Available online at www.sciencedirect.com

ScienceDirect

Information and Software Technology 49 (2007) 540–551

INFORMATION AND SOFTWARE TECHNOLOGY

www.elsevier.com/locate/infsof

Ethnographically-informed empirical studies of software practice

Hugh Robinson, Judith Segal, Helen Sharp *

Centre for Research in Computing, The Open University, Walton Hall, Milton Keynes MK7 6AA, UK

Available online 11 February 2007

Abstract

Over the past decade we have performed a sustained series of qualitative studies of software development practice, focusing on social factors. Using an ethnographically-informed approach, we have addressed four areas of software practice: software quality management systems, the emergence of object technology, professional end user development and agile development. Several issues have arisen from this experience, including the nature of research questions that such studies can address, the advantages and challenges associated with being a member of the community under study, and how to maintain rigour in data collection. In this paper, we will draw on our studies to illustrate our approach and to discuss these and other issues.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Software practice; Field studies; Qualitative analysis

1. Introduction

Qualitative studies of software practice appear to be unusual in empirical software engineering research. A recent taxonomy of the papers published in the journal *Empirical Software Engineering* over the period 1997–2003 [39] indicates that research in this field is dominated by quantitative studies that test hypotheses by means of laboratory experiments, using experimental and control groups and statistical analysis of the results. Seaman [35] makes the case for qualitative studies as being necessary complements to quantitative ones, in that qualitative knowledge is an essential prerequisite for the generation and testing of hypotheses and for interpreting the results of such tests. Whilst we acknowledge that quantitative studies are clearly important in advancing the field, the fact remains that there are circumstances when it is inappropriate to factor out the complexity that surrounds software practice. As Lethbridge et al. [25] say:

> 'Software engineering involves real people in real environments… Accordingly, to truly understand software engineering, it is imperative to study people – software practitioners as they solve real software engineering problems in real environments' [p. 311].

Although the sort of study advocated by writers such as Lethbridge et al. remains a rarity, there are a growing number of exceptions, including Button and Sharrock [7], Singer et al. [49], Beynon-Davies et al. [4] and Chong [8], as well as our own work.

Over the last decade, we have carried out a sustained series of qualitative studies of software development practice as a social activity. These studies have concentrated on four different areas: the adoption and evolution of software quality management systems [19,45], the emergence of object technology [42], the particular problems of professional end-user developers [36,37] and agile software development, particularly eXtreme Programming (XP) teams [30,31,41]. The findings from these studies have provided insight into the social dimensions that are an intrinsic part of software development, including shared values, assumptions and beliefs, and the influence of particular individuals. They have also elucidated a variety of practices that allow us to understand how software development teams operate, and hence to facilitate deeper reflection.

* Corresponding author. Tel.: +44 1908 653638; fax: +44 1908 652140.
E-mail addresses: h.m.robinson@open.ac.uk (H. Robinson), j.a.segal@open.ac.uk (J. Segal), h.c.sharp@open.ac.uk (H. Sharp).

0950-5849/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2007.02.007

This paper aims

1. To illustrate one approach to qualitative studies of software development practice, and the issues that are faced through such studies.
2. To support others who wish to conduct qualitative studies of software development practice by providing exemplars for them to follow.

We begin by providing an overview of the four areas of study and highlighting significant themes that have emerged in the planning, conducting and reporting of the studies, drawing on specific examples to illustrate our points. In Section 2, we briefly summarise four aspects of our studies: the research questions, methodology, findings and their implications. We then follow this structure in the rest of the paper, drawing out common issues across all four studies which we have recognised as significant. In Section 3, we consider the kind of research questions that are suited to this approach. In Section 4, we discuss methodological and practical challenges we have encountered. Then we explore the nature of the findings that result, and in Section 6 we discuss the responses we have experienced from two main audiences: industrial collaborators and fellow academics. Finally, we provide some concluding remarks and further references about qualitative research in this area.

1.1. Ethnographically-informed studies

Although the four areas of study listed above differ in various ways, including application domain and software technology applied, the studies are all united by focus and by a common approach. As to focus, our studies have all been of practitioners in the setting of practice, and have, in particular, focussed on the social interactions which arise as an intrinsic part of the practice of developing software.

The common approach we share can be described as 'ethnographically-informed'. Ethnography seeks to understand practice in its own terms, minimizing the impact of the researchers' own backgrounds, prejudices and assumptions. As such, researchers attend to the taken-for-granted, accepted and un-remarked aspects of practice, deliberately considering all activities as 'strange' [16]. Consequently, no feature of practice is discounted *a priori* and ethnography studies the totality of practice with all its 'messy' characteristics. We illustrate this concept of strangeness in Section 1.2, below.

All our studies were ethnographically-informed in two particulars.

1. The essential nature of practice is not known *a priori* and, importantly, cannot be assumed to be the 'official' view. Thus, the actuality of practice needs to be ascertained empirically by appropriate studies. Consequently, our research is not hypothesis driven but is driven by research questions, as we illustrate below.
2. Since our focus is on the practice of software development, as it is carried out in the real world in natural settings, we aspire, as researchers, not to perturb that practice. That is, we collect our data through a combination of methods such as observations of practice, interviews and discussions with practitioners, and study of the artefacts of practice, but do not attempt to change or unduly influence practice during the course of the study. We avoid any form of control, intrusion or experiment and so all our data were naturally occurring. Even our interview data may be viewed as naturally occurring in the broad sense of the term, since it was gathered from practitioners reflecting on their normal practice, in their place of work.

In classic ethnography, the researcher studies practice by immersing themselves in the area under study for several months if not years, documenting what takes place via a range of means that can include contemporaneous field notes, audio and video recordings of discussions and meetings, photographs and sketches of the physical layout, copies of various documents and artefacts, and records of interviews with practitioners. In some of our studies, immersion in the setting of practice was not possible because of considerations of time or of commercial confidentiality. It has become quite common to adapt ethnographic traditions in order to conduct shorter studies which fit more easily with a product development cycle, e.g. in interaction design [18,46], in software engineering [51] and in engineering design [1]. In keeping with these shortened approaches, our studies have not involved total immersion for several months. In some cases where immersion to any degree has not been possible, we have collected data by conducting semi-structured interviews, by studying practice artefacts and using serendipitous observation.

1.2. An illustration of our approach

Whilst we have appropriately adapted classic ethnography, it is worth underscoring our methodological commitment by an example from our data that illustrates attending to the injunctions of 'understanding practice in its own terms' and 'considering all activities as 'strange''. This example comes from the use of story cards in software development by XP teams [2,3]. Story cards are index cards which have written on them a very brief description of a required software task along with a small amount of other information, such as how much developer effort might be required. Story cards are often arranged on boards or walls in a way that displays what work an XP team has elected to carry out over a given period (an *iteration*, typically a matter of weeks) and how that work is progressing. One way to analytically view the use of story cards would be to focus on a conventional view of software development, where the story cards are informal statements of requirements that the XP team use to develop code. This account would be accurate, but discount key aspects of practice which can be revealed by an ethnographically-informed analysis. Such an analysis exploits the principle of 'strangeness' and views the business of the XP team as that of doing work that allows them to arrange and move cards in a purposeful fashion. Here is such an analysis, based on a recent observational study of an XP team.

> At the start of the iteration the team decided how they would arrange the cards on the wall. Sometimes they were not happy about the cards they were given and wrote a few extra ones. Developers always handled cards carefully and wrote on them very precisely. The team arranged their cards in three areas which were labelled Week 1, Week 2 and Week 3. In addition, three other areas were also labelled ('Done') and these had no cards in them.
>
> Once the cards were arranged on the wall, people took a card away to one of the machines, stuck it on the machine, and worked at that machine, usually with someone else. Cards were only taken from the Week 1 area – cards in the other two were not touched that week. Developers were very careful to record that they had taken a card from the wall, writing in a 'ghost' card just where the real card had been. After a day or so, the card would be put back, with a tick on it, and the 'ghost' card erased. The card was not put back where it came from but in a special place where cards that had been to a machine were returned to the wall. This was one of the three other areas that initially had no cards in them (the other two empty areas remained untouched that week). Gradually more and more cards were added to this one, initially empty, area.
>
> Sometimes developers wrote out a new card that was pink in colour and placed it on a separate piece of wall, labelled Bugs with other pink cards. People tended to notice this or remark on it.
>
> Early in each day, the team would meet around the wall and have a chat. Sometimes a card might be taken down – carefully – as part of the chat. A developer might (carefully) write or change something on the card before it was placed back on the wall, in the place where it had come from.
>
> At the start of the next week, the chat around the wall was quite a bit longer and the cards on the wall were rearranged so that the area where cards had been taken from in the first week was now empty.

What such an analysis brings out, which a more conventional view of software development would not, is the crucial role that the cards, the wall and the arrangement and movement of cards on the wall has for orchestrating and coordinating the work of software development. The cards/wall literally 'keep the score' in both a musical and a sporting sense. It suggests that XP teams make use of resources to manage work which are similar to those employed in patient care activity by nurses in ward settings (e.g. [12, Chapter 5]). Furthermore, the account reveals the highly disciplined nature of software development in an XP team. If story cards were regarded as just statements of requirements that are used to develop code, the disciplined nature of their use might not have been highlighted.

This perspective also illustrates what it means to provide an account that seeks to 'understand practice in its own terms', from the point of view of those being observed, i.e. the developers, rather than being constrained by any pre-ordained conceptual or theoretical framework. We shared the account above with the XP team concerned. The response was straightforward: *'Yes. The cards are in charge.'* Indeed, the team's own internal wiki has an entry entitled 'The Care and Feeding of Story-Cards'. This is not to say that the developers themselves would have written such an account if asked (indeed the wiki entry is not phrased in this way), but it is an account that draws out perspectives that the developers recognize as valid.

We discuss some of the challenges facing software engineers in taking such a perspective in Section 4.1. For clarity, we should emphasise that the account presented here is one way of analyzing a situation that illustrates the 'strangeness' principle and the perspective of 'understanding practice in its own terms'. This kind of account is not the primary form for presenting findings.

2. Our qualitative studies

In this section we briefly summarise the four areas covered by our studies. For each area, we highlight the research question(s) we aimed to address, the research methods we used, a summary of the findings, and the implications of those findings.

2.1. The adoption and evolution of software quality management systems

In this project, we explored the human-related, i.e. non-technical factors which affect the adoption and evolution of software quality management systems (SQMSs). Our particular focus was on the effectiveness of quality initiatives such as accreditation to ISO9000, and the social factors affecting their success, their adoption and evolution. Our main research question was

- What are the social factors that influence the adoption and evolution of software quality management systems?

To answer this question, we collected data from five organisations using a combination of semi-structured interviews, and ethnographic-based observation. The data collected consists of audio and video recordings, items from noticeboards, technical documentation, marketing brochures, observations of the physical environment and employees' habits. We used ethnography [16] to provide a framework for studying the culture of a setting, and for uncovering the knowledge, ideas, beliefs and values which inform activity. We used discourse analysis [28] to focus on the structures of text to explore people's interactions and interpretations, and uncover the ways in which cultural attitudes are transmitted, discussed, developed and perpetuated [43].

Through the project, we uncovered organisational and individual factors relating to our research question and also identified tensions that occur between the developers and the SQMS itself (e.g. [19,44]). The factors we identified were as follows:

- Organisational factors: the nature of the company's business; the customers, and market pressures.
- Individual factors: the charismatic leader, the quality manager, the maverick developer, and the pressures on the individual.

This work has implications for the introduction of standards documentation within software development and highlights the significant role which influential individuals (such as the maverick) have with such endeavours, and the need to be aware of the developers' attitude towards the SQMS and its introduction. More generally, this work also has resonances with any attempts to introduce new ways of working.

2.2. The emergence of object technology

In this project, we wanted to understand how object technology emerged to be a dominant paradigm, but we wanted to take a contemporaneous perspective, as opposed to *post-hoc* accounts produced once object technology had become dominant. To achieve this, we took practice to be what software developers were writing about object technology during that period. To this end, we studied materials of the period such as journal and magazine articles, trade newspapers, advertisements, and conference proceedings in order to develop a social account of the emergence of object technology [40,42]. That is, we attempted to see things through the eyes of those living through that emergence.

The research questions for this project are listed below, but it is important to note that these were investigated from the perspective of contemporaneous materials:

- How did object-oriented technology emerge?
- What were the key constituencies (individuals and groups) and the key events that shaped object-oriented technology?

Five researchers studied materials originating from the late 1970s through to the early 1990s. Our approach was to review this material and identify papers, articles, calls for papers, conference announcements or commercial product advertisements where mention was made of object-oriented technology in some way. We then analysed this selected material to understand how object-orientation was regarded at the time. We use both quantitative counts (e.g. the number of 'objectish' papers in some publication over time) in a style similar to the textual demographics of Cortada [10] and a qualitative approach using discourse analysis [28] where the discursive structures employed (how things are said) are analysed along with what is said. Importantly, our analysis of the material attends to the contemporaneous 'voice' of the author/participants, rather than imposing a view of what might be significant based on hindsight.

Using this analysis, we charted the emergence in terms of three periods: interpretative flexibility; community and dissemination; and a pervasive paradigm. We found that the emergence of object technology revolved around an informal community that was receptive to its development and acceptance, and consequently, the role of the community that built up around the technology was pivotal. One implication that this work highlights is that a social history of technology is not the same as the intellectual history of an idea. The intellectual origins of object-oriented programming lie in the 1960s with the development of the Simula language [11]. However, a technology is more than just an intellectual idea or technical insight: it involves significant use by communities of practitioners; advocacy and dissemination via fora that articulate, confirm and develop the technology; and social relations implied by the use of the technology. A consequential implication of this finding is that the role of communities within software practice advancement is key.

2.3. The particular problems of professional end-user developers

By 'professional end-user developers', we mean professional people working in highly technical domains, such as financial mathematics or research scientists, who develop their own software in order to further their own professional tasks. The overarching research question which underlies our studies in this area is:

- How might software engineers best support professional end-user developers?

In our studies [36,37], we collected data by interviews and studies of documentation relevant to practice. We addressed issues of internal validity by triangulating both our data and our methods. We interviewed people at different levels of the organisation, as well as different people at the same level, and people in different teams with different responsibilities as well as different people in the same team with the same responsibilities. We compared data from our interviews with the practices and processes laid down in documents and with the perceptions of managers of the practices and processes to which practitioners should adhere.

Our main findings were

1. That professional end-user developers regard software development as being a marginal activity and are thus not prepared to put in more effort than is strictly necessary to satisfy the immediate need.

2. That traditional staged software methodologies are inappropriate for use by professional end-user developers (especially with regard to the upfront specification of requirements).
3. That there are communication difficulties inherent in relying on project documentation that are over and above the difficulties inherent in communicating highly technical domain knowledge.

The implications of our findings are that software engineers should not attempt to support professional end-user developers by supplying them with tools, techniques or methods which involve anything but a small perturbation in their natural way of working [38]. We also suggest that some practices associated with agile methodologies might be more appropriate for this class of developers than the practices of more traditional software development methodologies.

2.4. Agile software development

Agile software development is a relatively new approach which represents a reaction to what are seen as the failings of the more traditional plan-driven approach to software development (e.g. see [5]). In our studies we have focused mostly on eXtreme Programming (XP), which is one of the more popular agile methods in the UK. There are several aspects which distinguish XP from more traditional forms of software development, and in our context it is key that XP emphasizes the social nature of software development, and champions the individual developer. In our studies we have addressed the following question:

- What is the reality of XP development i.e. how do software practitioners develop systems using XP?

This strand of our work is ongoing and to date we have conducted observational studies in five mature XP teams. These observational studies last typically a week, and involve two observers. The main observer is present for the duration of the visit, while the second observer attends for a day or so in order to understand the context of the study, and hence to be able to contribute sensibly to data analysis. Both observers are involved in the analysis of data. A follow-up visit in which the observers' findings are discussed with the participants is arranged after about six weeks. In all cases, we have found that this feedback session has provoked considerable discussion.

We have produced a variety of results from this work, including:

- The characteristics of XP teams [30].
- The effect of different organizational cultures on the practice of XP [31].
- Social aspects of XP's technical practices [32].

There are various detailed implications from our findings, e.g. suggestions of how different organizational cultures might view the adoption of XP, or of how to maximize the likelihood of an XP team being successful. More generally, our work has encouraged reflection and sharing of experiences within the practitioner community, and has provided tangible experiences and information about XP for fellow software engineering researchers.

3. Research questions

Ethnographically-informed studies of practice are not appropriate in all circumstances, and there are some kinds of research question which they cannot answer. For example, as discussed in Robinson et al. [32], ethnographically-informed studies may take various roles in a wider research programme – sometimes it is appropriate for them to play a major role, sometimes they are more suited to being used in combination with other methods and under some circumstances they can only be of limited use. So under what circumstances have we found these studies to be useful? How can you characterize the kinds of research question that this approach helps to address?

Ethnographically-informed studies do not address research questions which are based on testing hypotheses derived from theories. In fact, we try to eschew *a priori* theory in our research, treat all our data as 'strange' and try to interpret it in the particular setting of its collection. We thus ask 'how' and 'why' and 'what are the characteristics of' questions rather than 'Is X better than Y' or 'Will this technique make programmers more productive?' kinds of question. That is, we favour questions such as:

'How do software practitioners develop systems using XP?' rather than 'Is single programmer coding more productive than pair programming?'

and

'Why don't financial mathematicians adhere to a company manual of software development practice?' rather than 'Does structuring the manual in this way help financial mathematicians produce more lines of code an hour?'

and

'What are the characteristics of a technology adoption?' rather than 'How did the ideas of Simula develop into Java?'

The descriptions that emerge from qualitative studies addressing these kinds of question capture the rich interdependencies in the context of development [21] in a way that hypothesis testing does not. Hypothesis testing often focuses on questions such as: is X better than Y? But X may be better than Y in some respects and not others. For example, introducing a new software quality procedure might increase manager visibility of progress but decrease the motivation of programmers.

Understanding 'why' something happens the way it does, or 'how' a particular goal is achieved allows the participants and the observers to consider the consequences of changing current practices, or even removing them. For example, in our work on agile teams, we have considered 'what if' questions in the context of modifying the nature of story cards [45]. As Herbsleb [17] says:

> 'Understanding the underlying mechanisms in play as people tackle software engineering tasks – mechanisms rooted in human cognition, social practices, and culture – is critical to the progress of our field' [p. 23]

4. Methodological and practical challenges

There are many issues that need to be considered when setting up and running a qualitative study, many of which are covered in textbooks or others' reflections (e.g. [33,25]). For example, conducting field studies results in substantial collections of rich data, and this can be a disadvantage as well as an advantage as it is easy to be overwhelmed by the amount of data to be analysed.

Here we discuss two issues which have been particularly significant for us: the relationship between the researcher and the researched, and keeping rigour in qualitative studies. We have not seen the former discussed in detail elsewhere, and addressing the latter is an essential prerequisite to the acceptance of qualitative studies by the academic community.

4.1. The relationship between researcher and the researched

The relationship between us as researchers and our participants has been particularly significant in two respects. Firstly, we have chosen fields of study where we belong to the community being studied, i.e. we could reasonably be described as software engineers or as research scientists or as mathematicians, and this raises both challenges and advantages. Secondly, ethical issues concerning formal contracts or informed consent agreements have been problematic in our experience.

4.1.1. Researchers as members of the community

A common view of ethnographically-based research is that it is exemplified by a researcher from one culture, e.g. England, spending a period of immersion in another, quite different culture, e.g. a nomadic tribe in Africa. The researcher then tries to elucidate elements of the second culture, viewing everything as 'strange'. If the researcher herself belonged to this second culture, or was already closely involved in this culture, then nothing would be 'strange', everything would be as expected, and it would be difficult for her to discover insights. Of course, the researcher would not be entirely without expectations, probably derived from an informal theoretical framework. She could reasonably expect to find evidence of a friendship and kinship culture, of an oral language and symbol system, and so on, but as she comes from a quite different culture, regarding everything as strange is likely to be more instinctive.

Our studies differ from this view as we (the researchers) share the backgrounds and some of the traditions of our participants; as observed by Lethbridge et al. [25], software engineering researchers are often software engineers themselves. In both our object-oriented work and our agile development work, the researchers are software engineers by training, have been practitioners in the past, and have been accepted as members of the respective communities. In the case of the professional end-users, the researcher can reasonably be described both as a mathematician and a research scientist. The team involved in the software quality studies included a mixture of backgrounds; two researchers were software engineers with some experience of software quality systems while one researcher was not a trained software engineer but a social scientist.

Being a member of the community under study has both challenges and advantages for the researcher. The challenges arise mainly from the tension involved in moving between two worlds – understanding what is going on at one level, yet adopting the perspective of viewing everything as strange at the other level. The former understanding is necessary in order to be accepted by the participants, to ask pertinent questions, recognise what may be relevant and what not, and to be able to follow activity sufficiently so as not to be overwhelmed, while the latter adoption is needed in order to gain an alternative perspective which can trigger valuable insights. For example, the account given in the introduction about the story cards was only possible by adopting this alternative perspective.

Another facet of this tension arises from the need to be non-judgmental. When a field is familiar, it can be difficult to avoid offering judgmental opinions, yet this is necessary in order to avoid bias in collecting and interpreting data. For example, one of the software engineers involved in the software quality studies was surprised to find that the lead developer kept a print-out of the entire system's code in his filing cabinet drawer. He regarded this as completely unnecessary and indicative that the team had an outdated view of software engineering. He then found it difficult to disregard this impression and not to comment on it when observing other team activities.

In the object technology study our experience was slightly different. Here, our 'participants' were not people but contemporaneous artefacts. However some of the artefacts represented events or movements that we had lived through ourselves, e.g. notices for a workshop or conference that one or other of us had attended, journal papers describing the relationship between expert systems and object-based systems which one or other of us had read with a different purpose many years ago. In this situation, although the relationships between researcher and researched were quite different, the tensions around familiarity were still present. To help maintain our perspective of 'strangeness' and reduce the possibility of introducing bias, we encouraged each other to discuss our experiences and to put these observations in the context of the current study. We found that this could diffuse feelings of tension and help to re-focus analysis on the materials rather than our memories.

The main advantages of being a member of the same culture as the researched revolve around acceptance and common vocabulary. At a practical level, acceptance as a fellow mathematician, software engineer or whatever evokes a feeling of trust and fellowship which we have found makes it easier to negotiate access to the settings of practice. Once admitted to these settings, if the researcher is recognised as 'one of the tribe' then the participants can relax and focus on working in their natural way without feeling the need to treat the researcher as a guest. Similarly, if the researcher is comfortable with the same vocabulary as the participants then conversations and interviews are conducted more easily.

4.1.2. Formal versus informal

Ethical considerations are paramount when dealing with human participants (for a more detailed discussion of ethical issues in this context, see [48]). Here we focus on one particular area that has exercised us over the years: informed consent.

Throughout our studies we have been very successful in recruiting and maintaining good relations with our collaborators. We believe that one reason for this is that we have been guided by them as to the level of formality in the relationship, while keeping our approach and intentions informal. This, in part, stems from the relationship discussed above: we are observers who are already, in a strong sense, part of the 'tribe'. In practice, this means that we do not ourselves ask individuals to sign any formal agreement documents. We feel that this would cause more problems than it would solve. Signing a formal document would immediately change our relationship and prompt the individual to be suspicious: "why am I being asked to sign this agreement? What is it about the research which makes it necessary for there to be a formal agreement?" One of our principles is that practice should be disturbed as little as possible, and introducing unnecessary formality would undermine that principle. If the collaborators themselves (at an organisational or individual level) ask for this level of formality, then we, of course, agree, and have been party to non-disclosure agreements in some cases.

In conducting our research we are bound by several codes of ethics. One set of ethics we have found to be particularly relevant to this type of research is that of the British Sociological Association. They include several points regarding the relationship between the researcher and the researched. These points cover the need to inform the individuals about the nature and purpose of the research, their rights within the research activity relating to the recording and use of data, their ability to withdraw from the research whenever they wish, and the plans for confidentiality. The code also stresses the need to obtain "freely given informed consent" from individuals. Often this is obtained through a written document, but we have not used this approach with individuals for the reasons given above. As alternatives, we have used verbal consent captured on audio recordings, email exchanges and the organisation's own dissemination mechanisms. For example, when conducting interviews for the software quality study, we audio-recorded them and obtained oral consent from the interviewees at the beginning of the session. In the case of one of our agile teams, we had approached members of the organisation informally and they themselves sought opinions and agreement from their colleagues through their wiki site. Responses to this were recorded through the wiki, and we therefore could be assured that consent had been given.

Whatever recording mechanism is involved, we begin a study by explaining to all involved who we are, what we are doing and what are the rights of the participants. We have not yet encountered a situation where individuals have asked to withdraw from a study once it has started. However we have had access denied to various settings. For example, one potential collaborator in the agile development studies was concerned about the status of the team within the company. When we initially met with our main contact, he took us away from the company building to meet and discuss our research, and although we were able to briefly visit the team and see their environment, we were not allowed to conduct a study there.

The need to gain informed consent from participants is clear, but we have found that an informal approach is often more appropriate for this kind of study. We have considered asking for written informed consent, but it has been clear that, with our participants at least, asking individuals for written consent would alter the relationship significantly making natural observation impossible. We have found that, whatever decision is taken on this, it is important to consider the situation and to act responsibly under the current circumstances.

Above all, we take our responsibility seriously. According to the British Sociological Association's code of ethics, this responsibility covers:

> 'both to safeguard the proper interests of those involved in or affected by their work, and to report their findings accurately and truthfully…to ensure that the physical, social and psychological well-being of research participants is not adversely affected by the research'

4.2. Keeping rigour in qualitative work

Being rigorous and avoiding bias is important, yet it is all too easy to introduce it in both qualitative and quantitative studies (see, for example, Kitchenham et al. [21,22], Huff [20]). A common view is that there is a bias inherent in qualitative work which isn't present in quantitative, but we refute this view, and agree with the following quote from Lanubile [24]:

> 'The objectivity of empirical research comes from a review process that assures that the analysis relies on all the relevant evidence and takes into account all the rival

interpretations. This can be done (or not done) in both a quantitative and a qualitative analysis' [p. 102]

Taking a rigorous approach to data collection and analysis is essential in order to avoid bias. In our studies, this takes the form of three main principles: triangulation, seeking disconfirmation and iterative development of understanding. The last two of these are also found in the grounded theory approach to data analysis developed by Glaser and Strauss [13]. Although we do not follow their approach in detail, we recognise that there are similarities.

Triangulation is a well-known approach (e.g. [33,46]). In our studies we employ triangulation by using data from many different sources: for example, from interviews with different people having similar or different roles; from the artefacts of practice; from observations of different roles; from informal conversations; from contemporaneous articles of different types and from different sources, and so on. Because of its detailed treatment elsewhere, and the fact that we have no particular issues with this approach, we do not discuss this further here.

One aspect of the rigorous approach we take is that all our findings are grounded in the data, and we analyse our data to check that none of it refutes any of the conclusions we want to draw. For example, in one of our XP teams, we were struck by the nature of team leadership. We had observed that a recent recruit had chaired the company stand-up at the start of the iteration. Other, more senior members of staff accepted this, did not interrupt or attempt to guide him in this role, but took his lead. From this observation we wanted to explore the conclusion that the organisation had a very flat structure. So we looked for disconfirming instances where more junior members of staff were not allowed to take such roles, or were discouraged from making these kinds of contribution. In fact, we found several confirming instances. In team stand-ups, team leaders actively encouraged participation in the planning and estimating tasks that needed to be undertaken. Note that we focus here on behavioural confirmation, not just what people say but what they actually do. In this way, we recognise that many social or human characteristics are dispositional – e.g. evidence for the fact that A respects B is not so simple as A utters 'I respect you, B' but that A's behaviour and actions evince respect for B.

Another aspect contributing to rigour and which can mitigate against inappropriate assumptions or interpretations is the use of feedback. We check regularly that interpretations resonate with collaborators and that any particular understanding we are forming is accurate. In our studies we use both formal and informal feedback. Informal feedback happens on an ad hoc basis and helps to ensure that we have not misunderstood. As we attempt to construct an understanding of the situation, we present tentative findings to our participants. This allows them to say 'I may have said that but what I meant was …' or 'You should talk to X – he knows more about Y than I do' or other suggestions as to how the interpretation of the situation could be improved or extended. An example in which feedback corrected an inappropriate assumption constructed on the basis of the researcher's knowledge of software engineering is as follows. In a study of one agile team, the researcher was observing a pairing session. This pair were coding and running tests for about an hour; they kept getting 'red bars' (showing that the code repeatedly failed the set of automatic tests). Neither of the programmers seemed to be at all frustrated or upset by this. One possible interpretation is that these two programmers are not at all bothered by failure (or maybe that they do not care about their work). However, when the researcher asked one of the pair about their reaction to failure, he was told that he had misunderstood what was happening. In fact, the pair were just starting out and the first thing to do was to construct failing tests. Hence, the repeated 'red bars' were a sign of success, not failure.

This iterative construction of interpretation underpins hermeneutic investigations, (see, e.g. [23]). But it would not have been possible had not the researcher and researched been able to communicate in the same language and appreciate each other's concerns (see section above about the relationship between researcher and researched).

More formal feedback usually takes the form of a meeting, often over an informal lunch, during which we discuss our findings with the participants both to offer them some thoughts for their own reflection, and to check our observations with the participants to see if we have misunderstood. We discuss the response to our findings in more depth in Section 6.

5. The nature of our research findings

One major driver for empirical studies in software engineering is the aspiration to improve practice. In reflecting on our experience of conducting qualitative studies, we have found it useful to refer to another field of research with the same driver: education. Specifically, we found Hammersley's discussion of the use of qualitative research in education [15] to be particularly thought provoking. He describes the findings of qualitative researchers thus:

> 'they [qualitative researchers] have stressed the diverse orientations of people involved in social activities; the way in which people actively make sense of their surroundings, and how this shapes what they do; the unintended and often unforeseen consequences of actions; and the resulting contingency of most courses of events' [p. 394]

We have found similar themes emerging from our data, and have used them to

- uncover and elucidate problems and thereby delineate their solution spaces;
- challenge received views; and
- provide rich narrative accounts of practice.

We shall expand on each of these in turn.

5.1. Problems and solutions

Because we ask open-ended questions, the kind of answers we obtain are not knowable in advance. Instead, the research question acts as a focus for our investigations. There is no expectation that our answers provide direct solutions to specific problems. Rather, they may provide a context in which problems can be elucidated and the solution space delimited. For example, in our studies of professional end-user programmers, we found that one of their problems was in the sharing of software development knowledge. This problem arose in a culture in which it is assumed that 'Anyone can develop software', that is, in a culture where software development knowledge is perceived as being both trivial in itself and trivial to acquire. Our studies revealed that potential solutions to this problem were to be found in supporting informal communication and knowledge sharing, rather than in any formal document or computer-based tool [36,37]. We did not set out with the idea of studying this problem, nor of identifying ways in which to share knowledge; the problem and potential solutions arose from the data. Similarly, in our work on software quality, we did not expect to find a particular problem around the role of a maverick individual, but only discovered that this was an issue while collecting our data. In this case, we then developed potential solutions to the problem in consultation with one of our collaborators, and informed by our studies [19].

In some cases, we have uncovered problems, but investigation of the solutions lies outside our immediate study. For example, in our XP studies we have received comments from our collaborators that XP development can become boring after a while, and that pairing can undermine an individual's self-confidence. Although we have heard this anecdotally from more than one source, we have not yet pursued them to ascertain the kinds of solution that might be suitable, nor to find out how common these reactions are. This is an example of where ethnographically-informed studies may help to formulate research questions for the next study [32].

5.2. Challenging received views

Knowing the actuality of software development practice, rather than the received view, is an essential prerequisite to the design of a tool, technique, or methodology to support that practice.

The following two illustrations from our data, in which the actuality contrasted with the received wisdom, concern the emergence of object technology and the role of individuals in pair programming.

Based on our empirical data, object technology was not perceived as a significant development during the early 1980s. For example, in the 6½ years from June 1978 to December 1985, only nine papers (out of some 400) in *ACM SIGPLAN Notices* mentioned object technology in some substantive way. During this time, object technology is typically discussed in terms of its significance for some existing area of computing (such as programming environments) rather than as a significant development in its own right. Our evidence locates the constitution of a separate community that sought to discuss, promote and disseminate object issues via its own *fora* (such as the first OOPSLA conference in 1986) as being a crucial turn. This account of emergence based around the actuality of practice (what happened contemporaneously) stands in contrast to the 'official view' (what is now said to have happened) where the emergence of object technology is seen to be the straightforward and seamless adoption of manifestly significant developments in the SIMULA 67 language of the late 1960s (for example, see the celebration publication entitled *Happy 25th Anniversary Objects!* [47] published by SIGS Publications and sponsored by Borland).

The 'official' view of pair programming is one of each pair having distinct roles, often characterised as the *driver* (operating the keyboard and mouse) and *navigator* or *copilot*. Indeed, Kent Beck's original account [2] gives this 'official view': *'One partner, the one with the keyboard and the mouse, is thinking about the best way to implement this method right here. The other partner is thinking more strategically.'* We have observed pair programming in practice with five teams and have never found that pair programming conforms to this 'official' view. Rather, both partners in the pair engage together in activities such as thinking more strategically (or in implementing a method), as well as in a range of other activities such as reasoning about code behaviour or assessing the potential for refactoring. It is not uncommon for one of the pair to operate the keyboard and the other the mouse. Interestingly, Beck's latest account of pair programming [3] makes no mention of the distinct roles.

Challenging received views can inform and improve practice. For example, our work has also indicated that the computer itself is a key player in pair programming. An XP expert practitioner and accomplished coach has stated:

> "⟨This⟩ ethnographical work in eXtreme Programming is proving invaluable, … I am particularly interested in the most recent work, suggesting that the computer itself is a participant in XP's pairing practice… I intend to explore the idea at greater length, and to use ⟨this⟩ work directly in my forthcoming book on transitioning to the XP stance."

5.3. Providing rich accounts of practice

Our qualitative studies have been very good at providing rich accounts of practice. Hammersley [16] comments that 'Much of the knowledge on the basis of which practitioners work is tacit' (p. 397). In addition, he highlights the importance of these accounts in making what is implicit, explicit, and hence in making the nature of practice transparent and available for inspection, reflection, comment and critique. Such accounts thus provide practitioners with a mirror of their practice which might support their activities as ‘reflective practitioners’ [34], and provide researchers with rich accounts of the actuality of practice.

For example, our XP studies have shown the importance of understanding the various dimensions, both ‘technical’ and ‘social’, of software development practices such as pair programming. So, by way of illustration, we found evidence that pairing is intense and stressful for developers [32] and that this characteristic of pairing affected the way in which pairing was constituted (for example, in one case, not integrating code after 5 p.m.). Similarly, we found that the particular way in which the on-site customer role was fulfilled (the individual, their expertise and status, their responsibilities, etc.) in a team had consequences for the way in which development, including the involvement of the on-site customer, was progressed by that team [31]. In both of these cases, when we fed back our findings to our collaborators, these points sparked a lengthy and energetic discussion. It seemed to give our collaborators the chance to talk about issues that resonated with their own experience, despite the fact that the teams held regular retrospectives. This implies that either these issues had not been recognised by the collaborators themselves until we prompted them, or that discussion of the topics had been curtailed.

Rich accounts help to prompt resonances with our collaborators, but they also allow us to unearth ‘the many small actions’ described below which help to make a methodology successful:

> ‘Publishing a methodology does not convey the visceral understanding that forms tacit knowledge. It does not convey the life of the methodology which resides in *the many small actions* that accompany teamwork. People need to see or personally enact the methodology’ [9, p. 139, our emphasis].

It is precisely these ‘many small actions’ which we describe in the rich narrative accounts which result from our studies.

6. The impact of our research

We discuss the impact our research has had in terms of the responses we have received from two different audiences: our direct collaborators and fellow academic researchers.

6.1. Responses from our direct collaborators

Responses from collaborators have been varied, but generally positive. For example, feedback sessions in our agile studies have been greeted with warmth and satisfaction. Often our findings resonate with individuals’ own experiences, and these sessions lead to further discussions about a topic. This both enhances our understanding and promotes debate within the collaborator team. For example, issues that sparked further discussion included sustainability in terms of the intensity of pairing (which one team compared with marriage), use of space within the office, the role of the customer, and software usability. The financial mathematicians did not ask us to provide an account of their activities, and did not expect one, but when presented with our findings, they were pleased to receive a narrative which made the implicit explicit so that actions could be inspected.

On the other hand, one of our collaborators in the software quality studies assumed that we would be giving more judgmental feedback. In particular, they wanted to know how they compared with other organizations, or against the industry norm. They recognized that our ‘anecdotal’ evidence was fairly accurate, but were unhappy that we did not provide details of how many people expressed a certain view, as this meant that “It is quite difficult to decide which comments we should attach importance to”.[1] They also asked us directly for advice. Overall, we felt that they had viewed us as consultants rather than impartial researchers, although in conclusion they said that they did not feel misled.

Feeding back to our collaborators is an important part of the process we engage in, and we encourage them to reflect on our observations and to make their own conclusions about how (if at all) they should respond to any issues we identify. We have therefore found that it is important to make it clear to collaborators what to expect, both in terms of day-to-day activities and in terms of feedback.

In one team, we were told that just being there helped to support reflection in the team. Of course, it is not good news to us that we are having any kind of effect on the developers, but some kind of effect is inevitable. This was reported in a positive light rather than as a disadvantage.

In addition to responses from our direct collaborators, we have presented work at practitioner conferences and to other (non-collaborator) development teams, and many of our findings have prompted recognition, e.g. the rhythm of XP development, the characteristics of successful XP teams, and the significant role that a maverick may play in development.

6.2. Responses from academic researchers

Judging from the much higher percentage of published papers in software engineering which report quantitative, hypothesis testing, studies than those which report qualitative studies see [14,39], it seems that software engineering academics feel more comfortable with the former than the latter. Quotes such as the following, indicating the perceived supremacy of the former type of study, are not difficult to find in the literature.

> ‘Can we get by with forms of validation that are *weaker* than experimentation?’

[1] These are comments from a written reply they sent in response to a written report, which they had requested.

'When *control is impossible*, researchers will use case studies, observational studies and other investigative techniques' ([50]: our emphasis).

It is also quite easy to find quotes equating empirical studies with experiments – and hence totally discounting qualitative studies – such as the one below.

'Our goal of this workshop was… to make another step towards the vision of establishing systematic analysis of *experimental studies* as part of the foundations of software engineering' [6].

On the face of it, this quote seems uncontroversial, but the workshop to which it refers was called 'Realising *evidence-based* software engineering': experiments are not the only form of empirical studies, and experimental results not the only form of evidence.

We have come to realise that if you carry out qualitative research in a controversial area then one response is not simply to dismiss your findings, but to dismiss the area in its entirety. For example, on several occasions in seminars and talks, we have found it impossible to describe our work with XP teams in the face of a barrage of questions along the lines of 'but pair programming doesn't work' or 'The Planning Game is too inefficient'.

7. Concluding remarks

We have described our ethnographically-informed studies of various aspects of software development practice, and discussed several of the issues that have arisen from this experience. There are, of course, other qualitative approaches that are appropriate to different contexts and different research questions. Further information about these other approaches is readily available. For example Robson [33], Mason [26], and Oates [27] are accessible resources for those who are interested in conducting qualitative studies.

References

[1] LJ. Ball, TC. Ormerod, Applying ethnography in the analysis and support of expertise in engineering design, Design Studies 21 (4) (2000) 403–421.

[2] K. Beck, eXtreme Programming Explained: Embrace Change, Addison-Wesley, San Francisco, 2000.

[3] K. Beck, C. Andres, eXtreme Programming Explained: Embrace Change, Addison-Wesley, San Francisco, 2005.

[4] P. Beynon-Davies, D. Tudhope, H. Mackay, Information systems prototyping in practice, Journal of Information Technology 14 (1999) 107–120.

[5] B. Boehm, R. Turner, Balancing Agility and Discipline, Addison-Wesley, San Francisco, 2004.

[6] D. Budgen, B. Kitchenham, Realising evidence-based software engineering: a report from the workshop held at ICSE 2005, in: Proceedings of ICSE-2005. Available from: http:/portal.acm.org/dl.cfm.

[7] G. Button, W. Sharrock, Project work: the organisations of collaborative design and development in software engineering, CSCW 5 (4) (1996) 369–386.

[8] J. Chong, Social behaviors on XP and non-XP teams: A comparative study, in: Proceedings of Agile 2005, IEEE.

[9] A. Cockburn, Agile Software Development, Addison Wesley, San Francisco, 2002.

[10] J.W. Cortada, Using textual demographics to understand computer use: 1950–1990, IEEE Annals of the History of Computing 23 (2001) 34–56.

[11] O.-J. Dahl, K. Nygaard, Simula – an Algol-based simulation language, Communications of the ACM 9 (1966) 671–678.

[12] H. Davis, The social management of computing artefacts in nursing work: an ethnographic account, Ph.D. thesis, University of Sheffield, 2001.

[13] B.G. Glaser, A. Strauss, Discovery of Grounded Theory, Aldine, Chicago, 1967.

[14] R.L. Glass, I. Vessey, V. Ramesh, Research in software engineering: an analysis of the literature, Information and Software Technology 44 (2002) 491–506.

[15] M. Hammersley, The relevance of qualitative research, Oxford Review of Education 26 (3/4) (2000) 393–405.

[16] M. Hammersley, P. Atkinson, Ethnography, Principles in Practice, Tavistock, London, 1983.

[17] J.D. Herbsleb, Beyond computer science, in: Proceedings of the International Conference in Software Engineering, ICSE'05, 2005, pp. 23–27.

[18] K. Holtzblatt, J.B. Wendell, S. Woods, Rapid Contextual Design, Morgan-Kaufmann, San Francisco, 2005.

[19] F.M. Hovenden, S.D. Walker, H.C. Sharp, M. Woodman, Building quality into scientific software, The Software Quality Journal 5 (1996) 25–32.

[20] D. Huff, How to Lie with Statistics, Penguin Books, London, 1991.

[21] B.A. Kitchenham, S.L. Pfleeger, et al., Preliminary Guidelines for Empirical Research in Software Engineering, IEEE Transactions on Software Engineering 28 (8) (2002) 721–734.

[22] B. Kitchenham, S. Linkman, J. Fry, Experimenter induced distortions in empirical software engineering, in: A. Jedlitscha, M. Ciolkowski (eds.), Proceedings of the 2nd workshop in the Workshop Series on Empirical Studies in Empirical Software Engineering, 2003, pp. 7–15.

[23] H.K. Klein, M.D. Myers, A set of principles for conducting and evaluating interpretive field studies in information systems, MIS Quarterly 23 (1) (1999) 67–93.

[24] F. Lanubile, Empirical evaluation of software maintenance technologies, Empirical Software Engineering 2 (1997) 97–108.

[25] T. Lethbridge, S.E. Sim, J. Singer, Studying Software Engineers: data collection techniques for software field studies, Empirical Software Engineering 10 (2005) 311–341.

[26] J. Mason, Qualitative Researching, second ed., Sage, London, 2002.

[27] B.J. Oates, Researching Information Systems and Computing, Sage Publications, London, 2006.

[28] J. Potter, M. Wetherell, Discourse and Social Psychology, Sage, London, 1987.

[29] H. Robinson, H. Sharp, The characteristics of XP teams, in: Proceedings of XP2004 Germany, June 2004, pp. 139–147.

[30] H.M. Robinson, H. Sharp, Organisational culture and XP: three case studies, in: Proceedings of Agile 2005, July, ppp. 49–58, IEEE Computer Society Press, Denver, CO, 2005, pp. 5–28.

[31] H.M. Robinson, H. Sharp, The social side of technical practices, in: Proceedings of the Sixth International Conference on Extreme Programming and Agile Processes in Software Engineering (XP2005), 20–22 June, Springer Verlag, Sheffield, 2005.

[32] H. Robinson, J. Segal, H. Sharp, The case for empirical studies of the practice of software development, in: A. Jedlitscha, M. Ciolkowski (eds.), Proceedings of the 2nd workshop in the Workshop Series on Empirical Studies in Empirical Software Engineering, 2003, pp. 99–108.

[33] C. Robson, Real World Research, Blackwell Publishing, Oxford, 2002.

[34] D.A. Schön, The Reflective Practitioner, Temple Smith, London, 1983.

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

[36] J. Segal, Organisational learning and software process improvement: A case study, in: K-D. Althoff, R.L. Feldmann, W. Muller (Eds.), Advances in Learning Software Organizations, Lecture Notes in Computer Science, vol. 2176, Springer, Berlin, 2001, pp. 68–82.
[37] J. Segal, When software engineers met research scientists: a case study, Empirical Software Engineering Journal 10 (2005) 517–536.
[38] J. Segal, Two principles of end-user software engineering research, in: 1st Workshop on End User Software Engineering, WEUSE, International Conference of Software Engineering, St. Louis Missouri, May 2005.
[39] J. Segal, A. Grinyer, H. Sharp, The type of evidence produced by empirical software engineers, in: Proceedings of the Workshop on Realising Evidence-Based Software Engineering, ICSE-2005. Available from: http:/portal.acm.org/dl.cfm.
[40] H. Sharp, H.M. Robinson, Object Technology: Community and Culture, in: OOPSLA 02 Companion, ACM Press, 2002, pp. 92–93.
[41] H. Sharp, H.M. Robinson, An ethnographic study of XP practice, Empirical Software Engineering 9 (2004) 353–375.
[42] H. Sharp, H.M. Robinson, M. Woodman, Software engineering: community and culture, IEEE Software 17 (2000) 40–47.
[43] H. Sharp, M. Woodman, F. Hovenden, Tensions around the adoption and evolution of software quality management systems: a discourse analytic approach, International Journal of Human-Computer Studies 61 (2004) 219–236.
[44] H. Sharp, M. Woodman, F. Hovenden, Using metaphor to analyse qualitative data: vulcans and humans in software development, Empirical Software Engineering 10 (2005) 343–365.
[45] H. Sharp, H. Robinson, J. Segal, D. Furniss, The role of story cards and the Wall in XP teams: a distributed cognition perspective, in: Proceedings of Agile2006, IEEE Computer Society Press, 2006, pp. 65–75.
[46] H. Sharp, Y. Rogers, J. Preece, Interaction Design, second ed., Wiley, New York, 2007.
[47] SIGS, Happy 25th Anniversary Objects!, SIGS Publications, 1992.
[48] J. Singer, N. Vinson, Ethical issues in empirical studies of software engineering, IEEE Transactions on Software Engineering 28 (2002) 1171–1180.
[49] J. Singer, T. Lethbridge, N. Vinson, N. Anquetil, An examination of software engineering work practices, in: Centre for Advanced Studies Conference (CASCON), Toronto, Ont., 1997, pp. 1–15.
[50] W. Tichy, Should computer scientists experiment more? IEEE Computer (1998) 32–48.
[51] S. Viller, I. Sommerville, Coherence: an approach to representing ethnographic analyses in systems design, Human–Computer Interaction (1999) 14.