Cooperative Work in Software Testing

David Martin[1], John Rooksby[2], Mark Rouncefield[2], Ian Sommerville[3]

[1]*XRCE, Grenoble. France.*

[2]*Computing Department, Lancaster University. UK.*

[3]*School of Computer Science, University of St Andrews. UK.*

david.martin@xrce.xerox.com, [j.rooksby, m.rouncefield]@lancaster.ac.uk, ifs@dcs.st-and.ac.uk

ABSTRACT

Substantial effort in the development of any large system is invested in testing. Studies of testing tend to be either technical or concerned with the cognitive ability of testers. Our experience is that testing is not technical but socio-technical, involving a great deal of human and organisational effort, and that testing is not simply the kind of decontextualised 'puzzle solving' many cognitive approaches imply. We believe that cooperative work is foundational to getting testing done. In this position paper, we use data from four ethnographic studies to discuss just what that cooperative work is.

Categories and Subject Descriptors

D.2.5 [**Testing and Debugging**]

General Terms

Human Factors

Keywords

Ethnography, Testing, Cooperative Work.

1. INTRODUCTION

This paper presents examples of testing as it is done "in the wild". Such examples highlight the cooperative and human aspects of testing, aspects that are routinely overlooked in testing research but which we believe are foundational to 'getting testing done'. In the light of a number of transformations to software testing practice over recent years, we believe it is increasingly perilous to overlook these human and cooperative aspects. Transformations include: 1) a shift in focus from programs to systems; 2) 'usefulness' becoming relevant in testing alongside or in place of correctness; 3) iterative development and reduced time to market entailing issues being knowingly left until post-deployment; 4) testing increasingly becoming a professional, team activity and made accountable to the wider organization; and 5) technology transfer from research entailing the reorganisation of current practices and the acquisition of new skills.

CHASE'08, May 13, 2008, Leipzig, Germany.

2. EXAMPLES

We have written an example driven paper as we believe a ground-up approach provides necessary contrast to the theory and technology driven work that dominates testing research. Based upon our examples we will discuss four ways in which we believe socio-technical approaches can address practice relevant issues. Our examples are taken from ethnographic fieldwork, undertaken at four fieldsites in the UK: (1) A small, agile, software house producing an IDE, at which we spent 30 days over one year; (2) a large organisation developing an administration system in-house, with which we have spent 38 days over 8 months (ongoing); (3) development of scientific software by a professional programmer based in a University with whom we spent one week and (4) the distributed development of open source software, fieldwork on which is progressing. Fieldsite two is the only site with professional testers. Fieldsite one has scheduled testing phases, with testing carried out by programmers. Fieldsites three and four rely on regular regression testing, with other tests undertaken on an informal, often opportunistic basis. We make no claims these fieldsites represent 'best practice', but claim they encounter common issues in testing and go about work in common ways. We present our data thematically, beginning with a discussion of how tests are cooperatively scoped.

2.1 Scoping Testing Cooperatively

During our fieldwork, issues such as the responsibilities of testers, what tests could realistically be done in the allotted time, whether testing and training could be done simultaneously, and whether testing could be done with real or simulated tasks arose repeatedly. The following example is from a user acceptance test at fieldsite 2:

There are four 'users' testing the system. Barry (one of the users) yawns. Laura (another user) says jokingly "For goodness sake Barry!" Barry replies "It's just hard to take in … I need to look through it … I can't [keep concentrating on the screen]." The testers and users discuss whether it is possible for the users to have a look at aspects of the system at another time. They discuss what aspects could be looked at later, where and how this could be done and the time by which it is required to get them done.

This 'mundane' example is packed with relevant detail; it shows scope to be a problem that can repeatedly arise. This is a planned session, during which it arises that the planned scope might not be met. Planning and estimation of what can be achieved is an error prone art, we do not think the above can be put down to bad planning as we see that a scope achievable with one user (Laura) is not with another (Barry); one simply has more stamina than the

other. The possibility of reducing the workload by moving some of the work to another time and place is discussed. This re-scoping, as was the case with the initial scoping, is done with reference to some practical organizational considerations: what, where, how etc.. This, and all our examples, highlight the thoroughly practical, situated nature of getting testing done. We continue with two vignettes from fieldsite 1. The first example is a jibe made by one programmer about another.

Mick makes a little jibe at Tom: "Not all of us are doing testing". The jibe is said towards Paul and Dale, but a gesture towards Tom makes it clear who he is talking about. The programmers are in a 'test phase' but Tom is still writing code.

Coordination of effort is an important issue in testing. The jibe would seem to assume that Mick thinks Tom knows full well that what he is doing is incompatible with the purposes of software testing. Tom is publically shamed rather than instructed in correct procedure. The next, similar, vignette is also from fieldsite 1.

Mick jokes about the pile of pink cards produced by Dale "bloody hell, we're testing, not finding faults!". Mick goes through the cards: "so what!" "Don't care!" "no!" "none of them!". He says "Dale is making trouble!". Paul asks Dale "So you've found three other faults except for the defects?". Mick asks "What are they?" Dale says angrily "You just read them!" Mick, referring to Dale's handwriting, replies angrily "I couldn't actually read them! No one can read them!"

Finding faults is disputed here as being an appropriate task for the testing being done. However, faults must be taken seriously, the joking "so what!" etc., would not work without it, and Paul is interested to know what the faults are. As with the previous vignette from fieldsite 1, there is no dispute over whether work is being done, but the timing of that work is called into question. Although the period of time allocated to testing at fieldsite 1 is predetermined, what substantiates that phase is partly emergent through how people organize themselves and each other during that phase. Testing is replete with negotiation, with regulation, and ultimately ongoing definitions of what is and is not the 'correct' thing to do during the testing phase. These issues are compounded by the fact that there simply isn't much time to do testing, and also that those doing testing, if they are to do an effective job in the time available must put some effort into coordinating their work with others. Amongst other things, this coordination involves doing agreed things during agreed times, and communicating with others (ideally without confrontation, and with legible handwriting). What should also be apparent from these examples is that testing can be emotional, involving joking, confrontation, etc.. Programmers at fieldsite 1 sometimes spoke of the boredom of testing and even that people who enjoy testing are "freaks". We did not encounter similar attitudes at the other fieldsites, but emotions could sometimes run high.

2.2 Making Use of Available Resources

This next example is again taken from fieldsite 1. In this example they are testing something called "the push server". This push server should, in short, send a single message to multiple mobile devices. The push server was required by a customer who the programmers estimated to have around 1000 mobile device users. The programmers also recognized that they would hopefully soon have bigger customers who could potentially make use of a push server. The test is carried out by each of the four programmers plus the customer relationship manager installing software on their machines that simulates multiple connections to the server. They start by testing the server with 5000 messages, with 1000 connections simulated from each machine.

Pete remarks "5000 – Amazing! Now I'm going to send one message to all 5000 back". As the messages get sent back the programmers comment on them coming through. This is successful, so after a joke about testing being finished they decide to double the amount of messages. Pete says "I'm thinking about trying 10000, so we have to change to 2000 messages each … I think my machine will potentially [fail] with 10000 sockets, we need to change our offsets. Double them both, all of you."

So why did this test start with 5000 messages? Firstly, this number is divisible by 5; there are 5 of them testing. It should be a rather obvious point, but is nevertheless worth emphasizing, that tests and how something is tested draws from the resources at hand. These resources include who is available to run the tests. Additionally, 5000 is a number that is an order of magnitude greater than the 1000 connections they believe their customer may peak at. Therefore it is 'big', but, it is also not too big: running these sorts of tests can take time and it is better if an error can be found quickly. At this company, the user manual is written during the test phase. The tests serve not just as tests but as exemplars of how to use the system. The tests are written up in the manual as demonstrating how to use the system; again a sensibly sized example that does not break the system is necessary. Testing can be "to look for errors" as Myers [1] famously recommended, but we find sometimes this is sensible only after the software is shown to work, something that Myers specifically does not recommend. We can see that whilst trying to break software is, in theory, a sensible approach to testing, in practice it is sometimes helpful and sometimes not helpful to do so.

2.3 Failure

There are two parts to this section. This first part is taken from fieldsite 4. It is from an interview with a programmer working on the distributed, open source project. In this interview the programmer discusses the different mailing lists belonging to the project, and shows the lists to the interviewer as he does so:

"So the regressions mailing list tells you about, erm, whether things are passing or failing regression tests. So if I put something into the repository and it breaks everything. Then, I'll know here [in the email]. But this is typical [scrolling through email], erm, we're breaking lots of things. But they've actually been broken for months and months and months [laughing]. So you just get to the, you just get used to the fact that you fail 77 tests at the moment, and then normally it gets better, err, sometimes it gets massively worse, but then you kind of get told by the steering committee "don't worry too much about it". Heh heh. We're going to fix this eventually. … People will follow things in the regressions mailing list saying "this failed for this reason" or, "we need to look at this" or something."

In this second vignette the same programmer tells us about a new development in the mailing lists.

"So we ran this many tests, we failed this many. So erm one of the things that's being changed at the moment is … He's changed it so that erm, it says how many we passed and how many we

failed but it tells you what are new failures and what are new successes. … with colour and, so you can tell a bit more information then previously. .. so I mean, this is kind of the thing that helps the development community."

Testing research is often focused upon correctness, we find some irony in being told the thing that helps the development community is software to help overlook long-term failures. This requirement to overlook failures and the very means with which failures are presented are in most respects dependent on the way this project is organized. That is, a fix is not always urgent in this project and is sometimes best left to or coordinated with others. Conversely, at fieldsite 2 they were keen to keep and present to managers a low average fix time. At fieldsites 1 and 3 bugs were sometimes left until a user 'noticed'. An example of this from fieldsite 1 follows.

We observed the programmers spending a great deal of time trying to figure out why their software would not work well in a virtual machine (VM), and what they might do to resolve this. The programmers talked through and explored a variety of possible causes and solutions but were unable to find an answer. It was then that they discussed the 'value' of coming up with a solution:

Paul says "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." Soon after, Mick says "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."

It is seems common that after exhausting the possibility of a quick or obvious fix that the value of actually doing any fix is discussed. Deciding not to fix this problem but to record it as a 'known issue' is not laziness on the programmers' part. The effort to be put into solving this is unlikely to be worth it. The value of a fix is usually decided in talk according to the things the testers figure out as relevant and is not the subject of any numeric calculation. We find that value is "calculated discursively". However, in such calculations, the developers do not usually say anything for sure; here they are interested to see if they have calls "coming in". This is a feature of iterative development: decisions can turn out to be wrong and changed later.

2.4 Knowing the 'Users'

A substantial part of testing is concerned with users. Whilst a focus on 'users' is most readily associated with usability and user acceptance testing, we have found that in all forms of testing, ideas about users regularly arise in deciding what kinds of tests are necessary and the implications of a particular test result. In the last example we saw 'users' appearing in a justification for doing a fix. In this example, taken from fieldsite 3, the vignette concerns the programmer (Max) discussing the user interface with a scientist (Alex).

Max asks Alex "Why do you not like the bold? Do you not like the bold?" Alex replies "Its just a bit ambiguous I think. If I'm just coming into, I mean I'm trying to, imagine myself just being, using this for the first time" Max "yeah". Alex "It it could be, it could be a lot of different things". Max "err, Yeah" Alex "And, also because, imports for me is kind of a, a tertiary function, that you learn to use a bit later on" Max "Mm Mm, Mm Mm".

In this example we see Alex, the participant user, put himself in the shoes of somebody using this for the first time (i.e. "…imagine myself just being…"). Then we see him talking generally about what users do (i.e. "… you learn to …"). This is a phenomenon we have noticed with product development, that users are always set in the context of other users. Developers do this, and as we see, participant users also do this. Although ideas of what the user wants and will do are very important in testing, it seems that there is rarely a definite idea of who the user is; whilst some users can be spoken about with a high degree of certainty, and in cases such as in this vignette, even spoken to, other, vaguer ideas about other more vaguely defined users are never far away. The user is important in settling issues in testing, but 'user' is often not just a category referring to a definite person but also encompasses social and organizational knowledge about who users are or might be. Developers therefore are regularly required to fall back on practical social reasoning.

2.5 What Not to Test

We have observed decisions both to do with whether a particular test is done or not, and to do with levels of coverage or detail to which a test goes. Decisions not to test are often accountable, that is reasons and justifications are given for any particular course of action or non-action. Our example in this section comes from fieldsite 2. In this example we summarise how aspects of load testing are organised.

A proposal to the project board regarding the first go-live phase reads "The approach has been discussed with [the infrastructure department] and the preferred option is to carry out an in-house load test without support from the external supplier". Elsewhere the document reads "A decision on the extent of load testing for [Phase 2] will be made following testing for [Phase 1]." A board member says it "makes sense". Later, another asks "Is [a test in phase 1] the best use of your time … for a massively over-provisioned infrastructure?" After much discussion, the board decide to take a consultant's offer to do, for now, a technical audit of the COTS system regarding load.

Documents are very important in the work of fieldsite 2. These are not standalone information but are constituent in ongoing discussions and decision making. A number of discussions preceded the writing of this proposal. The infrastructure at this organization had been load tested recently and a general outcome from discussions was that phase 1 of the project was unlikely to increase load. Not doing the tests for now is a risk 'calculated' over time though ongoing discussions in meetings, corridors, emails and in documents. This risk is set in terms of resources (i.e. "is this the best use of your time"). The decision to load test and the extent to which this will be done is delayed, not dismissed. The choice might turn out to be wrong, and irrespective of that, might be different to choices made later as conditions clarify.

Tests are discussed in terms of importance or associated risk and scoped, scheduled or delayed (perhaps indefinitely delayed) accordingly. In section 2.3 we saw the developers at fieldsite 1 acknowledging that a problem that does not seem worth addressing now might well become worth addressing at some indeterminable time in the future. It seems to be the case with

continuous or iterative development that aspects of development work that are seen as unimportant, including the undertaking of various tests, are put off rather than dismissed.

3. DISCUSSION

"'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 and Anderson [2] p.161)

Software testing problems appear, from our empirical evidence, to be set within the range of mundane organizational issues recognized by Sharrock and Anderson. The overwhelming feature of the everyday, mundane reality of "getting it done" seems to be how to deploy limited testing resources to find faults or design problems, or to see if the system operates in the desired manner and meets customer needs. Testing, we have seen, involves finding satisficing solutions, often to ill defined problems. We use this term "satisficing" not to refer to any laziness or making easy on the testers' part but, following Simon (below), to refer to getting useful work done in organizational contexts.

"In the face of real-world complexity, the business firm turns to procedures that find good enough answers to questions whose best answers are unknowable. Because real-world optimization, with or without computers, is impossible, the real economic actor is in fact a satisficer, a person who accepts "good enough" alternatives, not because less is preferred to more but because there is no choice." (Simon [3] p28)

To borrow Simon's terms, in the face of real world complexity, the tester becomes a satisficer. We see this complexity in terms of the everyday but nevertheless important, organizational issues Sharrock and Anderson describe. These issues are of human and cooperative work. Building upon our examples, we finish this paper with a discussion of four directions we believe socio-technical research into testing might usefully purse: rationale, organisation, resource, and time.

In our studies, we have noticed a great deal of thought, discussion, documentation, argument, etc, being put into the reasons for doing or not doing a test. Testing is done in a way that is sensible to and practicable for the system being developed, or is not-done with a reason relevant to that software. As systems develop within a project, as understandings develop, and as the projects themselves develop, the tests and the reasons for doing or not doing a test also develop. We think more attention may be paid to the rationale behind the tests particular organizations do. Actually keeping track of this rationale, we believe, could sometimes be useful. Whether we feel testing is done with the right reasons or the wrong reasons it is important to attend to what those reasons might be and how they are important.

We have noticed there are strong organisational demands on testing, and testing is shaped and scheduled to suit the work of the organisation. Firstly we have seen how economic concerns, insomuch as a concern for where the money is coming from and what is it paying for, pervade testing. We have seen how efforts to figure out who users are and what they might want are related to these concerns, and related also to practical design decisions. We have also discussed how testers must supply effort into staying coordinated with other testers and also with people working in or with the wider organization. Software testers it seems put much effort into staying organized and into doing work that is seen as relevant and productive for their organization at large. Organisation, here, can therefore be read as both a noun and a verb.

We have claimed that testers are routinely trying to use limited resources in the most effective way. What are those resources? Of course it is possible to speak of time and money as a resource, but we have also found that testing involves working with and around resources local (e.g. Who is available, what equipment is to hand, how the workplace is arranged) to the testing work being done. Not only that, but a great deal of effort is placed into getting appropriate resources there at the right time. Testing is done by and with people, in rooms people can get to, using and working around the artefacts and equipment in place. When the "economics of testing" is referred to we would do well to remember this encompasses practical matters such as "how much time?", "who will do it?", "who might care?" etc..

Finally, what is perhaps the prime problem facing testers, that there just isn't enough time, seems to stem from the fact that other work is also being done. There is the development work that seems almost inevitably to overrun and slip into the time allocated for testing. Secondly, during testing we have noticed that testers are rarely just testing. We have noticed that this is when the manual gets written, that this is when users get trained, that this is when users see the software for the first time and generate a whole raft of new requirements, that this is where wider research is prompted into understanding exactly why something is some way or exactly how something works, and that this is when tests are done alongside other tests. To treat testing as some sort of puzzle solving activity doesn't seem right to us because it seems hard for testers to focus on one thing. Testers must balance their testing with other work going on and either organize their testing to suit this other work, or organize other work (often battle other work) to suit testing.

4. REFERENCES

[1] Myers, G.J. 1976. The Art of Software Testing. John Wiley & Sons, New York.

[2] Sharrock, W., Anderson, B. 1993. Working Towards Agreement. In Technology in Working Order, G. Button, Ed., Routledge, London. 149-161.

[3] Simon, H.A. 1969. The Sciences of the Artificial. MIT Press, Cambridge MA.