Users as Contextual Features of Software Product Development and Testing

David Martin
Xerox Research Centre Europe
Grenoble, France
david.martin@xrce.xerox.com

John Rooksby
Computing Department
Lancaster University, UK
rooksby@comp.lancs.ac.uk

Mark Rouncefield
Computing Department
Lancaster University, UK
m.rouncefield@lancaster.ac.uk

ABSTRACT
This paper examines how software developers discuss users and how such discussions are intrinsic to the negotiation and settling of technical decisions in the development and testing of a software product. Using ethnographic data, we show how the user features in conversations, not as a 'topic' but as 'context' to technical work. By understanding the user as a contextual feature in developers' group work we are able to draw attention to issues in the use of Extreme Programming for software product development. Extreme Programming is a participatory design method, but software product development involves envisioning and designing for future customers.

Categories and Subject Descriptors
D.2.1 [**SOFTWARE ENGINEERING**]: Requirements/ Specification – *elicitation methods, methodologies.*

General Terms
Management, Documentation, Design, Human Factors, Languages.

Keywords
Extreme Programming, Software Product, Ethnography.

1. INTRODUCTION
This paper discusses the role of users and customers in the implementation and testing of a software product. This is not a paper about participatory design; users and customers hardly participated in the work we observed and when they did they were kept at a certain distance. However, who the customers and users were or might be, and what they might want were prevalent concerns. Accordingly, in addressing the longstanding interest in user involvement and representation in the design process, we are interested in how the cooperative, group work between software product developers involves calling for, structuring and applying expertise and knowledge about users and customers. We find that users are a contextual concern insomuch as, in accordance with Garfinkel & Sacks' [12][26] re-specification of context, users are continually relevant to technical work but are spoken of as and where it becomes necessary to do so.

We focus on the use of the method XP (Extreme Programming) for the development of a software product. We do not criticize XP, but address: how XP for software product development, as a form of cooperative work with and for current and future customers and users is handled; how customers and users feature in on-going design and development decisions; and how risks are encountered and minimized in this form of development. We begin this paper with a background discussion of two significant studies of the representation of users during product development. We then describe our fieldwork at a software product company, and present and analyse three examples.

2. BACKGROUND
Woolgar [33][34][13] studied the development of a desktop computer, taking particular interest in the testing phase. He noticed that discussions of users were common amongst developers and that these users were not actual 'people' as such but categories for and stories about people that came to make sense through an organizational frame. This came about partly through difficulties of getting to know actual users from within the company, but also through conflicting demands in providing for various users and 'users in general'. Furthermore there were issues of competing understandings and of knowledge and expertise being distributed within the company, and of there being a need for the company to 'create the future'. Woolgar claims "The whole history of a system project can be construed as a struggle to configure (that is define, enable and constrain) the user" (Woolgar [34], p207). Woolgar's line of argument rests on the idea of the computer as an artefact that crosses an organizational boundary. This artefact is produced in an organizational setting which fixes certain normative modes and methods for its use, that if adhered to will facilitate and sustain its 'working'. In other words, developers try to 'figure out' what their users require, and then design their system to fit that. However, 'figuring out' is necessarily shaped by the way the organization works, technological and development constraints and opportunities, and partial and differential access to a heterogeneous user group. Whilst Woolgar discusses developers' talk about users, he does so in order to move onto ideas about the social relations between users and designers that sustain a technology. It is upon this point that the many subsequent takes on 'configuring the user' depart. Lindsay [19] notes "Woolgar's configured user is inextricably intertwined with the development, especially the testing phase of the technology" (p31) and expands

Group '07, November 4–7, 2007, Sanibel Island, FL, USA.

on ways in which users of desktop computers are otherwise configured, for example in advertising or how user groups sustain older technologies. Bowers reapplies the concept in looking at the ways bank staff get customers to follow preferable courses of action (Bowers & Martin [4]), and at how users talk back to designers (Bowers & Pycock [5]). Mackay et al [23] look at the rapid prototyping of bespoke software. Both Bowers and Mackay appropriate the concept 'configuring the user' to discuss 'something done to someone'.

Sharrock & Anderson's [29] work on users as a 'scenic feature of design space' is reasonably contemporary to Woolgar's but less well known. It follows a similar interest in the ways designers discuss users during technology development, in this case the development of a photocopier. They point out that "users were not participants in the design activities which we observed [but] what users would want, what they might do, what they would be willing to accept were treated as significant and sometimes even decisive" (p11). Significantly, they also point out "the issue of what users would or would not do arose in the context of some other topic" (p13). There is some parallel here with ethnomethodological studies of law, as Lynch [21] discusses of the preparation of a court case:

> "projections of courtroom procedure … are tied to the judge's, or a judge's, possible reciprocal actions. In different instances, the judge's preferences and tendencies are assigned to a particular individual, treated as a local preference for specific protocols, or ascribed to a disposition to act in a particular way in "cases like this". The attorneys invoke the judge as an organizational principle that locally governs the presentation of the case at hand. Whether or not the attributions about the judge made by the attorneys are accurate, they incorporate the judge into the practical organization of the projected cases, and by so doing they realize the judge in their procedures for presenting cases at hand." (Lynch [21], p103)

As Lynch observes, although a judge is not physically present, he or she is still invoked and oriented-to in differing degrees of generality by participants. The design of a software product is oriented towards potential users in a similar way. In discussing product development Sharrock & Anderson notice how knowledge and expertise about users appears as 'typifications'. These typifications appear for practical purposes in getting technical work done and accordingly user typifications are defined not for what users are, but for how they are significant to the work at hand.

Woolgar and Sharrock & Anderson are looking at similar aspects of similar situations and devise similar accounts of how developers realize the user. This paper builds upon their work not only by applying a similar ethnographic approach to a different software engineering method, but also by using more detailed ethnographic data enabling us to stress the user as a 'general' typification, and to highlight the role of argumentation. Also, by examining a small software company rather than a global corporation, and in particular by examining a complaint, we are able to address occasions where users are seen to have definite histories and abilities.

3. ETHNOGRAPHY OF AN EXTREME PROGRAMMING TEAM

Complementing previous ethnographies of XP in Practice (eg. Mackenzie & Monk [22], Sharp et al [30], Hunt et al [16], Chong & Hurlbutt [8]) this paper presents an ethnography of software developers in a small software company developing a software product for business customers. The study employed observational methods and in-situ interviews to view, capture and understand work as it happened via note taking, video, photographic and audio recordings. A total of 30 days fieldwork was undertaken in a period between July 2005 and April 2006. Over the same period we have also been involved with an agile methods interest group, convened for discussion and peer learning amongst developers, and one of the authors has prior experience of using XP in development.

The company 'IDEco' (a pseudonym) produces an IDE (integrated development environment) for end-users to develop applications, using an XML based language and a graphical screen designer. The applications are to run on various mobile phones and other mobile devices. During the study, the company had seven full-time employees, four of whom were programmers (Paul, Tom, Dale and Mick – all names have been changed). There was also a technical director and trainer (Shaun), a customer relationship manager - CRM (Gordon), and a financial administrator (Brian). The programmers at the study site use practices from XP including an on-site customer (although this role is filled by Gordon the CRM), frequent releases, and what is known as 'the planning game'. They do not do test driven development or practice pair programming, preferring (as seems very common for small programming teams) to work sitting around a large shared table.

4. THE CUSTOMER IN EXTREME PROGRAMMING

Extreme Programming, or XP, cannot be outlined in entirety in this paper, but it is necessary to give some flavour before discussing the role of the customer. XP is arguably not 'just' a method but also a set of skills. For example, along with the "core practices" of XP are four "core values": communication, simplicity, feedback and courage. These values are intrinsic to both justifications for the practices and advice for their effective use; therefore XP has recognizably human centric concerns. Jeffries [17] presently lists the core practices of XP as Whole Team, Planning Game, Small Releases, Customer Tests, Simple Design, Pair Programming, Test Driven Development, Design Improvement, Continuous Integration, Collective Code Ownership, Coding Standard, Metaphor and Sustainable Pace. The customer has a role in many of these practices, for example the 'whole team' includes an onsite customer, and this customer is to take a decisive role in the development and prioritization of requirements and the specification and running of acceptance tests (see Beck [1], Jefferies [17]).

Lippert et al [20] highlight that XP was originally developed with relevance to bespoke, in-house software development in large, North American organizations. As the method has been popularized it has been somewhat modified by its founders, and extensively appropriated by its users. Lippert et al call attention to the following appropriations: the development of application frameworks, the development of eBusiness applications, software

product development, and outsourcing. With each of these, Lippert et al associate a number of problems and common to each are problems to do with the customer relationship, particularly that:

> "In essence, XP combines the sponsor and user into the role of the customer. This can lead to problems if the user interests and the business interests of the customers are not reconciled" (Lippert et al [20], p21)

Concerning the development of software products, Lippert et al's main point is:

> "Sometimes the developers do not have access to users due to the project type, for example a standard product is being developed. In this case, we would have to think how the user can be substituted, perhaps by assigning the role of the customer to someone from the development organization." (Lippert et al [20], p20)

Lippert et al's recommendation is to use a product manager as the customer. In many software product companies a product manager or (as with the case study in this paper) a CRM (customer relationship manager) acts as such. This creates a situation in which the 'XP-customer' is producing requirements for something they are able to sell. This situation might lead to some ambiguities in what is required, but more-so creates the situation where much feedback and acceptance cannot take place immediately. It is not until the software has been bought and put to use that problems or new requirements related to a feature can emerge. As the product gains a user base, the XP-customer also begins to find herself as a mediator for requirements or issues that arise from customers. The person in this position clearly has an important role to play as the connection between the development team, current customers and future markets. Their work involves sorting through requests, opinions and so forth from all sides, as well as judgments over future market directions to come to decisions about how best to proceed with the design.

In product development then, it is necessary to handle multiple customers and 'ideas of and for customers' in a variety of ways. Software products must be sold, and thus the sale-ability is clearly an important development concern. However, the software is also a product to be used. These twin and sometimes conflicting concerns feature prominently in our material. To question what the relationship is between the programmers and the XP-customer would be accepting that in this model the CRM is ultimately responsible for decisions. However we have noticed that there are multiple and dynamic ways in which user and customer talk occurs amongst the 'whole team' in work. Our experience has been that the developers are far from shielded from figuring out customer requirements and tests but take central stage.

5. TALK IN DEVELOPMENT WORK

> "'Technical work' viewed from the point of getting it done involves the determination of such matters as how much work there is to be done, how long it will take, how many must be involved, how much time is available, how those involved are to combine their activities to carry the work through, and how they are to ensure that their activities will remain coordinated and synchronised over its course, what is to be done in various eventualities, who will make the judgement as to whether the work has been done satisfactorily and what it will take to satisfy them." (Sharrock & Anderson [28], p161)

XP, as much as any other development work (or 'technical' work, or the work of survey researchers as discussed by Garfinkel [11] and eluded to in the quote above), involves socially organized activity. That said, the XP literature places a much greater emphasis on interpersonal skills than do most other methods. Face-to-face interaction in particular is encouraged in place of and in spite of documentation. At the heart of the XP ethos is a notion that programming works best as an intensely cooperative enterprise, and that this cooperation results in better design and better code. On-going talk should enable code that is more of a group production and therefore better written, better tested, better integrated, and less likely to contain errors.

> "All members of the project team should communicate intensively. Special importance is given to personal conversation, since information is exchanged more effectively this way. In particular, misunderstandings and ambiguities can be ironed out immediately. If intensive communication between members of the project is guaranteed, a good part of the otherwise normal documentation can be done away with." (Lippert et al [20] p3).

The developers' office at the fieldsite has been purposefully chosen and set up to allow face-to-face talk and interaction. This is true of most 'agile' office spaces but by no means all (e.g. see Hunt et al [16]). Whereas the XP literature promotes office space with quiet zones where programming pairs cannot be interrupted (e.g. see Sharp et al [30]), the programmers at the fieldsite, as seems very common, share an open office space and sit around a single, large table. Their talk will generally be of differing levels of intensity during different stages of the development iteration, for example planning is predominantly talk based whereas writing the user manual is regularly done in silence. During development and testing there will sometimes be much talking, and sometimes hours of near silence. That the talk we report on, and the work it achieves, is routine and mundane is attested to by the fact that in our later interviews with staff it was often difficult to get them to recall the specific events of what were apparently slightly heated discussions. Ambiguities about how to proceed, conflicting ideas, different evidence for different solutions and disagreements are a routine part and parcel of the work. While we focus on the everyday, here, we do not discount the special and atypical circumstances that may also arise and be informative (cf. Nardi [24]), it is just that during our fieldwork we never encountered situations that stood out in this way. Problems, and indeed certain 'nasty surprises' (cf. Garfinkel [11]), were encountered, but not as remarkable events. That problems will arise routinely and that programming and development throws up nasty surprises is very much oriented to as the 'nature of the business'. Of course not all customer talk is in the thick of work around the table; customers might potentially be discussed at lunch, or over the phone.

Development work is an intensively cooperative endeavour (DeSouza et al [10]). The developers' awareness of each other's work is embedded and embodied within the particular configurations (Heath et al [15]) of the office space, the artefacts and the thoroughly normalised occasions to talk, things to talk about, methods to get someone's attention, and means of argumentation. Amongst developers at the fieldsite, premising a discussion is often a case of just saying something. We often saw

developers just say something out loud, for example exclamations such as “Build failed!” or “I can’t, I just can’t!” Such exclamations can be responded to by a question or simply ignored. Sometimes more direct statements are made or questions are asked, for example “Tom’s not worked on the manual for ages!” or “You got the VM?” There are also more coordinative remarks, for example “Flag’s up”. Body language also seems important in initiating or 'premising' discussions. One developer often appears to invite a conversation by sitting back in his chair (often accompanied by a large huff). The same developer will regularly sit back in his chair when holding discussions. Another developer often seems to move his head to the right hand side of the monitor when he wishes to initiate or engage in discussions. Such issues are not limited to initiating a conversation but also to enter a conversation that is ongoing between other developers. Despite such intricacies in appropriate behaviour, it does seem that the developers expect a right to talk and a right to question. This is in line with what Jeffries [17] promotes as ‘collective code ownership’.

We will discuss three example conversations about customers and users. Firstly talk about the usability of a downloadable demo version of the product for people who might be thinking of purchasing it. Secondly of resolving whether a problem in using the product within a ‘virtual machine’ matters. And thirdly an example of the developers discussing complaints and requests emailed to the Customer Relationship Manager by existing customers.

5.1 Example One: Usability and Sale-ability

In this first example we discuss the design of a menu for the selection of mobile-device emulators. It is quicker to test code using an emulator than to use one of the actual devices, and hence most users first run code on an emulator and on the actual device only when the code is complete (there are many exceptions to this rule, particularly amongst experienced developers and testers aware of the bugs in emulators). There are a finite number of emulators relevant to the software product and these are listed in a menu. The problem is that no emulator is installed automatically with the product but each must be installed separately by the user. Selecting menu items pointing to emulators that are not installed gives an error message informing the user of this. The developers discuss this:

S “Users don’t bother reading the error message … they will see it as an error with the IDE itself.”

Shaun suggests that emulators not installed should be “greyed out” in the menu. Tom disagrees and suggests writing a better error message, which at the moment is fairly cryptic. Dale seems annoyed and agrees with Tom.

S “look, I know its stupid but there you go … the most important thing is that it runs out the box … it is a business decision … I would want it working for the initial play. The first 10 minutes, it should work out the box.”

G “Is it hard to make it work out the box?”

S “No it’s trivial. We want it to work out the box.”

D “All you’re doing is postponing the issues.”

S “Good. Until after they’ve bought it.”

The conversation goes on about how the users will know how to install additional emulators if the menu item is greyed out. The conversation includes the sequence:

P “If you’re going to install an emulator, you have to read the instructions.”

D “Well people don’t read the manual.”

S “Well people don’t read error messages either.”

Other possibilities are now explored, including having a tick or a cross by each emulator or having hover help. Dale is adamant these are bad ideas and will make the IDE harder to use. The conversation is soon brought to a close:

S Refers to an email from someone who couldn’t get the demo to work “He wasn’t happy about the trouble to just get ‘hello world’ running … It’s a matter of impression. …”

D “You just make it easier for people to evaluate but harder to use.”

P “I thought we decided on the tick and cross. Because with a cross it’s not greyed out, you can still click on it.”

D Continues to defend use of an error message.

G “Do a tick and a cross and move on.”

M “I’ll do it.” said humorously, implying they must not let D be responsible for this task

Resistance from two of the developers (Tom, but particularly Dale) leads to Gordon the CRM making an order about what to do and for the conversation to move on. Up until this point a number of options have been explored, as seems typical with many decisions about the software. Things are often talked through, and different possibilities are raised. Therefore it is significant that Gordon not only tells them how the menu will be designed but also that the conversation will move on: it is ordinarily acceptable for any decision to be questioned. Mick, with humour, takes on the responsibility for this task, acknowledging that they must do what Gordon says, and that Dale is not going to agree. We see many interesting aspects to this example, not least the seemingly different orientations to the software as either an easy to use or an easy to sell system, and the accounts of what it is for the software to be working. We will introduce two more examples before a discussion of such issues.

5.2 Example Two: Causes and Solutions to Technical Problems

In this example we describe the discovery of a problem with the software. This problem is eventually categorized as a ‘known problem’ rather than solved.

P Working with D at D’s machine “Your machine’s just died.”

M Entering conversation from across the table “What you done?”

D “I’ve dragged a control …” …

P "You know what it is, you got a massive drag threshold."

D "No, I got it set to one."

M Goes round the table to stand beside D. "It's like the delay that's set in the threshold" There is a short period of silence. "I mean its slow all round and I don't get that. It's on a machine that's faster than mine. The only time, is when you've got many selections and you do a validate on drag.

Paul and Mick go on to compare the software running on Paul's and Dale's machines. Dale is running the software using a virtual machine (VM) and they find that it is much slower in this virtual machine. They all agree that "it's weird", three of them repeating this same phrase (but not in any 'dramatic' sense). After some further discussion they query whether the problem is in running a JVM (Java Virtual Machine) in the VM:

M "It might not necessarily be Java though."

D "Yeah I think its Java."

M "The JVMs running inside the VM?"

D "Yeah"

M "I suppose its something we can document as a known problem."

P "Yeah"

M "If you're running your JVM on a VM, or we could say…"

There is a period of silence

M "It could be where [unnecessary events keep firing] but normally in Windows it would lose them so it doesn't do anything. But in the VM its firing all the time. We could change that code. But I don't know how. Its not like we can intercept the events in Java. It won't let you."

D "I'll Google it, see if anyone else has noticed."

A short while later Paul finishes installing a VM on his machine. He confirms that the same problem occurs. Mick speculates that it is an issue with Swing (the Java graphical user interface utilities package). This would probably imply that the issue is out of their hands. Later they further evaluate the issue. They talk about "Joes off the street" and whether such people would use VMs. The consensus is that ordinary people wouldn't have a VM. They then talk about what they themselves do with a VM:

P "But you would never develop in a VM. Our stuff doesn't work well in a VM, but you wouldn't develop in a VM. And our guys are developers."

D "Well supposedly."

P (laughing) "Not really from what we've seen…"

D "Too harsh!" …

M "Its good that you're using that and that we've found it. If we got a call coming in we could say "Are you using it on a VM?" and they would say "oh yeah!". It would be interesting to see how many we got of that nature."

They decide to categorise this as a 'known problem', and it will be listed in the user manual as such.

In this example we see how errors arise and the strategies the developers have in dealing with them. A problem is 'discovered' and, when attempts to solve it get complex the relevancy of solving it gets discussed. In this instance the initial 'incorrect' proffered diagnoses and investigations are not simply a product of impatience and inexperience. These 'recipes' (Schutz [27]) are functional and often lead to quick results. They begin with appeals to organisational knowledge ("has this happened before?" "What does this remind you of?") and on this failing they switch to first principles, particularly a comparison of two machines running side by side. They never identify the cause of problem but are satisfied it is out of their hands and go on to consider whether this problem is a problem for them to worry about. By categorizing it as a 'known problem', the problem fits into the bureaucratic system and can be put off or dealt with within orderly and normal work (i.e. it could become a requirement for solution in a later iteration).

5.3 Example Three: Customer Requests and Complaints

Our final example is of the developers figuring out the meaning of a customer's complaints and requests. These were sent by email to the customer relationship manager, who in turn has forwarded them to Paul. The email is difficult to read and parts of it seem strange:

P "Do you want to hear the second of [this customer's] issues? This one's a bit more normal. Toolbar command not in the GUI demo and he's wishing it to appear."

M "Which toolbar? We don't have a toolbar!"

D "I think it's the buttons."

A discussion ensues about just what the customer is talking about. 'The buttons' is one idea of several that come up and they eventually decide that the topic is probably "the button bar". However the developers do not think there is any problem with the button bar. Mick requests a copy of the email.

M "I'll see if I can decipher [the email] because [the button bar] still works … he might be trying to do something entirely different."

P "It could be anything. Through what I've seen it could be anything."

D "Its not that it doesn't work, its what he's trying to do." …

M "If you don't put an image in, it just puts in a coloured button, and it just cycles round."

So it seems to be working. They discuss candidate 'mistaken use' or alternative senses to what the customer might be saying.

M "Unless he's using a stupid GIF like, that it can't load. A huge GIF or something. I mean I can't see [name of customer] making that mistake, I mean he's not daft. …In all fairness he's saying… you can't put a button on a toolbar. The only place you can put an image is on the system toolbar. I think that's what he's trying, or he's got a huge GIF file that he can't load."

This marks the end of the discussion of the toolbar. This issue will be noted down on a card and discussed later. However there are more suggestions in the email from this customer.

M (laughs) “I’m just reading in his email about the keypad. The email says that you guys should come out and see why these ‘picky things’ are in fact what will make this the finest application.”

They then talk about another customer who got excited about the system and was making a lot of suggestions.

M “He just couldn’t understand that a mobile device just didn’t have the processing power of a laptop.”

All three examples show the developers to have near-contact with customers, often but not exclusively mediated by the CRM (example one mentioned a customer email and example two mentioned support calls). Clearly, customer contact is a very important feature of development and while the developers may casually state a desire to be somewhat protected from too much direct contact they do regularly speak of and sometimes to users. The customer being discussed in this example is particularly vocal and despite not being someone paying a great deal of money, was valued for his enthusiasm for giving feedback. This example demonstrates that the developers take requests from customers seriously and that there is potential for customers to inspire requirements, but it seems the developers are under no illusion that “what this customer wants” is necessarily what he or she should be given. The developers sometimes laugh about their customers, especially their ability to make sensible well written suggestions, but the time spent in translating and working out the issues does seem valuable. We can see the developers deciphering a poorly written email, but assuming there is not just a presentation but also a translation issue here. They come up with likely translations, but these do not fit with the developers’ knowledge of what is and is not working in the system. It is suggested that it is an error of use rather than an error with the system, but this conclusion is also stalled as “the user isn’t that stupid”. Here we see combinations of knowledge about who specific users are, what they do, and how they might term things differently. In effect we see that if users are 'scenic features' of the development process, then they are scenic features that can move from the background to the foreground, from backstage to front-stage, from just 'users' (who don’t read manuals) to users with specific skills, experiences and known aptitudes ("he's not daft"). However, the topicality of a specific user, the problems they have and the kinds of things they worry about are fleeting and feed into developer’s discussions of what is and is not working in the software rather than what this user does or does not require.

6. USERS IN CONTEXT

“Through the timing, placing, pacing, and patterning of verbal interaction, organisational members actually constitute the organisation as a real and practical place. Furthermore, through a turn-by-turn analysis of organisational talk, it is possible to gain insight not only into how everyday business gets done at the level of talk, but also the interactional and organisational business that is accomplished through that talk.” (Boden [2], p15)

Our examples demonstrate some of the ways in which concerns can be raised or settled through various realisations of ‘the user’. As Boden would recognise, not only is the user spoken of in everyday business but is spoken of in talk that accomplishes that business. Advocates of the method XP clearly state the importance of talk, and therefore analyses which Boden suggest are important here. The examples show how, as Boden argues, problem solving is located in fine-grained, sequential organisational activities. Of particular relevance is the notion of 'local logics':

"As they sift through locally relevant possibilities ... social actors use their own agendas and understandings to produce ‘answers’ that are then fitted to ‘questions’." (Boden [2])

In the examples we document the contingencies of product development, the 'normal, natural' troubles whose 'usual' solution is, of a sort, ‘readily available’. Particularly visible in example two, usual solutions invoke horizons of tractability, containing candidate answers (seen before) and solutions (used-before-and-seen-to-work). Problems demand quick solutions, taking into consideration the present situation, the resources available, as well as any consequences:

"Caught in the pressing necessity of choice, organisational actors move through a fluid mix of problem identification, goal negotiation, solution seeking and decision-making." (Boden [2])

Such situated problem-solving results in fixes that may eventually become part of the repertoire of candidate solutions. And, as the extracts suggest, the boundaries between the types of problem are permeable and resolvable - for example through 'Google-ing'. Similarly, and unsurprisingly, different members view problems differently and this may lead to the resolution of the problem in, and through, the ability to improvise or recognise similarities with previous problems. The focus of this paper is on how the ‘user’ appears for and within such work.

6.1 Users as Typifications

In the examples, we have seen that the user can on occasion be such-and-such a person (eg. “[so-and-so] said in an email”), sometimes a more-or-less specific group of people (e.g. “our guys”), sometimes a more general category (e.g. “developer”), or sometimes a course-of-action category (“evaluator”) - i.e. their role as 'scenic features' can change. However, more often than not the user is ‘typified’ in the abstract (e.g. “they” or “people”). Whereas the developers studied by Sharrock & Anderson [29] had their contact with users mediated through report forms, the developers at our fieldsite have more direct links with (and no doubt far fewer and more readily available) users. While users may not be physically present or called upon in decision making at our study site, they do make themselves known and feature in discussions, even as ‘real’ people. However, different customers (and by extension users) are more valuable than others, likewise the future market is very valuable, and so certain users and their opinions and ideas are given more credence. Sharrock & Anderson cite economic and other practical factors as reasons for the diminished user role (diminished to ‘typifications’). We see it to be more inevitably so for product development; even when developers take greater interest in their customers and in figuring out and addressing their needs, those needs are inevitably cast against a general backdrop of other more or less important users.

Sharrock & Anderson [29] contrast the 'user as a scenic feature' with notions of user needs, requirements and evaluations being empirically assessed. At our fieldsite where users are more readily available, there is still no sure fire way of deciding that certain user feedback or that certain user requests are the ones that should be taken into account.

As stated, the vast majority of references to use of the system are made using abstract or bland (as opposed to colourful, or specifically drawn) types such as 'I' or 'people', and particularly 'you'. The generic 'you' is overridingly common in talk about functionality (for example "you can still click on it" or "you'd rather expect it"). This makes much of the functionality appear as common-sense for users and developers alike. In the final sequence of example one, we firstly see 'you' as referring to those developers present in the room, and then in the following line as a generic for users of the software. The second usage is the kind we are interested in and reflects, as Sacks describes, that 'you' is rarely said to refer to a single person but more-so in the plural as "a way of talking about everybody, and indeed, incidentally of me" (Sacks [26], p166); so here 'you' is a plural for 'every user and incidentally me'. In this case Paul is discussing normal work for programmers: "you can still click on it". This is one of many cases of references to the user-in-general where no obvious distinction is made between the developers of the software product and its users.

We see our third sequence of example one to contain an interesting switch from a general category 'you' to a third-person category 'people'. There is a generic imperative "you've got to read the manual", followed by a generic statement of fact in the third person "people don't read the manual". This switch to the third person can be regularly seen in conversations where the developers talk about what might be seen as bad practice. The switch to the third person is done when the developers wish to distance users' ways of working from their own, but note that this is no more a claim about their users being 'dummies' as it is an artful means for forming an argument (of the form "yes, but you don't understand the users"). The repost "well people don't read error messages either" maintains the third person category "people" and in so doing beats the previous statement at its own game; perhaps this could have been achieved in no other way.

The final sequence of example two demonstrates the negotiability of categories for users. Paul poses that "you wouldn't develop in a VM, and our guys are developers". This utterance seems to set up a certain equivalence or affinity between the developers and their users. Unlike in our previous case, the switch between categories is done here by the same person in the same utterance. However, as the talk progresses, we see Dale evaluating the categories given by Paul initially as "well supposedly", which leads to Paul changing his stance to agreement with Dale before Dale himself reformulates to "too harsh!". This is an example of both the fluidity of typifications and the developers' abilities and compulsions to negotiate, restate and evaluate them between themselves; jokiness aside, users are 'like us but just not too much'. We note the use of 'guys' here is the kind of 'gendered language' discussed by Cockburn & Ormrod [9], but to follow their arguments through would be to assume that 'guys' mapped to some sort of list of users the developers 'have in mind', a position we reject (which is not to reject inequalities in development).

6.2 Users as Courses of Action

Thus far our discussion has covered some general sets of common-sense actions for software use, but we can also point to instances where work practice or 'courses-of-action categories' (Sacks [26]) become a more substantiated feature of conversation. The developers discuss and tie in their decisions to the work practices of users in multiple and interesting ways. Firstly, we can see an account of 'working' related to what users do. The first sequence of example one contains "I would want it working for the initial play". This not only demonstrates knowledge about what people do on first encountering the software (they 'play' with it), but also gives an interesting account of 'working'. Working is used here to relate to something that does not have the user stop and think, read error messages or get the wrong impression. Working here then is strongly tied in with user practice, rather than into a technological sense of reliability. A concern for work practice is also apparent in example three where the developers question the work practices of a user in terms of whether the system, or the use of the system is faulty. Even more specific examples can be found although these tend to be where the customer is some special case, the following concerns a large organization they are hopeful to get as a customer:

P "With VehicleRepairCo, if you have to drive 30mins to where you've got to get then its no problem to wait five more seconds."

The above example is about a particular customer but draws upon general knowledge of what employees of that customer do, further informed by the knowledge of how that company would deploy applications built with IDEco's system 'in the field'. This knowledge may turn out to be wrong or not relevant but is good enough for their work at the time (figuring out worst acceptable cases for performance testing a high-load server).

6.3 Users as Timely Arrivals

As we have been arguing, users are contextual concern, brought into discussion 'as' and 'when' required. In this section we discuss 'when' they are required.

Regarding example two, it is in one sense fortunate and another convenient that in finding that they can't 'solve' an issue the developers satisfy themselves that it is acceptable not to. For the evaluative question of whether a problem matters, the user (to borrow from Sacks [26]) 'appears on cue'. This is not to say the developers are lazy and invoke the user to dodge the difficult work; we have been noticing the opposite whereby the developers seem to have a spirited curiosity towards technologies. This curiosity is useful, as it means that they are often concerned with understanding their application better, its limits, its problems and the possibilities for future development, all of which seems to be important in developing a good quality product. This however does not entail an 'aesthetic or objective' rather than 'pragmatic' orientation: we see the user often appears as a pragmatic justification for work being of aesthetic or objective quality. We too see the user in justifications for decisions that could be said to be un-aesthetic (although these are perhaps more controversial decisions). In example one the repost "until after they've purchased it" to "you are just postponing the issues" is clever as, through agreeing with and then diminishing the issue, Shaun is able to stifle further argument for an aesthetic design. Shaun does not imply that everyone will purchase the product but draws from

the idea that the customer will make their decision early. This idea is not contested. It also implies that problems post purchase have some acceptability and can be dealt with in a different way. If you do not sell the system you do not get anywhere.

Hand in hand with issues of justification are methods of measurement; of how the developers trade off the work to be done with the value of that work to the user and to the company (what Garfinkel [11] calls 'the administrator's problem'). In example two the growing complexity of a bug overshadowed the low possibility it would be noticed. Elsewhere we often see the amount of work to be done on something measured against what the users 'worry' about:

M "We might do a lot of work that no one would worry about."

"We might do a lot of work" refers to the likelihood that this is difficult rather than the knowledge and again "no one would worry" seems different to "no one wants" or "they don't". We see a kind of openness here about the possibilities of work and use, a guess that is good enough for now and means that we don't have to spend time on this and can do something else.

What we come to is views of the 'customer' and 'user' that are produced for specific (and varied) occasions, conversationally amongst the group as an aid in understanding problems and reasoning about development considerations. There is fluidity in the use of these types, which are enrolled, negotiated and dispatched according to the problems the programmers are facing then and there. No user type trumps another automatically but has value with respect to the work at hand.

Development tasks are defined and scheduled taking into account customers, users, internally generated ideas, market judgments and so forth. However smaller requirements and mundane design decisions are made routinely during development, coding being a praxiological and satisficing concern (Kristofferson [18], Button & Sharrock [6,7]). Sometimes these small design decisions occasion discussion, and a number of these invoke customers, users and user practices. However, we also need to acknowledge that in the constraints of development many 'decisions' are simply not discussed, or discussed in relation to the user, or even understood as being a 'decision'. It became clear in our interactions with the programmers that, the fact that the development could have progressed differently is not a matter that they spend much time talking about.

6.4 Users as Different to Customers

So what is the difference between user and customer? Most differentiating between customer and user seems to arise in discussions and disputes between the developers and the CRM and technical director. It is clear that the developers most readily identify with and care for the end users, whilst the customer relationship manager deals with both users and customers, and as such has an orientation to the sale-ability as well as the use-ability of the product. The first example showed the 'potential buyer' being invoked as more immediately important than users as a way of settling an argument. Distinguishing between the two during development seemed rarely necessary for developers (probably because much of their orientation is directed un-problematically towards users). In one case a 'high paying' customer who had lodged a feature request was given special care, and in another a non-paying user was ignored. Both of these actions were at the instruction of the CRM however. Care was also taken, again at the CRM's request, over getting the software into a shape that would be suitable for demo-ing to large corporations (such as the company 'VehicleRepairCo' briefly mentioned earlier).

7. DISCUSSION

We have focused upon how ideas about the user (who they are, what they need and what they will get) feature in the creative process of software product development and testing. We have focused on talk between developers and found that in such talk:

- The 'user' and 'customer' are regularly brought into discussions between programmers when the design is problematic
- The 'user' and 'customer' are not simply relevant to the interface and ways of working with the product. They are relevant in economic terms.
- Users and customers are often talked about in general terms such as 'we' or 'people', but specific users or kinds of users can be foregrounded when problems and disputes arise.
- Categories for users are fluid, they are negotiable and may change or be refined over the course of a conversation. The significance of any category is relative to the work at hand, and can be contestable.
- Any users and customers brought into discussion are cast against a background of other users. For the most part, single users are only interesting in terms of how they relate to a general set of users.
- There is an interest in what users do, but only to the extent of what they do 'in general'. The general 'you' user appears to justify aesthetic decisions, the more specific categories appear to justify more pragmatic decisions.
- User requirements are often traded off against the amount of work it is thought it will take to satisfy that requirement. The value of a requirement is in terms of its generality and how it will expand the market.
- Customers may know their requirements and what is technically feasible but they are unlikely to understand the market in which their requirements will be framed.
- The customer and user get differentiated where financial concerns are relevant.

These findings arise from a study of the group work of software product development using XP. XP is a method that encourages and relies upon conversation and close cooperation within development. Given these findings we suggest, in the case in using XP for product development, that users are a *contextual* concern to development work. That is, knowledge about users is formulated and brought forward in ways that shape and enable development work to be done.

This paper has contributed to the longstanding concern in Software Engineering with 'users' by drawing on hitherto overlooked or perhaps misinterpreted work by Sharrock & Anderson [29], Woolgar [33,34], and others. Although ours is a study of developers using an agile rather than plan driven method we have covered many similarities between Woolgar's and

Sharrock & Anderson's findings and our own. Although the (mis)use of XP for product design does enable new ways of organizing practices, it seems many of these practices remain similar or the same. As we discussed earlier, the kinds of user involvement asked for in XP are not possible for product design. We also note that Button & Sharrock [6] claim that software engineering methods do not underlie and generate particular practices, but following and implementing a method is a part of practice. There are, of course, several key differences between Woolgar's and Sharrock & Anderson's work and our own: Firstly, we note the dynamic and important relationship between notions of 'user' and notions of 'customer' in XP. Furthermore, for this small enterprise the way the 'user' serves as context for discussion and decision making seems more complex and dynamic than described by Woolgar and by Sharrock & Anderson in their studies of multinational organizations. For them the contextual user was largely an imagined future user in another place and time, of a *'me'* or *'everyman'* construction. In our case, while these constructions are still apparent, the 'users' and 'customers' are more proximal and 'real'; Developers do know some of these people, either directly, through email, or through the CRM, and they know something about their businesses and so forth. That they have more direct knowledge of users and customers is possible because they have a limited customer base. This is a small company but one that is trying to expand its market. As reported by Pollock et al [25], as a product develops the company may have to develop a more 'organised' set of procedures for dealing with customers and users, holding some in positions of greater 'privilege' or 'usefulness', and it is worth noting that some previously 'revered' users or 'friends of the company' may have to be cast aside as the product moves in different directions.

So how might procedures for dealing with customers and users become more organized? This, it seems to us, is a key challenge for CSCW and related areas. Much CSCW literature concentrates on design for singular situations or for a reasonably generic or consensual user base where it is theoretically possible to identify a clearer set of requirements that mesh with local, situated work practices. When producing a generic product, such work is not always relevant. Product developers seek requirements that apply across a number of user sites. Some site-specific requirements will be addressed for a number of reasons (ease of implementation, importance of customer, etc.) but ones deemed 'idiosyncratic' will not. Furthermore, software products are often designed and sold as instruments of change. In the work of Woolgar [33,34] we can see that product design involves achieving a balance between meeting multifarious needs and desires while also *restricting* the flexibility of the product, pushing the users down a certain set of agreed upon paths of interaction and use. In product design, as Pollock et al [25] discuss: in order for the product to remain viable for a wide market it cannot be tailored to the specificities of all its consumers; product developers try to get customers to tailor their requirements and processes to their product (at least as much as they tailor the product to their consumers); and certain customers and their requirements inevitably gain preference because they suit the developers and their vision, and 'fit' with the product. The business of good product development is getting the right balance in designing (specifically) for (some of) your (particular) users while reaching out to an ever more diverse customer base. We therefore strongly suggest that product development should be treated as having separate concerns to those of bespoke design. Grudin and Pruitt [14] outline the problems of participatory design and scenario based design for product development and suggest instead persona based design. Whilst techniques such as persona based design may be useful for products designed (in multinational companies) to suit, as Grudin and Pruitt term it, "millions of users", we find that participatory, persona and scenario based design approaches all fail to properly appreciate the practical and economic contingencies that impinge upon and shape the construction and evolution of a software product for application development such as the one discussed here.

The issues we have identified are associated with XP for product development. The developers we have studied do not attempt to do 'pure' or 'textbook' XP and therefore our descriptions of their practices cannot readily serve as criticisms of the method itself. We have not found evidence for Stephens and Rosenberg's [31] claim that a problem with implementing one aspect of XP will cause the whole method to collapse. We do not discount XP as a viable approach for product development. When we look at the kind of risks involved for this company we would argue that these are not simply caused by either the use (or misuse) of XP or the approach to involving users and customers in development. Indeed, in the end this company may prosper or fail like many other small software houses with a good idea and a malleable market. We must be careful not to, as Suchman criticizes of Woolgar, produce an "overestimation of the ways and extents to which definitions of users and use are inscribed into an artifact" ([32, p192]) and understand that unlike Woolgar we are looking at iterative development that can to some extent improve, correct and change designs over time. Despite the problems of user participation, for many reasons XP and other agile methods seem a reasonable approach to development. As Boehm and Turner [3] discuss, the choice of method cannot be made because one is simply better than the other. Boehm and Turner argue that the suitability of a method should be seen in terms of the particular risks faced in the development in which that method is to be used. The company is small, the project is manageable without the need for a lot of documentation, XP allows them to be responsive to changing requirements, and they are developers, developing a product to be used by other developers. The market does not expect (although it might like) a polished final product, instead it has an evolving system. Given that many smaller product companies like the one discussed in this paper will continue to want to use agile methods, further work in this area can be useful in articulating the particular contingencies of 'generic' product development and to aid in minimising the risks associated with bad decisions concerning product directions, users and customers.

8. CONCLUSION

In the use of XP (Extreme Programming) for software product development customers and users cannot participate as they might in other uses of XP, but are treated generically. This generic treatment features in development work as typifications of users and typifications of use. Such typifications are produced, negotiated, refined and occasioned with respect to the work at hand; that is, typifications are 'contextual'.

9. REFERENCES

[1] Beck, K. 2000. Extreme Programming Explained, Embrace Change Addison Wesley, Boston.

[2] Boden, D. 1994. The Business of Talk: Organisations in Action Polity Press, Cambridge.

[3] Boehm, B., and Turner, R. 2004. Balancing Agility and Discipline Addison Wesley, Boston.

[4] Bowers, J., and Martin, D. 2003. 'Making the Organisation Come Alive: Talking Through and About the Technology in Remote Banking' Human Computer Interaction vol.18 no.1-2, 111-148.

[5] Bowers, J., and Pycock, J. 1993. 'You and Whose Army? Or Requirements, Rhetoric and Resistance in Co-operative Prototyping' SIGOIS Bulletin 14, 40-45.

[6] Button, G., and Sharrock, W. 1992. 'Occasioned Practices in the Work of Software Engineers' IN Jirotka, M. and Goguen, J. (eds.) Requirements Analysis: Social and Technical Issues Academic Press, London. 217-240.

[7] Button, G., and Sharrock, W. 1998. 'The Organisational Accountability of Technological Work' Social Studies of Science vol.28 no.1, 78-102.

[8] Chong, J., and Hurlbutt, T. 2007. The Social Dynamics of Pair Programming. Proc. ICSE, (May 20-26 2007, Minneapolis MN, USA), .354-363.

[9] Cockburn, C., and Ormrod, S. 1993. Gendered Technology in the Making Sage, London.

[10] DeSouza, C., Redmiles, D., and Dourish, P. 2003. '"Breaking the Code" Moving Between Private and Public Work in Collaborative Software Development' Proc. SIGGROUP, November 09-12 2003, Sanibel Island, Florida, USA, 105-114.

[11] Garfinkel, H. 1967. Studies in Ethnomethodology Prentice-Hall, Englewood Cliffs.

[12] Garfinkel, H., Sacks, H. 1970. 'On Formal Structures of Practical Actions' IN McKinney, J., and Tiryakian, E. (eds.) Theoretical Sociology ACC, New York, 337-366.

[13] Grint, K., & Woolgar, S. 1997. The Machine at Work. Technology Work and Organisation Polity Press, Cambridge.

[14] Grudin, J., & Pruitt, J. 2002. Personas, participatory. design, and product development: An infrastructure for. engagement. Proc. PDC, (June 23-25 2002, Malmö, Sweden), 144-161.

[15] Heath, C., Sanchez Svensson, M., Hindmarsh, J., Luff, P., & Vom Lem, D. 2002. Configuring Awareness. Computer Supported Cooperative Work vol.11, 317-347.

[16] Hunt, J., Romero, P., & Good, J. 2006. 'Stories from the Mobile Workplace. An Emerging Narrative Ethnography' Proc. PPIG, (September 7-8 2006, Brighton, UK), 153-167.

[17] Jefferies, R. 2007. www.xprogramming.com (accessed March 2007)

[18] Kristoffersen, S. 2006. 'Designing a Program Programming the Design' TeamEthno-online 2, June 2006, 34-51.

[19] Lindsay, C. 2003. 'From the Shadows: Users as Designers, Producers, Marketers, Distributers and Technical Support' IN Oudshoorn, H. and Pinch, T. (eds.) How Users Matter: The Co-Construction of Users and Technologies MIT Press, Cambridge MA, 29–50.

[20] Lippert, M., Roock, S., and Wolf, H. 2002. Extreme Programming in Action. Practical Examples from Real World Projects John Wiley and Sons, New York.

[21] Lynch, M. 1995. 'Preliminary Notes on Judges' Work: The Judge as a Constituent of Courtroom Hearings' IN Travers, M., and Manzo, J. (eds.) Law in Action. Ethnomethodological and Conversational Analytic Approaches to the Law Ashgate, Dartmouth, 99-129.

[22] Mackenzie, A., Monk, S. 2004. 'From Cards to Code: How Extreme Programming Re-Embodies Programming as a Collective Practice' Computer Supported Cooperative Work Vol.13 No.1 91-117.

[23] Mackay, H., Carne, C., Beynon-Davies, P., & Tudhope, D. 2000. 'Reconfiguring the User. Using Rapid Application Development' Social Studies of Science Vol.30 No.5 737-757.

[24] Nardi, B. 1993. A Small Matter of Programming. Perspectives on End User Computing MIT Press, Cambridge MA.

[25] Pollock, N., Williams, R., and D'Adderio, L. 2007. Global Software and its Provenance: Generification Work in the Production of Organizational Software Packages. Social Studies of Science Vol.37 No.2 254-280.

[26] Sacks, H. 1995. Lectures on Conversation (Volumes 1 and 2). Blackwell, Malden.

[27] Schutz, A. 1964. Collected Papers (Volume 2) Studies in Social Theory Nijhoff, The Hague.

[28] Sharrock, W. and Anderson, B. 1993. 'Working Towards Agreement' IN Button, G. (ed) Technology in Working Order Routledge, London, 149-161.

[29] Sharrock, W., and Anderson, B. 1994. 'The User as a Scenic Feature of Design Space' Design Studies Vol.15 No.1. .5-18.

[30] Sharp, H., and Robinson, H. 2004. 'An Ethnographic Study of XP Practice' Empirical Software Engineering Vol.9 353-375.

[31] Stephens, M., and Rosenberg, D. 2003. *Extreme* Programming Refactored: The Case Against XP APress, New York.

[32] Suchman, L. 2007. Human-Machine Reconfigurations: Plans and Situated Actions 2nd Edition Cambridge University Press.

[33] Woolgar, S. 1991. 'Configuring the User, The Case of Usability Trials' IN Law, J. (ed.) A Sociology of Monsters. Essays on Power Technology and Domination Routledge, London, 58-100.

[34] Woolgar, S. 1994. 'Rethinking Requirements Analysis: Some Implications of Recent Research into Producer Consumer Relationships in IT Development' IN Jirotka, M. and Goguen, J. (eds.) Requirements Analysis: Social and Technical Issues Academic, London, 201-216.