Dwelling in Software: Aspects of the felt-life of engineers in large software projects

Harper, R., Bird*, C. & Zimmermann*, T. & Murphy, B.
Microsoft Research Cambridge and *Microsoft Research Redmond
r.harper@; bmurphy@;christian.bird@; tzimmer@microsoft.com

Abstract. The organizational and social aspects of software engineering (SE) are now increasingly well investigated. This paper proposes that there are a number of approaches taken in research that can be distinguished not by their method or topic but by the different views they construct of the human agent acting in SE. These views have implications for the pragmatic outcome of the research, such as whether systems design suggestions are made, proposals for the development of practical reasoning tools or the effect of Social Network Systems on engineer's sociability. This paper suggests that these studies tend to underemphasize the felt-life of engineers, a felt-life that is profoundly emotional though played out in reference to ideas of moral propriety and ethics. This paper will present a study of this felt-life, suggesting it consists of a form of digital dwelling. The perspective this view affords is contrasted with process and 'scientific' approaches to the human agent in SE, and with the more humanistic studies of SE reasoning common in CSCW.

Keywords

Software Engineering, CSCW, Dwelling.

Setting the scene

A recent paper in *Communications of the ACM* asks whether changes in software engineering (SE) represented under the moniker 'agile computing' are as applicable today as they were in the middle of the 1990s. The changes exemplified by agile computing – and various other approaches of that time (such as 'Xtreme programming', and sometimes by the more prosaic sounding 'End User Programming') - all turned around the realization that SE required more flexible processes to requirements capture and coding (Williams, 2012). Adherence to a plan came to be seen as something that should always be subordinate to the development of a product that worked and appealed even if this violated aspects of the plan. Bitter and expensive failures in the SE industry up to that time had made it clear that the right products could only be devised through

constant iteration of the design and associated software engineering; this meant that plans had to be looser and made flexible, and this in turn meant that coding itself had to be more dynamic in tempo, more 'agile' as the saying had it. Though the ACM article focuses on Agile SE, it notes that the basic need to be more flexible in design and development has become more or less the norm, certainly in the engineering of consumer products, even to some extent in open-source activities. Another recent paper, this time in the *IEEE Transactions on S.E.*, de Souza et al's *The Awareness Network* (2011), examines three different contexts of software development and finds that whatever the moniker given to the engineering process, coordination and change is the fundamental contradiction whose 'solution' needs to be ensured to deliver the product.

If one accepts this, and it seems reasonable to do so, then what these authors are arguing is that doing SE requires balancing of the relationship between plans, the ways that engineers orient to and used these plans, and the coding itself. Coding has to be done in a fashion that allows revision, and sometimes concurrency of revision in different places in the overall code base. This has to be achieved while the code remains of sufficiently good quality to be (easily) 'reconciled with' and 'fitted into' what comes to be the 'emerging plan' or 'evolving spec'. The relationship between tools of coordination (like plans) and actual instances of action (such as writing a line of code) are then complex, fraught with difficulties of comprehension and overview (De Souza, 2011, Ronkko et al, 2005). Processes need to ensure that engineers 'program to plan' but at the same time can alter their coding when a new plan comes into play, whenever that might happen, without the quality of coding diminishing – though there is always a cost of some form in terms of delay or even in the quality of the code - leaving aside the question of what measures can be applied to such notions (see Nagappan, et al, 2008).

All SE involves such problems. Very large scale SE development programs (sometimes involving hundreds, even thousands of individuals) have these problems in even larger scale. Coding is typically undertaken in 'branches' or in discrete units. Changing code is ring fenced and only reinserted into the 'code base' once complete. Such branching creates costs however: coders in one branch can lose sight of what is happening elsewhere and so even though their coding might fit the requirements of their 'own' branch, the overall goal of the application (or product) itself might be undermined. Considerable effort has been put into researching where these instances occur as well as to suggest ways of alleviating them (Philips et al, 2011). Key to this has been the development of coordination tools of particular kinds. Amongst the most important of these are Software Configuration Management (SCM) tools which allow communications and documentation of coding practices in different branches in an overall engineering program. These documentation practices can take various forms, some of which can be overly burdensome. Annotation of changes is obviously

important, but this takes many forms and is often affected by the attitudes of those doing the annotation (Storey et al, 2008). Besides, annotation tools are more than just methods of documenting; they are also articulation devices, and so need especial care in design and use (Storey et al, 2008; de Souza et al 2011). Just how to support 'mutual awareness' and articulation work through such tools is now an increasingly fertile area, suggestive of new theoretical constructs as well as empirical findings (de Souza et al 2008). The emergence of Social Networking Systems (SNS), a technology that is essentially about sociability rather than the division of labour, has been of particular interest recently (Storey, 2012).

One could go on; this is just to sketch of the scene of contemporary software engineering research and the particular features of research in large-scale SE. The research agendas in question are in many ways straightforward and clear. What we would like to suggest, however, is that answers to the question of balance one finds in the literature are often coloured by assumptions made by researchers. The assumptions we have in mind are not to do with choices between method; say between quantitative or qualitative, between statistical technique, interviewing protocol or sample size, or between ethnographic and other data collection modalities. They often have to do with how the human actor in the software engineering setting (the coder if you like), the relationship between the structures that constrain and guide this individual, and the form of the actor's actions (their reasoning or predicted behavior, say) are construed from the outset. This in turn drives or determines which analytical perspectives are chosen that deliver 'data' of a particular kind; these choices too are made at the outset before data is gathered. When these are put altogether (these assumptions, tools of analysis and forms of data collection), a certain picture of the individual at the heart of this view is thereby constructed.

What we are thinking of here is Foucault's insight that institutionally organised research concerns always emphasise particular aspects of human agency and exclude others (see in particular his 1966 book, *The Order of Things*). Different disciplinary perspectives don't simply offer diverse views on the same animal, like the proverbial blind men grasping the elephant. In Foucault's view, disciplines create the creatures they investigate. Taken to extreme, Foucault's view can be thought of as offering the most ardent relativism. We certainly want to avoid some of the exaggerated claims associated with his view, but we do want to preserve some of the merits that it affords. We are thinking of Rorty's 1979 interpretation of Foucault in *The Mirror of Nature*. Rorty argues that one ought to judge the value of any particular rendering of human agency by the practical implications that the view generates. In this way one can also assess any claims it makes about empirical validity. Rorty argues this is particularly important when different methods and assumptions are used, implying a kind of internal interdisciplinarity within a domain; it is also important when diverse approaches are being deployed to address apparently similar topics.

We have both these manifest in the SE research: certainly the list above viz-à-viz SMC tools, branching structures and coding practices, annotation and articulation, and so on, would seem to suggest so. In each instance – or set of instances - certain type of creature, an instantiation of the species named a software engineer, is cast. As a result, there is more than one type of animal described within the SE literature: the species generalis, *homo softwarus*, in other words, is made up of many kinds.

And yet given this, it also seems to us that the creatures presented in this literature are notice-able for the lack of attention they give to the felt-experience of software engineers - to describing and exploring the form of life constitutive of what Ingold calls 'dwelling' (see his *Being Alive*, 2011). Certainly there seems little interest in this felt-life and more shown in topics that seem too vague to be tractable, like 'culture', or, as in a recent case, to ideas that pertain to psychology rather than consciousness and experience. One thinks of the idea that cognitive abilities of coders are being affected by social networks, drawing them out of their somewhat private forms of reason to more public ones (Storey, Treude, et al, 2010). These perspectives do not look at how engineers enact their selves through the mixtures of sense, feeling and compression that we noted in our research with SE; that resonate with the 'praxis of living', as Ingold puts it.

Our claim is not that our findings are any more correct than any other, but that we focus on aspects of 'being' that other approaches need to neglect by dint of their analytic assumptions. We approached our software engineers with a view for 'dwelling' not for, say, situated 'reason' or 'objective fact'. An important corollary of our argument is that other approaches are likely to have their own merits, focusing as they do on different concerns and thus producing a different sense of the software engineer and their practices. Having a sensibility for all these views is likely to be hard, we believe, but ought to be sought, so that appropriate evaluations of different ways of constructing the animal at the heart of SE can be done. One set of criteria has to do with empirical merit, another set should be in terms of what each view leads one to do.

It is our purpose in this paper to justify these claims: specifically that, (a), there are different species of software engineer in the literature; (b) to show one that we do not think has been looked at greatly before, namely some aspects of dwelling in software engineering ; (c) and to make some suggestive remarks about how this and other views might be judged by their utility and empirical merits.

Overview of the evidence

With this in mind we present, in the body of the paper, two views already common in the literature. We then present a differing perspective. The two extant views are, respectively, from what we will call, for the purposes of exposition, the objective science of SE and, second, the view from the humanistic perspective,

one quite often taken in CSCW. As we explain and analyze each view, and to some extent we do this by showing each view in extremis, so we will also explain the benefits that each affords. Both views we present here lead to very practical suggestions, we shall show, some for design, others for learning and training, but both offering practical benefits for the subjects rendered in the analysis itself. We shall then explore our own set of data and convey what 'dwelling in software engineering' might look like (or be experienced as). Having completed our exposition we shall offer some remarks in conclusion that point towards how this sensibility for a Foucaultian understanding of the output of SE research might be leveraged in the future, as well as some comments on 'inter' and 'multi-disciplinarity' both within CSCW and other domains that treat SE as a topic.

View 1: The animal in the 'scientific vision' of SE

The first approach we want to talk about is one that could be illustrated with more papers than any other, we think, for no other reason than because most of the software engineering research we are familiar with is produced under its auspices. We don't want to get into the process of quantifying this view since it adds nothing to our argument, nor are we saying that other views are less important, or even whether there is a shift towards other views (for discussion see Somerville 2007). The studies we have in mind tend to have an engineering science background and thus emphasise the processual, the quantitative, the 'objective' over the 'subjective'. It is with these concerns in mind that research from this view address the processual efficacy and use of SCM tools, the quantitative efficiency of different branch structures, and the relationship between these structures and pre-existing organizational forms.

A paper that illustrates this view well is Bird & Zimmerman (2012; see also Bird, Nagaappan, et al, 2009; 2012). This reports on the values that different braches offer in a large scale SE program, namely Windows. Its premise is that some branches will be more useful than others. Data for this study was collected through a mixed method, though garnering data through survey was central. Survey data included engineers own calculation of the time they give to dealing with code changes in some branch, similarly their own judgment of the difficulties entailed doing code rewrites, and the perceived burdens that dealing with the branching placed on their work. This is juxtaposed with simulations and testing against objective data stored in the SMC tool used in Windows. This allows modeling that identifies whether the files amended in some branch are the same as in some other branch and if so, whether the teams that undertake the changes in question are one and the same. The analysis offered in the paper suggests that the coding work will be more efficient, with less likelihood of code conflicts, when code is reinserted into the code base, if the teams working on the same code are the same, or rather if it is the same people who are modifying the same files albeit

in different branches. If the teams are different, there can be potential problems between a pair of branches, as different people modify the same files and they are likely less coordinated. As it happens, this latter situation is less common, but the analysis helps identify where this is likely to be the case and makes suggestion as to how to alter the branch structure to minimise it.

Without wanting to comment on the empirical merits of the paper – it seems persuasive in its analysis – the view it offers entails understanding the software engineer in a particular way. To put it very simply, it treats the research problem as one of uncovering and understanding the overall SE process, the one into which the software engineer 'fits'. This picture is produced through corroborating findings, statistical investigations and hard facts. As we have remarked, one might say that it is a 'scientific-like' approach – and this is indeed the kind of nomenclature than the authors use. By dint of this orientation, the software engineer constructed creature at the heart of the analysis is a creature that is 'lacking'. It is lacking particularly in objective knowledge and it is not fully aware of all the considerations that affect its own behaviour. The view of this creature, the engineer, contrasts with the wisdom of the researcher, wisdom produced by the method of inquiries. The software engineer is not devoid of knowledge in this view, but when compared to the analysis, theirs is only a partial view, parochial at best.

Before we go any further, it is very important to bear in mind that we are not suggesting that this view, the one that casts the software engineer this way, is incorrect. As we allude above, this (and any other view) is to be judged by what it leads to, what actions that result, as well as in terms of evidence. In the case of this paper, the approach it embodies (and represents) has the great benefit of pointing out the ways understandings of the software engineer can be enhanced. If the engineer is parochial, the output of this research can be used to provide insights and tools that can educate and correct that parochialism. The empirical adequacy of the paper are to be judged in part by whether its insights do in fact lead to better reasoning about the relationship between branches and bug fixes.

Second, the construction of the software animal at the heart of this view are not necessarily bound to the method used, though the choice of method and the construction of the animal might appear to go hand in hand. This is not necessary, we do not believe. The fact that, for example, the authors of the branchmania paper use quite sophisticated statistical techniques to weigh the evidence does not mean they could not have supplemented that material with say, 'ethnographic' type evidence. There might be difficulties bringing the qualitative and the quantitative together but we do not think these are epistemic, so much as practical: so much of the stuff that comes from ethnographic evidence is orthogonal to processual matters for example and so needs sifting out. The important point is the presumption that research adds understanding to the

somehow deficient software engineer. All sorts of information can benefit these impoverished animals, quantitative and qualitative.

View 2: the engineer as 'creative reasoner'

The second view we want to highlight casts the engineer in quite a different light to the one mentioned above, in the scientific vision of the engineer. The long held emphasis on the ethnomethodological concern with reasoning in CSCW (Button & Sharrock, 1994; Schmidt & Bannon, 1992) has led to an approach to SE research that emphasizes and investigates how engineers 'work through' and 'work up' particular software concerns into solvable problems (see for example, the de Souza et al paper mentioned above, 2011). That this is so does not always show itself in remarks on the premises of the research, however; it is, rather, simply often a characteristic of CSCW-type papers.

The paper that Phillips et al (2011) provides a good illustration of how this view casts the engineer in this special way. This reports a study of the informational needs of those about to reinsert 'fixed' code back into the code database. Here the research does not assume that the software engineer is 'lacking'; rather, it shows that they have rich resources at hand which can be built upon and extended, made more general and made richer. The study uses evidence gathered through semi-structured interviews of seven individuals who, like the Bird et al study, worked in a development team on the West Coast. These included release managers, two team 'leads' and two developers – coders in other words. Data from these interviews were coded against a predefined set of topics, generated by a prior survey of branching activities in the SMC tool used in Windows.

More specifically, the study identifies ten 'rules' (or maxims) that are used by a sample of software engineers to determine what is an optimal time for submitting revised code into the code base. One rule holds that the number of lines of code being altered in a branch can be used to predict the likelihood of the difficulty that the engineers addressing that code will face. Thus the number of lines can act as a predictor of the likelihood that the branch in question will produce code later than planned. By the same token, differences in the number of lines between branches may also indicate the likelihood that branches will deliver their code changes on time. Another rule has to do with the ordering of branch integration or the sequence of different code 'reinserts'. Concurrent activities do not always lead to identical times of merging but a sequence - some are upload-able before others and others later. Knowing which, knowing the rule that determines which goes first, and which is dependent upon another, can allow engineers to predict the likely order of problems that will happen when code inserts happen. This sequencing can indicate the likelihood of problems and dependencies that arise after code is put back into the main code base. Another

rule has to do with distinguishing between ‘bugs’ and ‘features’: the latter are nearly always subject to ‘agile iteration’ and change and hence can take longer than predicted whereas the former, bugs, are more likely to have pre-determinable timelines, how long they take can be fixed as it were.

One could go on. The important point is that this approach, then, paints the engineer in quite a different light to view in Bird et al: here it is their capacity and ability that is celebrated, not the contrast between ‘the facts’ and ‘their knowledge’. Research in this vein tends to offer guidance for new kinds of information and data that will provide engineers with tools that refine their ability to use their rich knowledge (see also Martin et al, 2007; also the considerable corpus of research by de Souza and his colleagues). Thus, and for example, Phillips et al propose that more information about the actual timelines of code reinserts be made clearly visible in the SCM tools; engineers know how to use this information but simply need it to be better specified.

View 3: The felt-life: the ‘dwelling of engineers’

One of the things we have remarked upon repeatedly is that those who deploy these various ways of looking at SE don’t often see themselves as constructing a view of the human actor in the centre of their inquiries (the software engineer) in the way we have described. There may be a number of reasons for this, one being a lack of interest in this possibility – it may simply not concern them. Besides, there is also sometimes a conviction on the part of those doing research that their approach has a purity that would make any claims about it being constructive of the subject matter something to be resisted. One can readily imagine those who claim a scientific attitude and who deploy ‘scientific methods’ would hold this view. One might be more surprised that anthropologists hold it too, however, especially given the apparent affinity of their discipline to the soft sciences, and to the humanities in particular. But in fact this is often the case: anthropologists often claim that theirs is the ultimate arbiter of all studies of human action, the ‘totalizing science’ as one textbook writer on the anthropological method put it (see for example Sykes’ *Arguing with Anthropology* of 2005).

We mention this now because it is apparently a form of anthropology (and its methods) that we need to bear in mind as we approach the third view we want to expound. This too constructs its subject. And the way this one does resonate with the style and techniques of anthropology, treating the interview between the fieldworker, the so-called ethnographer, and the subjects of the enquiries as the essential mechanism and topic of the research. In the view of many contemporary anthropologists (even if historically this might have been disputed), it is not the world at large that is at issue, it is not the world that exists outside of the interview that matters; it is the specific interlocution of the anthropologist with ‘subjects’ in those moments that does. To paraphrase David Mosse, it is the

dialogue between the anthropologist and their subject and their subject's world that is the 'crucible' of the anthropological imagination (Mosse, *Anti-Social Anthropology*, 2006): the site of anthropology is thus the interview.

This has all sorts of consequences. One is that it becomes very important 'just who' the fieldworker meets. It also becomes important to do multi-sited ethnography, when the interviews are undertaken in different parts of a subject world (Coleman & Hallerman, 2011). This also begs questions as to what an ethnography might be if it is of the virtual world where the interaction between the anthropologist and that world happens: for this interaction might be with proxies, not real persons and so this might skew the data in certain kinds of ways (Hine, 2000). Besides all this, and to refer to Mosse again, there is also the problem of what happens when those interviewed come to dispute with the researcher: what happens when interviews turn into arguments?

Be that as it may, the reason why we are spending some time on the problem of interview is that unlike the other two approaches we have sketched, the point at which data is gathered, in this case in and through qualitative interviewing, is fundamentally the source and province of inquiry. It is not, say, a largely taken for granted resource, or one that has to be treated with candid corroboration from other data sources. In the Bird el al paper, for example, other data was gathered from a SCM tool; in the Phillips et al paper the process of interviews was treated as a gathering resource that produced information that had to be reinterpreted, 'coded' in reference to other (as it happens unspecified) resources 'drawn from a survey'. The function of the interviews was then to provide evidence to characterise something else, the reasoning of software engineers, say, for corroboration of quantitative facts about bugs, perhaps.

We can illustrate what one focuses on when the topic is the interview with our own data. Like the authors of both the papers mentioned above, we were fortunate enough to get access to the Windows programming team in Microsoft. With this access we sought to interview a range of 'subjects' in this domain. And like them, to get a comprehensive sense of the domain we sent out requests to individuals in a variety of roles, and this resulted in 17 interviews with engineers, from product managers (in charge of several dozen coders), branch managers (who had responsibility for ensuring the development and testing of code before it is re-entered into the main code database), coders and testers. In this way we had access to at least some of the 'sites' of SE.

All the interviews were qualitative, with a simple list of initial topics being used to foster an open-ended, 'constructive' interview that encompassed all that subjects felt important, and which could be combined with discussion of the topics that we thought valuable. Each interview thus informed the next, such that the process resembled a voyage, where the understandings provided at the end built on those created at the outset. As with all such fieldwork, extensive notes

were made in and after interviews, as well as transcripts made when participants allowed tape recordings.

The first thing that came out of the interviews – or rather was made visible in them - was not anything to do with things like 'what the engineer knows' or 'examples of algorithms', nor perhaps more pertinently given the Bird et al and Phillips et al papers: how the branch structure is, say, misleading. Certainly branches were talked about but something else happened first.

What was made clear at the outset of the interaction between us and the subjects was that 'the information' that we were about to be provided with was 'dangerous'. All of the interviews commenced with discussions as to who we were and who might hear or worse 'read' what was shared with us. As these discussions unfolded in the first minutes of each interview, so it became clear that we could only be given information if we understood something about the 'specialness' of what was being given us. Doors were closed, voices hushed, queries made about the security of our recordings. One interview (#3) came to an abrupt end when the subject exclaimed that he 'wasn't going to tell everything' – as if he was releasing something dangerous, fearful, something that our levity made him think we did not fully appreciate; he came to doubt us.

These actions, these patternings in the interviews were not merely people judging whether they ought to allow strangers to pass the gate – into the world of Windows programmers. This is a classical problem in fieldwork (reported in papers that use a different approach such as the Bird *et al* 2009; see also Button & Sharrock, 2006); rather we had to learn and acknowledge in the opening moments of the interview process – even during the interview as with #3 - that we understood that information would be dangerous if it got to the wrong person. We came to learn this by questions posed to us by our subjects such as 'Who would see this work?', or 'Is this just for research, not for Windows?'; or 'Who else are you talking to?' Somehow danger arose when information and persons combined, we were being told. But in being told this we were also being told something about ourselves: if we were to have the information, no danger was implied; our use was neutral, sterile one might say. If we were to act as couriers, on the other hand, that was another matter. These concerns seemed designed to evoke 'fear' in our part. Certainly, they did make us more timid, more respectful, keener to show silence.

What these interlocutions with our subjects lead was to think on was not what they were hiding. We did not see their injunctions as devices to put us off from seeing the truth. On the contrary, the apparently sensitive topics made us think that our prior readings of research on SE had not conveyed the felt-life of SE very well. This life world seemed somewhat drained of colour, certainly when compared with what we were experiencing. This life world also seemed more passionate, volatile and tendentious than the descriptions of reasoning and accountability presented in the CSCW literature. What our interviews, or rather

the experience of the interviews lead us to think of, was Ingold's claim (in *Being Alive,* 2011) that human life entails forms of embodied praxis in particular sites of movement. By this he means that people do not simply react to situations but produce their reactions through confronting the possibilities presented to them and their own aspirations in a kind of dialogue. This mixes material constraints with the trajectories of human beings in rich, resonant and evocative ways: it is the weave of all this that produces the thing called experience.

So in Ingold's view, a real place such as, for example, a software engineers' office will have certain sensual features – a kind of light, an ambient sound, a physical form and each of these material phenomena will have their own properties and dynamics. These are combined with the human-sourced tensions that result when occupants of those offices, engineers, think about and engage with the work they need to do in those spaces, work that entails them sitting at in the light of the windows (where they quiet their minds) and where they gently tap their fingertips on a keyboard. An office may afford silences that allow an engineer to concentrate, but they might need to talk with people elsewhere and so must break those silences to get their engineering job done. They need to navigate these constraints, Ingold's view suggests; engineers need to judge when to work quietly or when to speak noisily. They need to 'dwell' in ways that makes their domain work for them. It was this that was brought to mind when we started our interviews: when we got the 'push back' after our opening remarks, once we had described our own trajectories and 'work'.

Delving into *software* dwelling

To access 'dwelling' Ingold proposes that the fieldworker participate in the experience of the contexts, not in the sense of being a participant observer but in the sense of feeling some of the lived vitality of the places in question. They need to grasp just what it feels like to dwell there. For example, we soon began to learn just how fraught SE can be, bound up with fears and navigated through with powerful notions of who should not be given knowledge and who should be. We were being instructed on the importance of recognizing the trajectories of those who pass by and come through SE dwellings. We were learning to attend to what other researchers had chosen to bypass or ignore.

So what was it that was being pointed to in our interviews that seemed to demand timidity on our part? Moreover, why did this concern, this worry, have an almost mystical quality? And besides who were the wrong persons? As we say, we weren't given information about algorithms.

One of our interviewees (#6) worked on the 'security handles' in the Windows kernel and so one might have imagined that descriptions of code in that context could indeed be dangerous stuff to know – we could have walked out of the room with ways of breaking the Windows security paradigms. But that was not the kind

of stuff were we given. Nor, for that matter, were the persons that seemed implied as dangerous the kinds of individuals that one would imagine: when we listed bosses that had given us access the stating of their names did not get reacted to as if they were dangerous; the hierarchical location of these individuals didn't seem to drive a sense of danger for our interviewees.

What was mentioned continuously, however, and reasserted again and again, was the idea that their code revision decisions, their management of code inserts back into the branch system and so forth, was always motivated by what they wanted us to understand was 'good faith', a phrase used just in passing by two individuals. To paraphrase what we were being told by them and the others we met: the motivations we had described to us were meant to be embody or reflect sympathy, sympathy with and consideration for others. Good faith had to do with how individuals oriented to the problem of how their own work impacted on others. The tales we were presented with in interviews made it clear that behaving in good faith and, conversely, the presumption that others would behave with similar good faith, was something that was sought for and assumed but in practice chronically undermined and tested.

For example, subject #1 explained that, in one instance, the interdependency of his own team and that of another branch broke down. This was not because of poor faith by either party. It was due to poor understanding. As he put it, 'We didn't really take it seriously'. The 'it' he was talking about was a component in their code that turned out to be much more consequential for other branches than he and his colleagues had realized. When his team altered it they did not think it would affect their colleagues in another branch. He explained that they ought to have realized that it would have done so, but as he put it, they were too whimsical, not taking it seriously enough to find out how it had consequences until it was too late. The example was meant to show that it was not that our subjects acted in bad faith, or that their colleagues did so; it was rather that trying to act in good faith by our subjects was very hard work.

It was partly in this sense that the information being shared in the interviews was dangerous: it was dangerous because it begged questions as to whether we, us researchers, 'really wanted to find this out': the implication in this formulation being that we concocted a picture in our minds of a pure world of branched software engineering where decisions and practices were not sullied by the failures of engineers who didn't take everything seriously enough. This aspect of danger then was not pointing towards the danger of, say, managers 'finding this out'. It was not those who were the dangerous souls we had to avoid sharing our cargo with, it was us.

We shall come back to this concept of danger in a moment. But before we do so we want to note that in several interviews we were provided with examples not of passion and anger, but with tales about attempts to negotiate and bargain when the thing that was being bargained over was the right to adjust some code.

‘We didn’t own that problem’ (#6), was a common phrase used to describe attempts. What was being illustrated was not giving work to others but taking on work for oneself. Individual decisions about how long some coding work would take, about the consequences of that work and so forth, were not simply questions about one’s own activity but always and endemically about the implied work that this had for others.

Unfortunately, the decisions that an SE would make might not always have the interests of other SEs at heart; sometimes SE’s had to look after ‘themselves’. Sometimes ‘one’s own interest’ had to take precedence. Just as our subjects recognised that they would be selfish, so they accepted that others would be too: ‘they told us to politely go away’ one subject told us when they had sought to have responsibility for some code given over to them. If ‘they had been given it’, he explained, the other group judged that their work would be made greater. There was no knowing whether this was ‘objectively’ true, of course, but it was assumed to be a reasonable possibility; thus they rejection was ‘OK’, the engineer explained.

At the same time, we were told that there was danger in the relationships those we were interviewing had with others, danger when these simple negotiations failed. We were told that those ‘others’ got frustrated and angry, and would sometimes find excuses like ‘not taking it seriously’ simply not good enough. Their reactions ‘didn’t leave a very good taste in my mouth as .. they jumped up and down’ (#6).

So if one part of the danger had to do with whether ‘we’ (i.e. us researchers) could handle a real understanding of the world rather than a tidy ‘researcher’s vision’, another aspect of the danger had to do with the fact that things talked about were things that caused friction between persons: there really was anger, resentment, fury. In this view, software engineering naturally led to ‘dangerous situations’.

Were our subjects telling us that this organizational context was riven by ubiquitous personal animosity? Not at all; for our participant’s were seeking to instruct us that the relationship between software practice and large scale programs of coding, manifest in this case in a vast branching structure, was not simply a question of schedules and planning; the relationship between action and system was not mechanical.

What we were being led to see was that the world of a programmer is a contested one, where the motivations and desires of one individual can come to be played out at the expense of another, and where the danger that we needed to acknowledge had to do with the fact that people ‘naturally’, sensibly and understandably get angry and resentful, bitter at the lack of good faith of others. At the same time as learning this, we were not being told that the overall system of which our subjects were a part was collapsing; quite the reverse: our subjects told us stories that not only instructed us to see the natural order of passion in

software engineering, but also the fact that most often, and in most instances, targets were hit and deadlines met despite the tensions that arose in the work itself.

In this sense, our subjects were saying that although the maxim of having good faith in others might not be constantly abided by, the organisation itself could have faith that its workers would deliver the goods despite the stresses and strains. In sum, what our interviewees drew attention to is human passion. Their goal was to teach us, to instruct us to see, that SE is not a computational-like activity, it's about an activity that is all too human.

Conclusion

Our proposal has been that within the SE literature various kinds of human animal can be seen. These animals are created by a mix of assumptions about topic, choices about methods, and following on from that, treatments of evidence. Key to the perspective we have just presented is the way the subjects of it, in this case software engineers, don't simply act as conduits of the facts, conduits between their world and the world of the interviewer. Rather, interviews with these subjects were opportunities wherein the software engineer in question instructed us, the interviewers, on how to understand and orient to their world, the felt-life world of software engineer. As they did so, so the nature of themselves as engineers in their dwellings was, as it were, 'determined' in what Ingold would call 'the lived praxis of interlocution between interviewer and subject' (Ingold 2011, particularly 229-243).

One important distinction between this view and the prior views sketched is that here it is the engineer who was treated as the expert; the researcher is treated as parochial. Or rather, the researcher is treated as gauche – hence the concern about what the researcher wants to find out. The reason for this concern is that the software engineer, the animal at the heart of the world, is quite different from the one that the SE's think the researchers expect; this is particularly in what we have called, without any perjorative intent, the scientific view – the one in for example Bird *et al*.

A similar distinction is to be found between the view that the real, embodied, software engineer wants to convey about the 'human predicament' of their circumstances and the 'reasoned professional' described in what we called the CSCW-type view. This latter view is an approach that emphasises the creative, situated purposeful reasoning of engineers exemplified in Phillips' *et al*. This 'reasonable person' view is also exemplified in de Souza *et al's* rich and well-argued study of awareness (2001: pp325-339); just as it is in, for example, Storey *et al's TODO or To Bug* paper (2008). But this is quite unlike the world as the subjects in our study wanted to convey.

The world that the view we have emphasised is populated with experts, to be sure, but these experts are of a particular kind, one that might frighten the researcher. For these individuals want to make sure that visitors to their space, to their dwellings, walk away with a sense of the heat within: SE is not about calm algorithmic reasoning alone; indeed that hardly conveys the sense of it. Those who dwell seem to want to highlight the structures of human passion within. Theirs is the world that looks, for all practical purposes, like the one the great Scottish philosopher Hume populated, one where the logical reason of the individual has to be understood as bound to the human nature that produces that reason: that is to say one where action is affected by anger, by choices about good faith, or coloured by resentment about the distribution of labour. In the world of Hume and so in the dwelling of software engineers, the animal at its heart is 'flesh and blood and reason'. Just as passion and logic are married for Hume in ways that appalled the eighteenth century rationalists, so it would appear that passion with coding is the key to be what can be seen when studying SE. In this view, SE is to be understood as an all too human affair; that is why it is dangerous to ask about and dangerous to know. Dwelling here is all too human.

How different this vision from Bird, from Phillips, from de Souza or Storey *et al*; how particular its topics and insights. The discussions in this paper have suggested quite a different way of thinking about evidence about software engineering. This holds that we might well be able to merge views on some cases but we might also find that the kinds of views we are marshaling are seeing quite different phenomena, constituting different animals all called engineers. When we recognize this we might not be so easily persuaded to conclude that one view on this animal is 'righter' than another; nor might we be so keen to bring them together. When we are confronted by such possibilities, the suggestion put forth in this paper is not that they should be avoid doing this at all costs but rather that one should treat doing so with care – triangulating the arithmetic of branch costs with the sense of dwelling in SE seems a hard thing to do, perhaps not even a sensible thing to do.

Besides, a more important concern might be to investigate instead what a one of these approaches allows one to do alone. In the case of the last view we have presented, the view that emphasises the sense of dwelling articulated in interview, then a number of obvious implications follow on. One has to do with realisation that SE in large corporate enterprises (where the code base is also large) involves considerable interpersonal and organisational skill, skills it should be clear of a peculiar kind, 'human ones' we have said. This is not simply because there is a loose fit between branch structures and code elements; it is because of the felt-life of existence within such enterprises: to feel concern for an unknown other is surely a different feeling for those one knows; but to worry about the travelers who pass through one's office and their likely destination is another. Part of the sensibility required to leverage this view turns around the fact that what is a

finding in other approaches is the assumed starting place here: those who dwell in the spaces described know that large code bases are not designed around nucleated elements and that there is an inevitable blurring between one component and another. They know too that even the most ardent branch design may well fail to ensure that the mapping between who does what can in all cases guarantee that what one person's code does cannot have (in some obscure way) an unexpected consequence for someone else's work. In this sense, the suggestion by the agile theorists like Williams mentioned at the outset (if theorists is the right word) that there needs to be a constant desire to communicate on the part of engineers, often face to face, is as true now as when the agile turn was first initiated all those years ago. But what the approach focused on dwelling highlights is how profoundly this is felt; how agitating of the spirit this can be. Those approaches that have looked at reason and accountability seem to eschew this very fact: just how communication is facilitated, managed, controlled, and acted upon in the dwellings of software engineers. If our characterisation has pointed the way, then much further consideration of these spaces is required. The view that the SE animal is passionate means that the human arts of reasoned negotiation are all the more important to enable but this may not be something that will only entail reasonableness. The turn to SNS may not lead to the more effective articulation of needs, as Storey, Treude, et al imply (2010). It might also facilitate the vituperative and the ill-considered; off-the-cuff explosions of vented passion. As Rettberg notes in her book *Blogging*, this is certainly what appears to be the emergent norm in the blogosphere (2009). Harper (2011) confirms this view.

Another has to do with how attempts to theorise reasoning by software engineers need to recognise the importance of the dwellings in which it occurs. For what one can say about this last study, brief though it was, is that it highlights how software engineers undertake regular, continuous and often artful ethical decision making. This decision making looks nothing like the abstract rendering of reason one finds in, for example, philosophical studies of ethical choice, especially those that deploy the so-called trolley method of enquiry, where ethical questions are posed in a totally hypothetical manner. There is nothing hypothetical about ethics in SE. Nor does it look like those representations of human reasoning articulated in game theory which is increasingly popular. As O'Connor notes (2012) the trolley method is so devoid of linkages to real situations of choice that the kind of reasoning it does illuminate are almost completely egregious. This will apply to game theoretic models when attempts are made to apply them to abstract renderings of SE for the same reasons.

What this approach to SE brings to bear then is the kind of ethics that SE entails. Some years ago John Bowers made the claim SE in CSCW should become ethical. This assertion was evidently without any reference to what engineers actually do in organisational life, day in, day out. They make decisions

not about what to code but about how their coding choices will affect others and how the choices will affect them in turn. This is endemic to SE practice.

That this is so should make it clear that a concern for dwelling does not make available the world of coding; it highlights how coding turns around relationships between organisational roles as kinds of identity, the community to which identities owe affiliation (the workgroups or gang in which an SE fits), such things as the moral rights to comment and act upon code elements and not others, as well as the felt-life of that ensemble.

As we conclude this paper, so one ought to be able to see now that one can see the analogies between the structures in the felt-life and such things as tribal structure, kinship systems, distributions of 'rights to know' and 'rights to act' in places far distant from the development offices of Windows. These analogous ensembles are most often to do with religious matters, with who has access to the inner sanctum of some holy place. But, here, we have seen through the casting light of a concern for dwelling that when it comes to the world of Windows software engineering on the 'West Coast', access to and control of bits of code is similarly drawn. Who can get to the code in the branch, who has access to that branch and who is prohibited are not strictly rational matters, we have seen. Like priests protecting the sanctum of their temples, those who dwell in software have much to protect. But whether it is software or theology, it is all the same from the analytic point of view used in this paper. As Foucault suggests, it's a question of how one constructs the human at the centre of the world one is interested in, and, as a consequence, what that world comes to look like given the human put in the heart of it.

References

Bird, C. & Zimmerman, T. (2012) Assessing the Value of Branches with What-if Analysis. In *Proceedings of the 20th International Symposium on Foundations of Software Engineering* (FSE 2012), Research Triangle Park, NC, USA, November.

Bird, C. Nagappan, N., Devanbu, H., Gall, and Murphy, B (2009) Does distributed development affect software quality? an empirical case study of windows vista. In *Proc. of the International Conference on Software Engineering.*

Bird, C. N. Nagappan, P. Devanbu, H. Gall, and B. Murphy. (2009) Putting it All Together: Using Socio-Technical Networks to Predict Failures. In *Proceedings of the 17th International Symposium on Software Reliability Engineering.* IEEE Computer Society.

Bird, C., Gall, H. Hagappan, N., Devanbu, P. & Murphy, B. (2011) Don't Touch My Code!, *ESEC/FSE'11*, Sep 5- 9.

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

Button, G. & Sharrock, W. W. (1994) Occasioned practices in the work of software engineers. In Goguen & M. Jirotka (Eds.), *Requirements engineering: Social and technical issues*. San Diego: Academic Press.

Cleidson R. B. de Souza David F. Redmiles (2008) An Empirical Study of Software Developers' Management of Dependencies and Changes, *Proceeding ICSE '08 ,* pp241-250.

Cleidson R. B. de Souza David F. Redmiles (2011) The Awareness Network, to Whom Should I Display my Actions, And, Whose Actions Should I Monitor?, IEEE *Transactions on S.*.E. VOL 37, NO 3, 325-339.

Coleman, S. & Hellerman, P. (2011) *Multi-Sited Ethnography*, Routledge, London.

Dittrich, Y., John, M., Singer, J., & Tessem, (2007) Awareness in the Wild: Why Communication Breakdowns Occur, *Global Software Engineering, ICGSE 2007*, pp81 – 90.

Cleidson R. B. de Souza1,Redmiles, D. Cheng, L. Millen, D. Patterson, J. (2004) Sometimes You Need to See Through Walls —A Field Study of Application Programming Interfaces, *CSCW'04*, November 6–10, 2004, Chicago, Illinois, USA.

Foucault, M. (1966) *The Order of Things*, Chicago University Press, Chicago.

Garfinkel, H. (1967) *Studies in Ethnomethodology*, Englewood Cliffs, NJ: Prentice Hall.

Gutwin, C., and Greenberg, S. (2004) The importance of awareness for team cognition in distributed collaboration. In *Team Cognition: Understanding the Factors that Drive Process and Performance,* APA Press, 177–201.

Harper, R. (2011) *Texture: human expression in the age of communications overload*, MIT Press, Boson.

Hine, C. (2000) *Virtual Ethnography*, Sage, London.

Ingold, T. (2011) *Being Alive: Essays on Movement, Knowledge and Description*, Routledge, Abingdon, UK.

Kuhn, T. (1962) *The Structure of Scientific Revolutions*, Chicago.

Martin, D., J. Rooksby, M. Rouncefield, and I. Sommerville (2007): 'Good' Organisational Reasons for 'Bad' Software Testing: An Ethnographic Study of Testing in a Small Software Company. In *Proceedings of ICSE'07*, pp. 602611.

Mosse, D. (2006) Anti-Social Anthropology? Objectivity, Objection and the Ethnography of Public Policy and Professional Communities. *Journal of the Royal Anthropological Institute* 12 (4): 935——956.

Nagappan, N., Murphy, B & Basili, V. (2008) The in Influence of organizational structure on software quality: an empirical case study. In *Proc. of the 30th international conference on Software engineering.*

O'Connor, J. (2012) The Trolley Method of Moral Philosophy, *Essays in Philosophy*, Vol 13, 1, 14.

Paul, S., and Reddy, M. (2010) Understanding together: sensemaking in collaborative information seeking. In*, CSCW '10*, ACM (New York, NY, USA, 2010), 321–330.

Phillips, S., Sillito, J., and Walker, R. (2011) Branching and merging: an investigation into current version control practices. In *International workshop on Cooperative and human aspects of software engineering, CHASE '11,* ACM (2011), 9–15.

Rettberg, J. (2009) *Blogging,* Polity , Cambridge.

Rönkkö, K., Y. Dittrich, and D. Randall (2005) When Plans do not Work Out: How Plans are Used in Software Development Projects. *Journal of Computer Supported Cooperative Work*, vol. 14, no. 5, pp. 433-468.

Rorty, R (1979) *Philosophy and the Mirror of Nature*, Princeton University Press: Princeton.

Schmidt, K and L. Bannon (1992) Taking CSCW Seriously: Supporting Articulation Work. Journal of *Computer Supported Cooperative Wor*k, vol. 1, nos. 1−2, pp. 7−40.

Sharrock, W. & Read, R. (2002), *Kuhn: Philosopher of Scientific Revolution*, Polity, Cambridge.

Sommerville, I. (2007). *Software engineering*. 8th edition, Pearson Education.

Storey,M. Ryall, J. Bull, R., Myers, D. inger, J. (2008) “TODO or To Bug: Exploring How Task Annotations Play a Role in the Work Practices of Software Developers”, *30th international conference on Software engineering (ICSE)*.

Storey,M., Treude, C. van Deursen, A. Cheng, L. (2010) “The Impact of Social Media on Software Engineering Practices and Tools”, *2010 FSE/SDP Workshop on the Future of Software Engineering Research (FOSER 2010)*.

Sykes, K. (2005) *Arguing with Anthropology*, Routledge, London.

Williams, L. (2012) ‘What Agile Teams Think of Agile Principles’, *Communications of ACM,* pp71-76, Vol 55.no 4.

Walrad, C., and Strom, D. (2002)The importance of branching models in scm. *Computer 35* (September 2002), 31–38.