A logical mind, not a programming mind: Psychology of a professional end-user

Alan F. Blackwell

Computer Laboratory
Cambridge University
Alan.Blackwell@cl.cam.ac.uk

Cecily Morrison

Engineering Design Centre
Cambridge University
cpm38@cam.ac.uk

Keywords: POP-I.A. social organisation and work, POP-I.C. health records POP-II.A. end-users

Abstract

This paper reports a case study of a specific end-user programming context, in which an electronic patient record system was being customised by a healthcare professional. Our research involved making an unusual intervention, employing a professional programmer as a quasi-experimental participant, in order to explore and contrast the different ways that the same situation was conceived by an end-user programmer and by a professional programmer. We found a range of pragmatic strategies that were employed by the end-user, causing her to resist some conventional views of how programs and source code should be interpreted. Rather than different 'cognitive styles', we believe these differing mental models can be accounted for by the context of the practical work the two need to achieve, and the organisational contexts within which they work. We make some observations and recommendations about the design of tools for end-user programmers, extrapolating from our in-depth observation of one particular product.

1. Introduction

This research investigates a context in which a complex software product is expected to be customisable by (some of) its users. The customisation facilities offered by the product include a range of scripting and programming facilities, some quite sophisticated. Although the product supplier intends that end-users should be able to use most of the customisation facilities, in practice some of these facilities would be more familiar to professional programmers. We present a case study in which we compare the experience of a non-programmer end-user who is responsible for local customisation of this product, with the experience of a professional programmer whom we employed temporarily in order to carry out our research. The interaction between the two can be contrasted with the types of situation studied by Segal (2005), in which scientific end-user programmers interact with professional software engineers. In Segal's research, the scientists being studied are confident programmers, but faced challenges when asked to integrate their work practices into more structured software engineering processes. In our research, it is the end-user who is most engaged with structures of professional practice (in this case, healthcare rather than engineering), while not being familiar with programming language concepts. In her own phrase, she has 'a logical mind, not a programming mind'.

From a psychology of programming perspective, it is well understood that end-user programmers and professionally-trained programmers are likely to have different mental models of programming tools. If that is understood, then one might expect that interaction between the two will reveal systematic challenges. In this paper we are interested both in the pragmatic consequences of those challenges – what can be done to assist such collaborations – and also in the opportunity to understand more about the mental models on both sides, by analysing the content of the conversation.

2. The Case-Study Context

We conducted this research in the Intensive Care Unit (ICU) of a specialist cardio-thoracic hospital, over a period of three years from the summer of 2006 until 2009. At the start of this period, the director of the ICU had agreed a contract to purchase and deploy an electronic patient record system (EPR). In a research-active hospital such as this one, new clinical initiatives are routinely accompanied by a research agenda. The ICU director therefore approached the University of Cambridge to identify researchers likely to have an interest in EPR deployment, and a research team was assembled via the Crucible network for research in interdisciplinary design. That team included researchers in Management Information Systems, Social Psychology, Anthropology and Computer Science. A number of studies have been conducted during the study period, but here we describe only one of these, conducted primarily by the two authors.

The EPR that had been selected after a competitive tender was MetaVision ICU, a product from Israeli company IMDsoft. Among the selection criteria applied by the ICU director, features supporting end-user customisation were a high priority. The shortlisted candidates had included another EPR product that had previously come to the attention of the End-User Software Engineering research community (Orrick 2006), meaning that end-user programming was identified from the outset as a research question of potential interest.

Over the following months, we observed the arrangements for deployment and commissioning of the product, the product training conducted by IMDsoft personnel, the customisation of the standard product to fit local practices, the operational training of the ICU nurses and clinicians, and the transition from paper records in the ICU to full reliance on the EPR. We also continued to observe over the next year as the system 'bedded-in', and became integrated into the ICU operation. In the third year, we observed ongoing maintenance and customisation work as an aspect of the routine operation of an ICU-based EPR. We also participated in convening a user group of other UK hospitals that were deploying MetaVision. The user group meets biannually in Cambridge, and provides an opportunity to compare the experiences of other hospitals to the one in which our case study has been located.

3. Programming Intervention

In the final six months of this study, we decided to experiment with an unconventional research technique, in order to gain a new perspective on end-user programming. By this stage, two members of the ICU staff had assumed primary responsibility for ongoing customisation and systems management of the EPR. The ICU director, a consultant anaesthetist, takes responsibility for medical customisation, while the EPR manager takes responsibility for customisations related to nursing, and for systems management. The EPR manager, C., had by this time become familiar with the limitations of the product, and with the constraints that it placed on the kinds of customisation she could achieve. In some cases, problems could be solved with assistance from IMDsoft technical support staff (although often remotely, from Israel). In other cases, it would have been possible to commission product customisation work from IMDsoft, but this was an expensive option, not anticipated in the ICU operating budget, and had only been done once since the EPR had been deployed.

We therefore proposed that we should employ a professional programmer for a short period of time, to provide an opportunity for a new research study. We advertised in a local forum for freelance programmers, and recruited J., a programmer with broad experience of Visual Basic-like scripting within a database environment, of the kind that the MetaVision system employed. We intentionally recruited a programmer without prior experience of MetaVision, of ICU practices, or of clinical computing more generally. The reason for this was that we wished to observe, record and analyse all conversations between J. and C., as a situation where each of them would need to make their understanding of the system very explicit to the other. We saw this as an opportunity to contrast the mental model of an experienced end-user programmer with that of an experienced professional programmer. We recorded all conversations between the two participants, and used those verbal protocols as our primary source material for analysis. The main purpose of our research intervention was thus to set up a situation in which we could elicit descriptions of mental models in a natural

manner, rather than through think-aloud protocols. This research method is intended as a variant of 'constructive interaction' – a dialogue-based elicitation technique (Miyake 1986) that has previously been applied with some success in HCI (Kahler, Muller & Kensing 2000)

Our research intervention, in addition to eliciting dialogue for later analysis, also provided us with an opportunity to study a realistic professional situation. It is often the case that end-user programmers, when faced with a specific technical problem, or a design task that is outside the scope of their experience, will seek assistance from a more experienced or technically knowledgeable professional. We believed that it was important to gain more understanding of this kind of situation, and that observing a specific and well-defined task over a defined period of time would provide a reasonably well controlled opportunity for doing so.

The specific problem indentified by C. as a focus for this intervention was the creation of a more versatile reporting mechanism by which a paper report could be created with information about a specific patient. MetaVision has many standard reporting facilities, all of which are customisable, but it had proven impossible to create a particular report incorporating historical information on a single patient in a readable format. (In a previous publication (Morrison & Blackwell 2009), we have described the extent to which the ICU regularly customised its paper forms). The one piece of customisation work commissioned from IMDsoft in the past had, in fact, been the development of a particular customised report. However, the customisation of reports is a sufficiently common activity that it was not desirable to have this amount of expenditure every time a report was customised. C. had been disappointed, after that earlier experience, with the fact that the custom code created by IMDsoft was not delivered in a form that she was able to further modify it herself. The practical motivation for employing J. was therefore to observe the report customisation process, and if possible, learn to do it herself in future. This meant that there were two practical objectives – one short-term, in which J. might create a new customised report, and one long-term, in which C. might learn to create such reports herself in future. There is clearly a degree of conflict between these. On reviewing a draft of this paper, J. wished to emphasise that he could easily have achieved the first goal, and that it was the introduction of the second that gave rise to many of the results we report. This is precisely the dynamic that we consider to be interesting. As noted, C. had already commissioned a programmer to carry out system customisation, but now wanted to learn how to achieve the same results herself. We believe that this experimental situation, although to some extent manufactured through our intervention, is also typical of end-user programming experiences.

Our research budget provided resources to employ J. for a total of five days. Approximately 40% of this time was spent away from the ICU, carrying out background research into the MetaVision features and architecture, technical preparation and some exploration of potential approaches. The remainder of the time was spent in C's office at the ICU. The second author observed all of the periods in which C and J worked together, and recorded all conversations for later transcription. Both authors observed the final day of work, at which the results were reviewed and 'handed over'. At the close of the day, C and J were interviewed together, reflecting on their experience of the project. As before, these conversations were recorded and transcribed.

After transcription of all the above conversations, the two authors independently coded the transcripts, and then jointly reviewed those codes and their interpretations. An initial analysis of the project results has been prepared for a clinical audience (Morrison et.al., in press). The draft manuscript presenting results of that analysis was reviewed by C., and each author discussed with her the interpretations that we had drawn, modifying any misinterpretation or misunderstanding. The remainder of this paper presents the findings of our analysis as they relate to end-user programming more generally, rather than the specific clinical context. We also draw on observations made through the previous years of our case study, and on field recordings and notes by both of us, where necessary. As with our earlier report for a clinical audience, this paper has also been reviewed in manuscript by C. and by J. Both of them responded with detailed comments (C. returned a marked up manuscript), and we have taken those comments into account in our analysis. In the following text, we specifically indicate those situations in which we had initially drawn a particular interpretation, but one or both of them asked us to correct it.

4. Findings

4.1. Organisational context

As reported in more detail in (Morrison et.al., in press), this study revealed unanticipated aspects of the role played by end-user programmers within an organisation. Our observations reinforce the importance of the 'EUSES' view as advocated by Ko et al. (in press), which is that end-users must engage in all aspects of end-user software engineering, rather than simply end-user programming. However, the software engineering factors that arose in our research suggest some alternative emphases to those described in the survey by Ko et al.

The main theme of our findings in this area could be summarised as 'end-users have users too'. Despite not being professionally trained as a programmer, C. has assumed responsibility within the ICU for operation, maintenance and customisation of the EPR. Because she is the person with the most technical knowledge of the system, and because the system is known to be customisable (and to have been customised by her), she receives 'feature requests' from the other ICU staff, with proposals or recommendations for further customisation. In responding to those requests, she must not only consider programming work, but also business process analysis, user interface design, and many other aspects of software engineering that in a professional software engineering team are considered to require special training and experience for people taking particular roles within the team.

Once C. has identified necessary changes, either in response to requests, or as part of her own continuing refinement and enhancement of the system, she must plan the deployment of those changes. As with many end-user customisable systems, the customisation interface is a part of the live system. However, an ICU operates on a 24-hour, 365-day basis, as with many critical business systems that have dedicated software engineering teams. In such businesses, it is normal to maintain three parallel systems: one 'production' system that business users actually interact with, one 'test' system configured in the same way as the production system, and one 'development' system that the programmers interact with. Complex versioning and configuration control is maintained between these systems, with specialist teams dedicated to each of the different systems, and to the overall build and release process. In our case study, C. was responsible for all three kinds of work, but using one system for all of them, and with few tools for version management and configuration control. As noted in our earlier publication for the clinical audience, this situation is far from ideal, and ought to be addressed in future tools for end-user software engineers.

Furthermore, it should not be assumed that the same tools used for these purposes in professional software development organisations will also be appropriate in an end-user software engineering context such as this. C. herself, despite having users of her own, does not regard herself as a professional software engineer. When she engages with IMDsoft support staff, or in her collaboration with J., she approaches her work very much in the manner of an end-user, with user concerns her key focus. This places her constantly in a kind of border-land, where none of the people she interacts with are exactly peers. This is another respect in which she is unlike professional software developers, who usually work within teams of professional peers sharing their own assumptions and practices. At many points in our observation, it was clear to us that C. did not share the assumptions and practices advocated by J., and had no desire to do so. For a new software engineering tool to be of value to her, it should be possible for her to adapt that tool to her particular working practices, rather than requiring her to abandon them.

4.2. Mental models and motivation

In analysing the conversation between C. and J., we were particularly interested to identify differences in the way that they conceived of programming tools and the programming task. This aspect of our findings is the one that is most clearly related to established research concerns in psychology of programming.

We were impressed by how actively C. resisted the conventional programming conceptions of system behaviour. She described large parts of the system behaviour as involving technical facilities that were 'in the background'. Most of the operation of the underlying database management system, for

example, fell into this category. She was aware of certain behaviours of this system, for example in carrying out data integrity checks, but did not want to be distracted by explanations or investigation of those behaviours. She clearly regarded these as important, but as lying within the domain of responsibility for IMDsoft. It would not be a good use of her own time to learn about aspects of the system that other people will be responsible for. We found that this highly pragmatic attitude of a busy and responsible person was in contrast to a typical technical attitude, occasionally expressed by J., which is that aspects of system behaviour are interesting in themselves, and that it might always be useful to know a little more about them for future use. C. regularly referred to these aspects as being 'esoteric' interests.

In part, this distinction is encouraged by IMDsoft, whose product documentation draws a clear distinction between the system model that must be understood in order to do customisation work, and the underlying technical behaviour of the system. As a company with long experience working with a particular community of end-user developers, their design approach seems to be an appropriate one. In the psychology of programming community, this has been described as 'the black box inside the glass box' (du Boulay, O'Shea & Monk 1981). However the work of du Boulay et al. was oriented toward the teaching of conventional programming languages. In this case, the end-users do not want or need to know about a conventional programming model, but a customisation model. For the purposes of tool design, the glass box must provide a virtual machine suitable for customisation work – but there are challenges in achieving this in the MetaVision product, as we discuss below.

C. is quite willing to spend time learning about aspects of the system that may not be directly relevant to her current task. In this respect, she is not completely captive to the 'paradox of the active user' (Carroll & Rosson 1987), and is able to make attention investment decisions with longer term payback (the attention investment model, previously presented at PPIG, describes the way in which some people choose whether to construct abstract/programming models of a task based on assessment of future payback – see Blackwell& Green 1999, Blackwell 2002). C described a conscious strategy of finding opportunities to make basic modifications in an area where she hadn't worked before, in order to open up possibilities for future enhancements, or understand the potential for new uses of the system capabilities within the ICU. However, these are very specific decisions, such that the various factors involved in the attention investment model become partitioned between areas of the system functionality. There are some aspects of the system where C. has done a significant amount of 'tinkering' (Beckwith et.al. 2006), and as a result has developed a high degree of self-efficacy with regard to that particular aspect. But at the same time, there are areas of system functionality that she avoids, and where she was not confident to make changes or 'tinker'. In terms of the early 'garden shed' analogy for tinkering behaviour (Blackwell 2006), C. is happy tinkering in certain parts of the shed, but there are others that she avoids.

Perhaps the most distinctive aspect we observed of the approach taken by C. to the end-user customisation tools was the way that she approached reading of the source code itself. A number of the MetaVision facilities (some report generation tools, a 'query wizard', event trigger rules and so on) use relatively conventional programming language syntax, generally quite closely related to Visual Basic. After several years of use, C. had become perfectly accustomed to reading, modifying and adapting chunks of this code, but the way in which she did so was quite different to J's reading of the same chunks. Whenever we observed the two of them reading and discussing an extract of source code, it seemed that any given word of text that was considered interesting by C. would not be considered interesting to J. and vice versa.

When reading source code, C. generally found the identifier names meaningful, and treated them as most likely to reveal the function of that code segment, and offer clues on how to modify or customise it. She did not generally comment on programming language keywords, control constructs, type declarations or syntax elements. Our impression was that these aspects of the code were relatively uninteresting, were not perceptually salient, and perhaps even annoying distractions. At one point where J. drew attention to them, she commented that these were typical of his 'esoteric' interests. J., on the other hand, like most professional programmers, was adept at reading past the (technically speaking, arbitrary) names chosen for identifiers, in order to read the underlying structure and

behaviour of the code. In his reading, it was the keywords, control constructs, syntax and so on that were the important topics of conversation.

Of course we are not suggesting that either C. or J. are unaware of the importance of those aspects of the source code that the other found more salient. In particular, the way that we had set up our research intervention had left J. with a partial impression that we wished to study him 'teaching' C. more about programming (we address this further below). This may have caused him to emphasis formal features of the language in discussion precisely because he was concerned that she had been paying insufficient attention to them. Nevertheless, we believe that the design of programming tools for end-user programmers may have paid insufficient attention to their need to focus on their own domain as the primary feature of the program representation.

There is a further possibility that these different orientations toward the code arise from applying different personal styles within the professional context, of the kind reported by Wray (2007) and Hudson (2009). As might be expected from their respective professional backgrounds, we found that C. was more likely to describe the functions of the system in terms of people, and in terms of particular roles that people play within the operation of the ICU. In contrast, J. was more likely to describe the system functions from an abstract perspective, describing not only technical features, but even the illustrative examples he chose in terms of data ontologies and abstract functions. These alternate perspectives are completely typical of the end-user versus programmer orientation, but they also demonstrate a contrast between characteristically empathising and systemising styles, as observed by Wray and Hudson. Furthermore, they exhibit a contrast between human/social and computational thinking, as discussed by Blackwell, Church and Green (2008). As C. stated when reflecting on her own cognitive style, this experiment has led her to believe that she has 'a logical mind, but not a programming mind'.

All of these factors lead C. to regard programming, and investing time in learning about programming, as an activity that although it has extrinsic benefits, does not have intrinsic ones (Nash, Church & Blackwell, submitted). She described herself as becoming 'impatient' in the course of the collaboration with J., because he persistently discussed topics that appeared to her not to be relevant to the work at hand. To some degree, this resulted from the central concern of the study, which was the extent to which C. might be able to learn how to do work of this kind in future. Although this had been an objective from the outset, time spent discussing 'educational' topics was also perceived by C. as time in which she was not making progress toward the immediate goal of creating the report. As a result, J's priorities appeared to her to be motivated by an attitude to programming as having some intrinsic satisfaction, independent of the results achieved. This contrast between the priorities of technical and non-technical users has been a common theme in past work that has emphasised the importance of providing tools for end-users that offer visible progress toward achieving the user's own goals, rather than teaching abstract principles with no clear relationship to user priorities.

4.3. The collaborative relationship

Although the collaboration that we set up between C. and J. was in many ways an artificial one (created explicitly as a component of a research project, with an explicit research agenda, paid for by research funding), we believe that the conversations were quite typical of settings in which short-term collaborations occur between end-user and professional programmers. Furthermore, in those settings, the quality of the collaborative relationship is of interest because it can have direct commercial value. We therefore analyse this relationship more closely.

As noted in our description of the way in which the study was set up, there was a fundamental tension between the immediate goal of creating a customised report, and the longer-term goal in which C. would learn how to create such customised reports herself. The first goal required J. to carry out conventional professional programming work, while the second required him to act in the capacity of a teacher. As he observed when reviewing a draft of this paper, he does not have teacher training or experience, so we could not have expected that he would find this aspect of the project easy. We agree that it may be unrealistic to expect professional programmers also to be professional teachers of programming. Nevertheless, this situation is one that arises relatively frequently in the experiences of

end-user programmers. As end-users, they have often not received professional training in programming. Instead, they acquire knowledge from a variety of sources, including programmers such as J. who have relevant technical knowledge but no teaching experience.

Although almost certainly exacerbated by the artificial nature of the research project, we observed a significant degree of discomfort in the collaborative relationship between C. and J. Based on the discussion in the previous section, we believe that this arises to a large extent from the different psychological styles that are characteristic of 'programming' work and 'user' work. As described by Bucciarelli (1994), technical professionals work within an 'object world' rather than one of primarily human relations. However in professional software development organisations, there are a number of specific technical roles dedicated to bridging these two worlds, such as systems analysts, technical authors, helpdesk operators and so on. Each of these professions has an established culture, tools, methods and practices that allow them routinely to carry out this bridging work. In the case of an end-user developer such as C., her professional role does not include the cultural repertoire of tools and methods that support easy interaction with professional programmers such as J.

For his part, J. accepted his responsibility in part as an educational one, providing C. with valuable skills that would enable her to be more effective when carrying out future programming work. Given that the two had comparable professional seniority, this educational objective often amounted to a clash of alternative intellectual styles. In the educational approach taken by J., the future development needs described by C. could be achieved more effectively by adopting more abstract analysis and problem-solving strategies – strategies that are completely characteristic of programming work. However, as described in the Attention Investment model (Blackwell 2002), abstract strategies can be expensive ones, requiring greater initial investment of attention, with potentially low return, or even negative return. When pursuing the educational objectives in his collaborative relationship with C., J. was therefore acting as an advocate of a more abstract strategy, attempting to persuade her that she should adjust the attention investment decision factors that she already applied. In many of their conversations, C. resisted this advice, saying that she would prefer to make progress on the specific, concrete, problem that had motivated the project in the first place. For his part, J. tried to persuade her that it would be in her long-term interest to address not just this particular problem, but the general category of problems of this type, or indeed the general capabilities to be acquired by gaining experience of the most general capabilities of the programming language.

In this last respect, the conversation between the two became a tutorial discourse, in which J. spent time explaining the fundamental concepts of programming languages, such as classes and functions. In doing so, he used the same kinds of example that are routinely presented in programming language textbooks. To explain the notion of class type and aggregate relationships, he talked about the parts of a car, and to explain functions, he constructed an example based on financial calculations. Although these examples may have been effective tutorial illustrations of the necessary abstract principles, C. was not necessarily motivated to acquire those abstract principles in the first place. The fact that the tutorial examples were so far removed from her own problem domain exacerbated her impression that formal aspects of programming were esoteric and irrelevant, and fuelled her impatience with the collaboration as a whole.

After reviewing a draft of this paper, J. explained that he would certainly have preferred to use teaching examples from the ICU itself, but felt that this would be risky, given the very limited time that he had to become familiar with that domain. Although both collaborators felt frustrated by the constraints arising from the short-term nature of our study, it is also typical of end-user development dynamics, where it is always the case that the end-user is the person with the most complete understanding of their own domain, and professional programming assistance, when available, will come from people with less understanding of the end-user's own work. However, even where J. explored a particular approach in terms of C's own domain tasks by re-implementing an existing function (a common programming strategy), C. commented in her review of our draft paper that this had seemed especially unproductive, when the time could have been spent implementing new functionality.

Our study is open to critique, regarding the extent to which this collaborative relationship can be taken as generally representative of end-user programming experiences. Although C. has extensive responsibility for customisation, her experience specifically of programming is relatively limited. This experiment, casting her in the role of a potential end-user programmer, was therefore not completely typical of her work. J. also has extensive experience of working within client organisations, alongside end-users, but has not in the past been expected either to teach programming skills to those end-users, or to create productive collaborative relationships in such a short period of time. He commented that his typical contracts would include time to learn about the environment before technical work starts. Both entered into this experiment with an open attitude to collaboration, and it is likely that the challenges they encountered in working together arose in part from the experimental constraints that we had placed on them. Nevertheless, we believe that the experiment has been more illuminating of actual professional practice than most laboratory studies, and that the combination of skills and experience that we encountered within this specific professional situation is one that is often encountered in real end-user programming contexts.

5. Implications for design

In earlier sections of this paper, we have noted aspects of the MetaVision tool that could be improved to support C's working environment and work processes. In a number of cases, such as the need for change control and configuration management, these provide evidence for the importance of the end-user software engineering perspective. End-user programmers, if working in a professional environment, are very likely to need software engineering facilities like these. Failure to provide them could be taken as a lack of respect for the professionalism of MetaVision customers, because MetaVision's own software engineers would certainly not be prepared to work without such facilities. Many software companies are concerned to avoid this kind of attitude among their developers, which has led to the practice described as 'eating your own dog food', where developers are required to use their own products. In addition to the lack of change control features, we observed a number of annoying deficits in the MetaVision customisation tools that would probably not have been considered acceptable in professional development tools, such as screen space restrictions meaning that it was not possible to view a complete form layout at the same time as running the form customisation tool. We recommend that companies creating end-user programming tools should regularly use those tools, rigorously comparing the features and limitations to those of the tools used by the company itself.

We observed a number of issues related to the use of names in the MetaVision product. As discussed above, it is identifier names that C. finds particularly salient when viewing source code of customisation scripts. This suggests that the choice of identifier names is particularly important. Unfortunately, customisable products with database back ends seem often to impose restrictions on the choice of identifier names, and make it difficult to change these names, where they correspond to aspects of the underlying database schema. We have previously noted the problems this caused during the deployment of a student record system in Cambridge (Blackwell, Church Green 2008). In the case of MetaVision, some identifier names can be altered when the system is first installed, but become embedded through scripts over time, such that they are very difficult to change or modify. It might be sensible to avoid this kind of premature commitment, by separating identifier names from the behavioural relations they describe, so that they can be modified if necessary without breaking the system.

We also observed some problems with the names of built-in functions, although these may have arisen in part from the fact that the product was developed by engineers who were not native English speakers. One example is that in MetaVision, the table of site-specific data values, which can be extended and used in scripts and forms by the end-user, are described as 'parameters'. In terms of the design model of MetaVision's own engineers, this no doubt refers to the aspects of the system behaviour that can be 'parameterised' – customised at each customer site. However, the term 'parameter' is confusing to MetaVision users. To users without previous programming experience, the word 'parameter' is relatively meaningless, and hence unhelpful. To end-users who do have previous programming experience, the word is confusing, because in their view of the system, these values do

not act as parameters. In the collaboration between C. and J., the word 'parameter' was a source of confusion, because when J. was assuming his educational stance, he tried to describe the conventional definition of a parameter in the context of functions and classes, but these explanations did not correspond to C's experience of the product. In order to avoid problems of this kind, we therefore recommend that end-user development products should not use existing programming language terms unless they are implementing a feature that has precisely the expected technical meaning.

As might be expected in a project of this kind, we noted a number of relatively routine usability problems with various aspects of the MetaVision system. Some might be addressed by using techniques borrowed from other end-user programming research – for example, the scripting language used for event processing looked as though it would have been easier to use if it had presented a publish/subscribe event model such as that used in the Scratch language, and the report scripts could have benefited from a better way of defining layout constraints subject to varying amounts of data on the page. Others appeared to have more fundamental problems, as in the case of the Query Wizard, which all MetaVision users appear to find problematic. In that case, it was interesting for us to observe the workarounds that C. had adopted to gain some value from the tool, using the wizard to generate samples of script code that she then cut and pasted into other parts of the system, making minor adaptations if necessary. This generate-and-paste approach, although somewhat clumsy by comparison to the intended operation of the Wizard, was in practice both pragmatic and empowering.

In the course of observing the collaboration between C. and J., we also saw some unexpected and positive uses of tools. When explaining the operation of Visual Basic within a Word macro, J. demonstrated the code execution by following it in the VB debugger. C. had not seen this view of code before, and found it very compelling as a visualisation of the system behaviour. We suspect that a simple script debugger could do much to assist more sophisticated end-user developers. In combination with the Wizard strategy, one might imagine an approach to scripting in which candidate scripts are generated, and their operation is animated, as an alternative to programming approaches in which code is generated from scratch, relying on an execution model that the programmer is supposed to have learned and internalised before starting work.

6. Conclusions

We have presented findings from a field study of a real customisation project that involved collaboration between a professional programmer and an end-user programmer. The context in which they were working drew attention to the ways in which end-user programmers working within an organisational context can have many of the same responsibilities as professional programmers (for example, they have users too), yet without access to the same tools that professional programmers take for granted. We observed a number of differences between the approaches taken by the two participants in our study, providing rich illustrations that tend to confirm previous research observations regarding the challenges of end-user programming relative to professional programming. However, we also observed a number of ways in which the collaboration itself is problematic, as a result of significant differences in mental models and in strategic approaches to technical problems. We believe that collaborations of this kind are relatively frequent, though little studied, and that further research is justified. Finally, we have developed a research method that is relatively novel in psychology of programming research, involving a funded intervention in an organisational context, with ethnographic observation and analysis of the resulting collaboration.

7. Acknowledgements

This research was funded by Boeing Corporation. We are grateful to our collaborators at the ICU described in this study, including ICU director Alain Vuylsteke who initially invited us to study the deployment of the MetaVision system, and especially C. and J. who have been both patient and tolerant to a generous degree.

8. References

Beckwith, L., Kissinger, C., Burnett, B., Wiedenbeck, S., Lawrance, J., Blackwell, A. and Cook, C. (2006). Tinkering and gender in end-user programmers' debugging. In *Proceedings of CHI 2006,* pp. 231-240.

Blackwell, A.F. & Green, T.R.G. (1999). Investment of Attention as an Analytic Approach to Cognitive Dimensions. In T. Green, R. Abdullah & P. Brna (Eds.) *Collected Papers of the 11th Annual Workshop of the Psychology of Programming Interest Group (PPIG-11),* pp. 24-35.

Blackwell, A.F. (2002). First steps in programming: A rationale for Attention Investment models. In *Proceedings of the IEEE Symposia on Human-Centric Computing Languages and Environments*, pp. 2-10.

Blackwell, A.F. (2006). Gender in domestic programming: From bricolage to séances d'essayage. Presentation at *CHI Workshop on End User Software Engineering*.

Blackwell, A.F., Church, L. and Green, T.R.G. (2008). The abstract is 'an enemy': Alternative perspectives to computational thinking. In *Proceedings PPIG'08, 20th annual workshop of the Psychology of Programming Interest Group*, pp. 34-43.

du Boulay, J.B.H., O'Shea, T. and Monk, J. (1981). The black box inside the glass box: Presenting computing concepts to novices. *International Journal of Man Machine Studies*, 14:237-249.

Bucciarelli, L.L., (1994). *Designing Engineers*. MIT Press.

Carroll, J.M. and Rosson, M.B. (1987). Paradox of the active user. In: J.M. Carroll, Editor, Interfacing Thought: Cognitive Aspects of Human–Computer Interaction, Bradford Books/MIT Press, pp. 80–111.

Hudson, W. (2009). Reduced empathizing skills increase challenges for user-centered design. In *Proceedings of the 27th international Conference on Human Factors in Computing Systems (CHI'09),* pp. 1327-1330.

Kahler, H., Muller, M., Kensing, F. (2000). Methods & tools: constructive interaction and collaborative work: introducing a method for testing collaborative systems. *Interactions* **7**, 27-34.

Ko, A.J., Abraham, R., Beckwith, L., Blackwell, A.F., Burnett, M., Erwig, M., Lawrence, J., Lieberman, H., Myers, B., Rosson, M.-B., Rothermel, G., Scaffidi, C., Shaw, M., and Wiedenbeck, S. (in press). The State of the Art in End-User Software Engineering. Accepted for publication in *ACM Computing Surveys*.

Miyake,N. (1986). Constructive interaction and the iterative process of understanding. *Cognitive Science* **10**, 151-177.

Morrison, C & Blackwell, A.F. (2009) Observing end-user customisation of electronic patient records. In V. Pipek, M.-B. Rosson, B. de Ruyter and V. Wulf (Eds). *Proc. 2nd International Symposium on End-User Development, IS-EUD'09*. Springer Verlag (Lecture Notes in Computer Science - LNCS 5435), pp. 275-284.

Morrison, C., Blackwell, A. and Vuylsteke, A. (in press) Practitioner-Customizable Clinical Information Systems: a case-study to ground further research and development opportunities. Accepted for publication in *Journal of Clinical Engineering*.

Nash, C., Church, L. And Blackwell, A.F. (submitted). Designing Computational Tools for Creativity, Flow and Mastery. Paper submitted to *VL/HCC 2010*.

Orrick, E. (2006). Position Paper for the CHI 2006 Workshop on End-User Software Engineering. Available online from http://eusesconsortium.org/weuseii/docs/ErikaOrrick.pdf

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

Wray, S. (2007). SQ Minus EQ can Predict Programming Aptitude. In Proceedings PPIG'07, *10th annual workshop of the Psychology of Programming Interest Gro*up, pp. 243-254.