ISSN 1744-1986

Technical Report N^o 2004/25

Professional end user developers and software development knowledge

Judith Segal

12th October 2004

Department of Computing
Faculty of Mathematics and Computing
The Open University
Walton Hall,
Milton Keynes
MK7 6AA
United Kingdom

http://computing.open.ac.uk

Professional end user developers and software development knowledge

Judith Segal

Department of Computing

The Open University

Milton Keynes MK7 6AA

UK

Abstract. This paper seeks to explore how IT professionals might best support professional end user development. By 'professional end users', we mean practitioners of some recognized technical, scientific or mathematical profession, who develop software in order to further their professional goals. Our field studies demonstrate that such software development takes place within a culture in which it is perceived as being very much a secondary activity, and this perception may be reflected in the inadequate allocation of resources both for the development of software and for the acquisition, creation and sharing of knowledge of both software development and the software product. We suggest that importing some practices from eXtreme Programming might ameliorate (though not entirely solve) these problems of knowledge creation and sharing.

1. Introduction

In this paper, we are interested in professional end user developers. By the term 'professional end user developers', we mean people who are not professional software developers, but who are members of some identifiable knowledge-rich technical profession developing their own software as a tool for furthering their professional goals. Examples of professional end user developers include scientists and financial consultants. Our over-arching research question is: how might IT professionals best support professional end user developers?

The literature on end user development appears to form two strands. The first strand concentrates on the individual end user; the second, on the organization. The former tends to focus on people who are assumed to be unused to, and uninterested in, programming in general purpose programming languages such as Java. There is thus an emphasis on development environments which have been specifically designed with these people in mind, see, for example, Panko, 1998, on the use of spreadsheets, perhaps the most common of such environments, and Sutcliffe & Mehandjiev, 2004. There is some cognizance in this literature that end users need support in aspects of software engineering other than coding, such as testing, see, for example, Burnett et al., 2004. The second strand is concerned with organisational issues such as the management and support of end use computing, see, for example, Powell & Moore, 2002, and Downey, 2004.

We believe that the first strand of literature has little to offer professional end users. These people, with their experience of formal languages such as mathematics, do not tend to be uncomfortable with coding. Indeed, there is a long history of such people developing their own code. For example, scientists have been producing their own programs in Fortran since Fortran first arrived on the scene. This facility with coding, together with the variety and richness of professional end users' interests, leads us to believe that the construction of bespoke development environments is not as important a concern for professional as for other classes of end users. Despite this facility with coding, however, we should emphasize that professional end users are very different from professional software developers, as shall describe in Section 2 below. The second literature strand, focusing on the relationship between the organization and end user computing, is, however, as relevant for professional as for any other end users.

In order to address the question as to how IT professionals might best support professional end user developers, we believe it vital firstly to investigate through field studies the problems that such developers actually have. In section 2 below, we briefly describe two such field studies, of financial consultants and of research scientists, from which arose the empirical data informing the discussion in this paper.

Section 2 presents evidence to illustrate what we perceive to be the essential difference between the cultures of professional end user and professional software development. This difference is that in the case of professional end user developers, the knowledge, skills and effort required to develop software are not adequately recognized, and this lack of recognition may be reflected in inadequate resource allocation. This leads to the identification of a major set of problems facing the professional end user developer: the acquisition, creation and sharing of software development knowledge. In section 3, drawing on both recent literature and on our empirical data, we discuss various ways in which software development knowledge might be created and shared between professional end user developers. In section 4, we explore whether certain practices which have been tried and tested by professional software developers, might profitably be adopted in the context of professional end user development. We are especially interested here in practices associated with a class of development methodologies, agile methodologies, which are currently the subject of great interest. These methodologies place great emphasis on supporting 'communication and community' (Cockburn, 2002, p.199). In section 5, we summaries our discussions and present our conclusions.

We begin with a brief description of our field studies.

2. A brief description of our field studies

Here, we briefly describe how we collected the data which form the evidence on which we base our discussion, and go on to discuss the salient characteristics of professional end user development.

2.1. The context of collecting the data

Our data were collected in field studies, which focused on two organisations: one, a financial consultancy, described in detail in Segal (2001), and the other, a scientific research laboratory, discussed in Segal (2003).

The financial consultancy has three levels of professional staff. The first level consists of people, referred to as students in the text below, who are working for the consultancy while progressing through the series of exams which allow them to be admitted to the relevant professional body. These people are invariably high calibre graduates of numerate disciplines; many have PhDs. If and when they pass all their exams, they become consultants, advising financial services companies. The most successful of the consultants become partners.

The research laboratory is attached to a University and pursues fundamental research supported mainly by government funding agencies. Most people working in the laboratory are either working for their PhDs, or working for specific projects as post-doctoral fellows on short-term research contracts. Some people have been employed on a succession of such contracts. A few people are on permanent contract: these people are eminent, or pre-eminent, in their scientific field.

In the financial consultancy, we gathered data by means of separate interviews with a total of 12 people representing all three levels (students; consultants; partners), following up these interviews, where necessary, by phone-calls. At the research laboratory, we interviewed a total of 10 people, including some on short-term contracts; some who had worked on a succession of contracts, and one with a permanent position. As with the financial consultancy, we conducted further interviews or used telephone or email contact where a follow-up was necessary. All the quotes used as illustrations in this paper are taken verbatim from transcripts of the interviews in the two organisations.

At first sight, it might appear that these two organisations are quite different. For the financial consultancy, the bottom line is making a profit; for the research laboratory, it is advancing scientific knowledge. Nevertheless, as we shall now describe, there are commonalities in the context of software development.

2.2. The context of professional end user software development

We firstly consider the role of the software produced by professional end user developers within their organization. This software, in contrast with that produced by professional software developers, is generally not the primary product of their organization, but rather a means of achieving that product. In the case of the financial consultants, the software development results in a financial model on which the organization bases its advice to clients. In the case of the research scientists, the software both controls instruments which implement various experiments and handles the analysis of data from the instruments. In both organisations, there is a high price to pay for getting the software wrong (as there is for much end user development). In the case of the financial consultancy, an incorrect software model could lead to the consultancy being sued and losing its reputation - or, in the worst-case scenario, being forced into bankruptcy. In the case of the research laboratory, software failure could lead to the loss of experimental results – or, worse, because more insidious, a corruption of those results to the detriment of the advance of scientific knowledge.

The importance of correct software to the organization appears not to be reflected by appointment and promotion policies, and by perceptions of the value of software and the amount of effort and specialized knowledge required for its development, as we now discuss.

Regarding appointment and promotion policies: one of the defining attributes of professional end users is that, even though they may spend a large proportion of their time on software development, they do not see themselves primarily as software developers, but rather as domain specialists (research scientists or financial consultants, in our studies). This view of themselves is shared by their organisations, who appoint and promote such people almost invariably on the grounds of their domain expertise. For example, in the research laboratory, even a man appointed to a post with the title of 'Project Programmer' regarded himself primarily as a scientist rather than as a software developer: indeed, he had had no prior experience of such development.

The lack of recognition of the importance of software is exemplified by a research scientist telling us that he felt that the development of software to control instruments was not considered to be "real work", as opposed to modifying the hardware of the instrument, which definitely was. The following quote from a financial consultant describing post project reviews also, we believe, testifies to the (lack of) value afforded software development knowledge: as opposed to domain knowledge ('intellectual capital'), new knowledge of software development ('technique') is not worth capturing:

> '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' [financial consultant]

As to the amount of effort required to develop software, a research scientist commented:

> '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.' [research scientist]

The knowledge and ability necessary to develop software appears to be regarded as part of the generic skill set of the professionals, as the following quotes illustrate:

> '[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 …' [research scientist, emphasising the word 'so'].

> '[software developed by oneself] is just a tool that you… pull out of the metaphorical cupboard when you need it' [research scientist]

> 'everybody in theory knows how to do [software development]…. It's assumed that everybody knows what to do' [financial consultant]

How 'everybody' acquires the requisite knowledge and skills in an organization in which software development is seen as being 'not real work' (or at any rate, not the primary work of the organization), is the focus of the rest of the paper. In discussing this, we shall refer both to data from our studies and to recent relevant research literature.

3. Acquiring, creating and sharing knowledge of software development

In general, in our view, there are various means by which knowledge can be acquired and shared:

(i) by formal training

(ii) by information artifacts, such as documentation and code. Here we are concerned both with code as documentation and with the reuse of code.

(iii) by the community, or network, of practice.

(iv) by specific knowledge archives and/or knowledge management tools.

We shall now consider each of these separately in the light of our field study data.

3.1. Formal training

The literature on end user computing emphasizes the importance of training, see, for example, Powell & Moore, 2002.

In the research laboratory, formal training in software development played very little part. Only one of the research scientists we interviewed had done some software engineering as an undergraduate; the exposure of the others to university level courses in computing was limited to simple programming. One of our respondents had attended a formal training course in LabView during the time of the project on which she was employed, but that was only after considerable lobbying of management on her part. In general, the scientists were expected to learn and deploy the appropriate programming languages (Visual Basic; Fortran; LabView) on their own. On the face of it, this expectation is a reasonable one. Nearly all the scientists had PhDs and were used to finding and deploying their own knowledge resources. In addition, they were all familiar with using formal languages and notations in physics, chemistry and mathematics, and so it is reasonable to assume that they would have no problems with learning formal programming languages. And indeed our field study bears out this assumption: coding per se appeared to pose little difficulty. But programming is more than merely coding, and without suitable education and training, there is a danger that developers might simply concentrate on constructing a piece of software which appears to be fit for the immediate purpose, neglecting higher level issues such as how to ascertain that the software is, indeed, fit for purpose and how to construct software which is both robust and easy to modify and maintain.

In the world of financial consulting, of the 14 exams which students are required to pass in order to be admitted to the professional body, none includes any software development. Within the specific financial consultancy, students were given courses on how to use the in-house software development package. As with the research scientists, we feel there is a danger here that the focus is on how to produce a piece of software which appears fit for purpose and that higher level issues might be ignored.

3.2. Information artifacts

Here we consider three such artifacts: documentation, self-documenting code, and reusable code.

3.2.1. Documentation

In a software engineering project, one document may have several roles. It may act as a contract, for example, between customers and developers, as in a requirements specification document; it may act as an instrument of coordination and control, and it may act as an information and communication artifact, for example, by describing the system to future maintainers or by capturing processes and best practice for use in other projects. We shall focus here on its role as an information artifact. We question, however, whether project documents, with their multifarious roles, can function effectively as information artifacts. As Henninger, 2001, says:

> '.. some creative thinking is needed on how the artifacts created in project development can be turned into knowledge assets' [Henninger, p.10]

In our field studies, project documentation was rarely, if ever, produced spontaneously (with the exception of user manuals). However, one project at the research laboratory, described in detail in Segal, 2003, involved collaboration with other partners (including professional software developers, and groups from outside the UK) and a requirement for project documentation from the overall project managers. In this project, there was a need to convey knowledge over a period of some years. Some of the scientists expressed skepticism that the project documentation would achieve this.

> 'All the documentation is there…. But if people don't know the experiment it is very hard to look up the documentation and do anything with it…

> … it is going to be very, very difficult for people coming in 6 or 7 years' time, pick up the documents, read through them, which will probably take.. about 4 years, and try and understand the experiment' [Research scientist]

In addition, problems with finding relevant information in the plethora of documentation were noted by two of the research scientists.

Others were more positive. The senior research scientist claimed that in order to communicate knowledge over time,

> 'it is critical that the original designers document very well' [senior research scientist]

and the project programmer claimed that the use of documents and spreadsheets to track documents meant that

> 'we have a well used and well defined framework already existing for management of knowledge within the project' [project programmer at the research laboratory].

It should be noted, however, that neither of these people had much experience of such documentation, the former by virtue of his seniority and distance from software development, and the latter, by virtue of his inexperience.

The situation at the financial consultancy was somewhat different. Management initiated the introduction of a manual, which aimed to share software development knowledge and standardize development procedures throughout the organization. As we describe in Segal, 2001, although the manual proved useful as a repository of arcane knowledge, it proved difficult to maintain, in the face of pressure to get on with the actual development, and the community of practice – colleagues – appeared to be valued more than the manual as a source of knowledge. We shall say more about communities of practice later in this section.

3.2.2. Code as documentation

> 'Software maintainers have long known that the only documentation you can really trust is the code.. ' [Paulk, 2002, p.17]

But if code is to be an information artifact, then it has to be written with a view to being comprehensible. And in the absence of any training or awareness of higher level issues, as discussed above, our evidence is that, in general, it was not.

> 'he's the only one who understands his programs' [one laboratory research scientist describing another]

Another research scientist commented that when a subroutine was no longer used, it was still left in the code, thus needlessly increasing its complexity.

A similar situation was seen among the software developers in the financial consultancy. Here a senior manager reported asking a developer to expand some terse software in the interests of comprehensibility.

3.2.3. Code reuse

Reuse, for example, of library components or of objects, is an intrinsic part of most current software development. Provided one knows the exact effects of a component (and we saw in our study of the financial consultancy, Segal, 2001, the potentially catastrophic results of not so knowing), this follows excellent engineering practice (one doesn't have to manufacture a chip oneself before constructing a computer) and also might partially address the knowledge sharing problem, in that knowledge of how to solve a particular problem is embodied in the code. As with code-as-documentation, however, so with code-for-reuse, the code must be designed for reuse, and this requires effort beyond that necessary simply to produce code sufficient for the job in hand. One scientist commented on the difficulty of reusing code which has not been expressly designed for the purpose:

> [research 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 most people would have…. 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]
> [research scientist] 'that's right'

[investigator] 'So another person wouldn't?'
[research scientist] 'No no chance. Not a chance'

We saw plenty of evidence of the research scientists reusing publicly available code and components, including NAG libraries, and routines available on various web sites. But the only local reuse episodes we saw were two scientists who had a long history of working together, reusing and extending each other's codes, and an individual scientist who made some effort to reuse his own code, keeping it, appropriately commented, in a structured directory with accompanying example input and output files.

As to the financial consultancy, it was hoped that developers might include code portions for others to use as part of an on-line manual. This was dependent on the developers having the time to extract their code and an experienced consultant then putting aside more time to generalize the code so that it would be useful in contexts beyond that for which it was originally developed. Needless to say, such time was rarely available.

These findings are consistent with those of Morisio et al., 2002, who found that the success of a software reuse programme depends on organisational changes, such as (crucially) creating a culture of reuse, one effect of which is that the effort required to create reusable resources is recognized, and this recognition is reflected in a suitable allocation of resources.

3.3. The community, or network, of practice

Brown and Duguid, 2000, distinguish between communities and networks of practice. In the former, people sharing the same work culture and working on similar jobs are collocated; in the latter, they are not actually in the same place but are connected through conferences, newsletters, mailing lists, web-sites and the like.   Brown and Duguid, among others, argue convincingly that a community of practice is a powerful instrument for creating an organisational shared memory, with knowledge distributed among the members of the community and shared perhaps by means of stories, and also for creating new knowledge, by negotiation and collaboration between the members.

We saw in 3.1. that the participants in our field studies had some reservations concerning the efficacy of documents as information artifacts. This is not surprising: the success of a document in sharing knowledge depends on the writer making many successful predictions about potential readers. These predictions include: the content of the information that any reader is going to want; the search strategies that will be used by potential readers, so that the writer can make information easy to find; the prior knowledge of potential readers, and, finally, the use to which they will put the information, so that the writer can present the information in a way which optimizes its utility. All these problems may be addressed, in part at least, by a small community of practice. A member of such a community is likely to know who has any information relevant to his/her query; the informant is likely to know the level of expertise of the questioner and how he/she wants to use the information and hence be able to present the information in such a way as to make it most useful.

It is not surprising, then, that our studies reveal that our respondents had more faith in the community of practice than in documentation.

'[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' [research scientist]

'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'. [a scientist who kept an annotated repository of his own code]

'[The manual is good] as a guideline but …[I] always had to ask someone else as well.'' [student at financial consultancy]

'[if I need help]… if I'm working with someone who's done similar things, first point of contact might be: have you had similar problems doing this? ' [another student at the financial consultancy]

In the research laboratory, despite the appreciation of the strength of the community of practice among its members, we, the investigators, wondered whether this informal community might be better recognized and supported by the organization.   For example, there is the issue of the choice of programming language. Our data indicate that the choice was often made for non-technical reasons, for example, Visual Basic was chosen by the software supremo on one project because it was similar to Basic, which he had learnt as an undergraduate, and by the project programmer, on the recommendation of the software supremo. On other projects, management had issued the directive that LabView must be used. The use of different languages presumably inhibited knowledge sharing and creation.

There also appeared to be a break in the community between the group of scientists who were working on projects established at the laboratory, and another group, who had, at the time of the field study, joined the organization relatively recently. The laboratory's effort to create a community by instituting a series of seminars, was not perceived as being successful.

> 'Funnily enough, a year on [after the new group of scientists joined the institute], what happened is we're more polarized than we were when they first got here…. Those Monday morning meetings are bizarre… Almost like a them and us. Like they speak. And then we speak. … the group's so broad… no matter what you're talking about, a third of the group are just .. bored out of their minds' [scientist]

Perhaps, rather than imposing any formal meetings, the organization might support more informal interaction, by, say, providing informal coffee/common rooms.

Given the undoubted strength of the community of practice in sharing and creating knowledge, the question arises as to whether any other information artifacts are necessary. We believe the answer to this is 'yes', for several reasons.

The first is that there are some types of information, perhaps rather arcane and arbitrary, which are better communicated by a concrete artifact. This was the case in the financial consultancy, where developers generally sought information from other developers, but consulted the manual for details such as the precise form of the input required by a particular software module.

The second reason is that an organization might not have as cooperative a culture as we observed in our two field studies. For political or personal reasons, a person may not want to ask for information (she/he may not wish to look a fool, or may not want to disturb a colleague), or to make his/her knowledge available to others, see, for example, Pipek et. al, 2002.

The third reason is that the required information may not be available in the community of practice. This is especially pertinent to professional end users. Others can cast their net further from the community into the network of practice, asking for the information by way, say, of a mailing group. For professional end users, however, the network of practice is largely based on the domain: research scientists and financial consultants attend conferences, subscribe to mailing lists etc. on the basis of their scientific and professional interests. Because of the secondary role of software development within a professional end user's organization, as described in section 2 above, it is unlikely that knowledge of it is considered important enough to be shared via the network of practice.

The final reason, again of particular pertinence to professional end users, is the fragility of the community of practice. In the financial consultancy, in section 2, we described how, when software developers passed all their professional examinations, they became consultants. On the whole, they then stopped being concerned with software development (which they, in their turn, delegated to students): the consultants' software development knowledge was then largely lost. The situation is similar for the research scientists. Generally, the software is developed by people on post-doctoral short-term contracts associated with a particular project. Their aspirations are normally to use these short-term contracts in order to gain experience, and then obtain a permanent job, where they can concentrate on extending knowledge of their domain, and, in their turn, employ people on short-term contracts to develop any associated software.

This fragility of the community of practice was recognized in both organisations. In the financial consultancy, they sought to ameliorate it by means of the manual, as described in 3.2.1.; in the research laboratory, one of the research scientists cited a break in community as being one of the main risks in a scientific project extending over a protracted stretch of time.

3.4. Specific knowledge archives and knowledge management tools

We mentioned above that one of the projects at the research laboratory involved a collaboration. During the course of the field study, these collaborators were in the process of setting up a knowledge archive, apparently intended to store relevant emails and videos, as well as documents. The research scientists in our study appeared unimpressed. The archiving representative, at the time of our interview with him, displayed little enthusiasm: he had no plans to meet his counterparts from other organisations. Other scientists didn't know of the archive, or did know but were not interested. One summed up the general feeling:

> 'I think we all have some reservations about how practical the system is going to be' [research scientist]

In 3.2. and 3.3. above, we discussed the manual intended to archive and share software development knowledge at the financial consultancy, and the difficulties in maintaining it, given the pressures on the developers to concentrate on their primary job to develop software.

The provision of software tools to support knowledge management is currently a hot research topic. Many writers, however, acknowledge that knowledge management involves more than merely the provision of tools, and accept the importance of the community of practice. As Brown and Duguid, 2000, say:

> 'The importance of people as creators and carriers of knowledge is forcing organisations to realize that knowledge lies less in its databases than in its people' [Brown and Duguid, p.120-121]

Henninger, 2001, and Fischer & Ostwald, 2001, concur with Brown & Duguid, 2000, and with Cook and Brown, 1999, that new knowledge, such as knowledge of software development, is created by a community of practice deploying existing knowledge. They argue that adding newly created knowledge into a repository should be integrated into work practices (as was not the case in either the research laboratory or the financial consultancy), and that knowledge should be disseminated at point of need. By making the latter point, they are expressing doubt at the efficacy of formal training, decontextualised from the task in hand; pointing out that potential users may not know that useful information exists, and echoing the concerns seen in our data concerning the retrieval and filtering of large amounts of information. Henninger, considering knowledge management in software organisations in particular, discusses the use of intelligent software agents in making relevant information available at point of need. Both Henninger and Fischer & Ostwald emphasize the role of technology in supporting, rather than supplanting, the community of practice, though both recognize that work practices and culture may have to change to exploit knowledge management tools most effectively, as in the following quotes:

> ‘One of the most critical elements of support for learning organizations is integrating technical solutions into daily work activities. Not only are tools needed for knowledge capture and representation, we also need to design work practices that incorporate the use of existing knowledge as a normal part of the workflow. … Technical solutions need to be informed by social and organizational processes or risk being ignored as irrelevant or distracting’ [Henninger, p.10]

> '[Knowledge Management] requires changing work practices and attitudes to acknowledge the importance of the knowledge worker and the contexts of work in transforming information into capability for effective action' [Fischer & Ostwald, p.64]

3.5. Summary of this section

In section 2 above, we saw that in the two organisations in our field studies (and we have no reason to believe that this would not be replicated in any professional end user organization), software development is an endeavour that is afforded little respect, and that there is a perception that it requires little effort, and that ‘anyone can do it’. We believe, however, that software development does require considerable knowledge and skills, together with an appreciation of the meta-level issues involved aside from merely producing a piece of software. These meta-level issues include testing, which we do not consider in this paper, and sharing knowledge about the software, so that it can be modified in the future where necessary.

In this section, we discussed how software development knowledge, together with awareness of the meta-level issues involved, might be acquired, created and shared within a professional end user community. Based on our discussion, the most useful vehicle for this appears to be the community of practice. Reliance on this community has its drawbacks, however. One of these is the fragility of the community over time. This is especially salient in a professional end user organization, where there is a tendency for people to leave software development behind as they advance up the organisational hierarchy. Another drawback is that communities of practice may not be as good as documents for sharing some sorts of information, such as arcane details. The ideal seems to be to have a strong community of practice, supplemented, where necessary, by documentation.

Given the perception of software and software development within professional end user organisations, we doubt that more formal knowledge management regimes, involving, for example, a heavy reliance on code reuse or a knowledge repository, would be effective. Such regimes involve heavy investment in creating and maintaining the repository or code library, together with major changes in the way software is developed. It does not seem realistic to expect such investment or change in a professional end user organization. As to formal training, we noted in section 3.1 above that even where it exists, it focuses on code rather than on any meta-level issues, and the literature shows a concern, touched on in 3.4, that knowledge disseminated at point of need (for example, again, by a community of practice) is more useful than any formal training.

In our introduction, we referred to our overarching research question as being: how might IT professionals best support professional end users? We now discuss whether the introduction of some elements of software methodologies, as practiced in the professional software development community, might enhance the creating, acquiring and sharing of software development knowledge in the professional end user community.

4. Software methodologies and professional end users

In this section, we firstly explore the issues raised in the literature concerning software methodologies and end user developers, and then discuss how some of the practices of agile methodologies might be useful in this context.

4.1. Some issues raised in the literature

The literature is clear that end user software development poses many problems from the organisational point of view, see, for example, McBride & Wood-Harper, 2002, and Sutcliffe & Mehanjiev, 2004. These problems include concerns about the quality of end user developed software, about its integration into existing systems, and about duplication. One view in the literature is that the adoption of standard software development methodologies may reduce these problems. For example, Taylor, Moynihan & Wood-Harper, 1998, make several suggestions as to how end users might make use of methodologies. They suggest that end users should develop and maintain systems to the same standard as the IT department, or that they should, with guidance from IT professionals, adopt a cut-down version of an IS methodology. This view is challenged in McBride & Wood-Harper, op. cit., who argue, in essence, that Taylor et al. treat end users as being poorly informed IT professionals, whereas end users and IT professionals are fundamentally different, with the focus of end users being on achieving domain tasks with the aid of software, whereas that of an IT department is on the technical quality of the software, the extent to which it follows the prescribed standards and integrates with other software.

In that section of the literature that focuses on end user software development from the end user, rather than from the organizational, point of view, there is some recognition that this development involves more than programming. As Burnett et al., 2004, say:

> 'Giving end user programmers ways to easily create their own programs is important, but it is not enough. Like their counterparts in the world of professional software development, end user programmers need support for other aspects of the software life cycle.' [Burnett et al., p.58]

Like McBride & Wood-Harper, Burnett et al. recognize the distinction between end users and IT professionals:

> '… because end users are different from professional programmers in background, motivation and interest, the end user community cannot be served by simply repackaging techniques and tools developed for professional software engineers.' [Burnett et al., p.58]

The mismatch between the development culture assumed by traditional software methodologies and the reality of professional end user development is further illustrated in Segal, 2003, describing the problems caused by the clash between the necessity of requirements being articulated upfront as opposed to their emergence in successive iterations.

McBride & Wood-Harper suggest that, rather than prescribing software methodologies, software professionals should support end users by providing component technology or tailorable evolving systems.

These ideas are explored further in Morch et al., 2004, who describe a system to support end users in tailoring components at run-time. The basic difficulty here lies with the provision of components and/or basic tailorable systems. Professional end users tend to work in worlds where domain knowledge is very rich, and very difficult to communicate to non-domain specialists, such as professional software developers. Given this, and given the long history of professional end users developing their own software, we believe that scientists, financial consultants and the like, will continue to develop their own software ab initio for the foreseeable future. As we discussed in section 3, it is a moot point as to whether or not professional end users can be persuaded to divert some of their resources from their main task of meeting their professional goals, into improving the software tools required to meet those goals, by constructing their own components, or basic tailorable systems.

Burnett et al., focusing on spreadsheets and administrators rather than on professional end users, implemented a spreadsheet environment with additional features intended to motivate and support end user developers in using some tried and tested software engineering testing techniques such as backwards tracing and assertions. The provision of specific environments, as with the provision of components or tailorable systems, is not necessarily helpful for professional end user developers, given their rich diversity of interest and their comfort with using standard programming languages. We are interested, however, in the idea of importing techniques and practices, tried and tested by professional software developers, into the professional end user development culture. It is this idea that we shall now explore further.

4.2. Agile methodologies and the professional end user

When Taylor, Moynihan & Wood-Harper, op.cit., suggested that end user developers adopt standard software development methodologies, or subsets thereof, we assume that, given the date on which their paper appeared (1998), they were referring to traditional, waterfall-like, staged, document dependent methodologies. We, on the other hand, are more interested in a class of methodologies, agile methodologies, which have seen a great burgeoning of interest within the professional software development community over the last few years. Agile methodologies seem to us to fit better with end user development than more traditional methodologies. For example, Boehm & Turner, 2004, discuss the differences in the perception of quality:

> '[In traditional methods exemplified by the Software Capability Maturity Model] "quality assurance" is defined as "specification and process compliance." Agile methods see quality as essentially customer satisfaction' [Boehm & Turner, p.5].

This seems to us to match with McBride & Wood-Harper's distinction between IT professionals and end users, as described in 4.1. Again, the following quote from Beck, 2000, mirrors the dependence on emergent requirements seen in Segal's field study of research scientists, 2003:

> 'This is an absolute truth of software development. The requirements are never clear at first. Customers can never tell you exactly what they want. The development of a piece of software changes its own requirements. As soon as the customers see the first release, they learn what they want in the second release… or what they really wanted in the first. And it's valuable learning … that can only come from experience' [Beck, p.19]

Agile methodologies are based on the principles laid out in the Manifesto for Agile Software Development, 2001. Agilists (supporters of this manifesto) value

- response to change over following a plan;
- individuals and interactions over processes and plans;
- working software over comprehensive documentation;
- customer collaboration over contract negotiation. (see http://agilemanifesto.org/, accessed August, 2004)

Examples of agile methodologies include the highly articulated and disciplined eXtreme Programming, XP, Beck, 2000, DSDM (Dynamic Systems Development Methods), see http://www.dsdm.org/, accessed August 2004, and the Crystal family of methodologies, Cockburn, 2002.

In section 3 above, we discussed the importance of the community of practice to end user developers. Agile methodologies stress the importance of informal communication and sharing of knowledge within a community of practice (though they don't always use this term). For example, the four values of the agile manifesto are supported by 12 principles, of which one is:

> 'The most efficient and effective method of conveying information to and within a development team is face-to-face conversation' [http://www.agilemanifesto.org/principles.html, accessed September, 2004]

Boehm, 2002, characterizes the main difference between agile and more plan-based methodologies as the following:

> '… agile methods derive much of their agility by relying on the tacit knowledge embodied in the team, rather than writing the knowledge down in [documented] plans' [p.66]

And Boehm & Turner, 2004, comment that:

> 'Many of the agile practices [of XP] – pair programming, daily standup all-hands meetings, shared code ownership, collocated developers and customers, team planning – are as much about developing the team's shared tacit knowledge as they are about getting the work done' [Boehm & Turner, p.33-34]

As an aside, one might query the use of the term 'tacit' here: it is clear that Boehm & Turner are referring to undocumented, rather than unarticulated, knowledge.

We now explore which agile practices might usefully be imported into a professional end user environment, with a view to strengthening the community of practice. The first of these is, we think, pair programming. It is entirely plausible that pair programming will foster the dissemination of software development knowledge throughout the organization and strengthen the community of practice. It should also provide redundancy in the sharing of knowledge, so that the loss of a key developer might not be accompanied by a concomitant loss of key development knowledge. There is considerable evidence in both the financial consultant and research scientist case studies, of professional end users helping each other, so institutionalized pair programming might just be a natural extension of this phenomenon. As to whether or not pair programming constitutes a tried and tested practice, Cockburn & Williams, 2001, and Wood & Kleb, 2003, produce evidence that the productivity of having pairs program the same task is very similar to that of using a single programmer, and any (small) extra cost is justified by the quality of the code so produced.

Another practice which might usefully be adopted in professional end user development communities is that of adherence to coding standards. This should make the code easier to comprehend, and hence easier to modify. Although this practice might appear to reflect sound common sense, it may prove surprisingly difficult to implement: we saw in Section 3, that the professional end user community in the research lab did not even adhere to a common programming language.

The final practice which we suggest be adopted by professional end user developers is that they should plan to deliver software in small releases in an evolutionary fashion, thus supporting emergent requirements as in the quote from Beck above.

Given that the focus of professional end user developers is on accomplishing their professional tasks rather than on polishing their software, we have hesitated to suggest any practices which require significant differences in the way they currently work, or impose a significant discipline. Based on data from our field studies, we believe that the organisational adoption of the practices of pair programming and evolutionary delivery of small releases merely regularizes the way that professional end user developers currently work. In addition, we do not believe that adherence to common coding standards imposes an intolerable burden, though this belief remains to be tested in the field.

5. Summary and Discussion

This paper aims to provide a partial answer to the question posed in the introduction, viz. : how might IT professionals best support professional end user developers?

We firmly believe that answering such a question must be dependent on data gathered in the field as to the actual nature of the practice of professional end user development. In section 2 herein, we describe how the culture within which professional end users develop software differs from that of professional software development. In the former case, the effort and knowledge required to develop software is not recognized, and this lack of recognition is reflected in inadequate resource allocation. We posit that one area in which IT professionals might support professional end user development is by providing support for the acquisition, creation and sharing of software development knowledge. In section 3, we discuss various means by which this support might be provided, together with the actuality revealed in our field studies. These include

- formal training. But there may not be resources allocated to this, and even if there are, there are concerns that it may only focus on coding (which is not usually a problem for professional end users), and may be too separated from the point of need.
- Documentation. But there are various problems involved with this formal articulation of knowledge, including those of search, retrieval and deployment. In addition, there is the cost of writing the documents in the first place.
- Self-documenting code/reusable code. The major problem here is the investment of resources required to produce self-documenting and/or reusable code, as opposed to code which simply does the job in hand. A further problem is that introducing a program of reuse involves acknowledging the importance of software and changing the development culture and the practices therein.
- Formal knowledge archives/knowledge management tools. Like self-documenting/reusable code, the maintenance of these require both an investment of effort and a change in practices and culture.
- Community of practice. This seems to be the most useful way of acquiring, sharing and creating software development knowledge within a professional end user community, but it has several problems. Foremost among these is the fragility of the community together with the fact that some types of knowledge do not lend themselves to informal sharing.

We then explored the question of whether IT professionals could support professional end user developers by exporting some of their tried and tested practices. We believe that some of the practices of XP, pair programming, adherence to coding standards, incremental evolutionary development, could be adopted by professional end user communities with little perturbation to their normal ways of working and with the effect of strengthening the community of software development practice. It remains to test this belief in the field.

We discussed in section 3 the fact that the community of practice might be complemented by documentation. This issue (when documentation is necessary; what form it should take) is currently actively engaging those software practitioners who espouse agile methodologies (see, for example, Cockburn, 2002, and Scott Ambler, http://agilemodeling.com/essays/agileDocumentation.htm, accessed September, 2004), and those who seek to meld traditional and agile methodologies according to the context of an individual project, such as Boehm & Turner, 2004. We intend following this debate closely with a view to ascertaining whether it might further inform the problems of professional end user development.

References

Beck, K. (2000). eXtreme Programming Explained: Embrace Change. Addison Wesley.

Boehm, B. (2002). Get ready for agile methods, with care. Computer, 35(1), 64-9.

Boehm, B. & Turner, R. (2004). Balancing Agility and Discipline. Addison-Wesley

Brown, J.S. & Duguid, P. (2000) *The Social Life of Information*, Harvard Business School Press.

Burnett, B., Cook, C. & Rothermel, G. (2004) End user software engineering. CommACM, 47(9), 53-58

Cockburn, A. (2002). Agile Software Development. Addison Wesley.

Cockburn, A. & Williams, L. (2001). The costs and benefits of Pair Programming. In Succi, G., Marchesi, M. Extreme Programming Examined. Addison-Wesley, Reading MA.

Cook, S.D.N. & Brown, J. (1999). Bridging epistemologies: the generative dance between organizational knowledge and organizational knowing. Organization Science, 10(4), 381-400

Downey, J.P. (2004). Towards a comprehensive framework: EUC research issues and trends. Journal of Organizational and End User Computing. 16(4), 1-16.

Fischer, G. & Ostwald, J. (2001). Knowledge management: problems, promises, realities and challenges. IEEE Intelligent Systems, 16(1), 60-72.

Henninger, S. (2001). Organizational Learning in Dynamic Domains. Advances in Learning Software Organisations. Althoff K-D, Feldmann R.L. & Muller W (eds.). LNCS 2176, Springer-Verlag, pp 8-16.

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

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

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

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

Paulk, M. (2002). Agile methodologies and Process Discipline. Crosstalk, The Journal of Defense Software Engineering, 15(10), 15-18

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

Powell, A., & Moore, J.E. (2002). The focus of research in end user computing: where have we come since the 1980s? Journal of End user Computing, 14(1), 3-22.

Segal, J. (2001). Organisational learning and software process improvement: a case study. Advances in Learning Software Orgnizations, LSO 2001. Springer-Verlag LNCS 2176, 68-82.

Segal, J. (2003). When software engineers met research scientists: a field study. Technical report 2003-14, Department of Computing, Open University, Milton Keynes, MK7 6AA, UK. http://computing-reports.open.ac.uk/index.php/2003/200314

Sutcliffe, A. & Mehandjiev, N. (2004) End-User Development. CommACM, 47(9), 31-2

Taylor, M.P., Moyniham, E.P. & Wood-Harper, A.T. (1998) End user computing and information systems methodologies. Information Systems Journal, 8, 85-96.

Wood, W.A. and Kleb, W.L. (2003) Exploring XP for Scientific Research. IEEE Software, 20(3): 30-36.