out now! Do it now! You haven't got time!"'[Space scientist, quoted in [4]]

This is consistent with the observations of the computational scientists in [2] who refer to scientific simulation programs as being under constant evolutionary development, rather than having separate phases of development and maintenance. This 'not knowing when to stop' is especially problematical in developments where there is a definite delivery date (for example, of the instrument to a satellite which is going to be launched on a particular date come what may). Later parts of the development process – such as testing – get squeezed.

There are several further points to be made about testing. Firstly, not taking testing seriously is consistent with a development culture in which the value of software is not fully recognized. Secondly, we note that our findings differ from those of [2] who observed that regression testing was regularly carried out by the computational scientists as they evolved their software. Thirdly, as we mentioned in the introduction to section 2 above, system testing is inherently difficult in poorly understood domains. Finally, the problem of testing being squeezed out by time pressures might (theoretically, at least) be solved in part by test-driven development. The project leader of the biologists is currently trying to promote test

driven development within his team. It remains to be seen whether this will be successful.

5. Summary and discussion

In this paper, we have demonstrated that the context of professional end user development is not the clichéd one of the inexperienced user of formal languages developing some inconsequential software for his/her own use over an hour or two of spare time. Rather, the professional end user developer has no fear of formal languages and develops complex software which has an important role to play in furthering his/her professional goals and which can incur grave consequences in the event of failure. Unfortunately, the skills, knowledge and effort required to develop this software, its value and the risks associated with it, do not seem to be recognized by the organizations within which professional end user developers work. We discussed two fundamental characteristics of professional end user development: the reliance on iterative, evolutionary development in order to evolve an understanding of the requirements and the problem domain, and the difficulty of carrying out system testing when the correct outputs are not known. We noted problems with sharing software development knowledge and with testing. Although neither of these problems are unique to professional end user developers as opposed to career software developers, both might be exacerbated by the lack of value afforded software development in the professional end user development culture.

As discussed in the introduction to this paper, the purpose of these field studies is to determine the tools, techniques and methods that will best support professional end user development. Based on our findings, we advise that professional end user developers need support in sharing software development knowledge and in testing. Any supporting tools must acknowledge both the reliance on iterative development and the perception of software development as being very much a secondary activity. The implication of this latter is that any tools, techniques or methods designed specifically to support professional end user developers that require major changes in roles or in work practices, are doomed to failure. Rather, to have any chance of success, they must fit as seamlessly as possible into the current pattern of software development.

6. References

[1] Singer, J., Lethbridge, T., Vinson, N. and Anquetil, N. (1997) An examination of software engineering work practices, in *Centre for Advanced Studies Conference (CASCON)*, Toronto, Ontario, 1 - 15.

[2] Carver, J.C., Hochstein, L.M., Kendall R.P., Nakamura, T., Zelkowitz, M.V., Basili, V. R., Post, D. E. 2006 'Observations about software development for high end computing' CT Watch Quarterly, November, 2006, pp 33-38.

[3] Segal J., 2001, 'Organisational Learning and Software Process Improvement: A Case Study', in *Advances in Learning Software Organizations*, K-D Althoff, R.L. Feldmann, W. Muller (Eds.), Lecture Notes in Computer Science, Vol. 2176, Springer, 68-82.

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

[5] Segal, J. (2004) .Professional end users and software development knowledge. Technical Report 2004/25, Department of Computing, The Open University, UK, http://computing-reports.open.ac.uk.

[6] Seaman, C. 1999. Methods in empirical studies of software engineering. IEEE Transactions on Software Engineering 25(4): 557-572.

[7] Nardi, B.A. *A Small Matter of Programming: Perspectives on End User Computing*. MIT Press, 1993.

[8] McBride, N., Wood-Harper, A.T., 2002. Towards user-oriented control of end user computing in large organizations. Journal of End user Computing, 14(1), 33-44.

[9] Morch, A.I., Stevens, G., Won, M., Klann, M., Dittrich, Y., Wulf, V. 2004. Component-based technologies for end user development. CommACM, 47(9), 59-62

[10] Morisio, M., Ezran, M., Tully, C. 2002. 'Success and failure factors in software reuse. IEEE Transactions on Software Engineering, 28(4), 340-357

[11] Brown, J.S., Duguid, P., 2000. The Social Life of Information. Harvard Business School Press.

[12] Pipek, V., Hinrichs, J., Wulf, V., 2002. Sharing expertise: challenges for technical support. In Ackerman, M., Pipek, V., Wulf, V. (eds). Beyond Knowledge Management: Sharing Expertise. MIT Press, Cambridge MA

[13] Panko, R., 1998. What we know about spreadsheet errors. Journal of End User Computing. 10(2), 15-21.