Uncorrected proof. To appear in the Journal of Computer Supported Cooperative Work

Testing in the Wild: The Social and Organisational Dimensions of Real World Practice

John Rooksby[1], Mark Rouncefield[2], Ian Sommerville[1]

[1]University of St Andrews, UK. [2]Lancaster University, UK.

Address for Correspondence:

John Rooksby.
School of Computer Science,
Jack Cole Building,
North Haugh,
St Andrews.
KY16 9SX. UK.

Email: jr@cs.st-andrews.ac.uk
Tel: +44 (0) 1334 46 3268

Abstract

Testing is a key part of any systems engineering project. There is an extensive literature on testing, but very little that focuses on how testing is carried out in real-world circumstances. This is partly because current practices are often seen as unsophisticated and ineffective. We believe that by investigating and characterising the real-world work of testing we can help question why such 'bad practices' occur and how improvements might be made. We also argue that the testing literature is too focused on technological issues when many of the problems, and indeed strengths, have as much do with work and organisation. In this paper we use empirical examples from four systems engineering projects to demonstrate how and in what ways testing is a cooperative activity. In particular we demonstrate the ways in which testing is situated within organisational work and satisfices organisational and marketplace demands.

Keywords: Dependability; Ethnography; Ethnomethodology; Organisational Issues; Software Development; Systems Testing; Work Practices.

1. Introduction

"Software companies face serious challenges in testing their products, and these challenges are growing bigger as software grows more complex. The first and most important thing to be done is to recognize the complex nature of testing and take it seriously" (Whittaker 2000)

Testing is a key part of any systems engineering project. The human effort spent on testing can easily exceed half the project total (Juristo et al 2006a, Tassey 2002, Brooks 1975). Inadequate testing has been the source of many systems failures (Patton 2006); Tassey (2002) has estimated that inadequate testing is costing the USA 59.5 billion dollars every year. In the UK, a recent report by the UK Home Office Identity and Passport Service (2007) into the turbulent go-live of the PASS system at the United Kingdom Passport Agency (during which a backlog of 565,000 passport applications built up) recognises inadequate testing as one of the three "important causes" of the failure, and three of the ten "high level lessons to learn" focus on testing. Likewise, according to a report by the UK Transport Select Committee (2008), inadequate testing played a central role in the disastrous opening of Terminal 5 at London Heathrow Airport (which saw the temporary loss of 23,205 bags, the cancellation of 54 flights, an estimated cost to British Airways of 25 million pounds and served as a major contributing factor to the UK Government's decision to break up the British Airports Authority). In the words of Juristo et al "testing practices in industry generally aren't very sophisticated or effective" (Juristo et al 2006a, p.19).

Testing has, accordingly, been a major focus for systems engineering. This literature has predominantly focused upon new technologies and new methods for testing, and are often of the view that existing practices should be swept away. But rather than view real world practices as an embarrassment, we believe that an analysis of these can produce useful insights. Firstly, we are of the mind that whilst testing in industry might not always be sophisticated or effective, the ways in which it is done and the priorities it addresses are not naïve. Secondly, if testing in industry is to be improved, then some appreciation of the circumstances in which testing is done can reveal challenges that new methods and technologies must overcome. We seek to address how testing is done 'in the wild' as a real world, real time activity. Looking at four software projects, we investigate what testing involves, what demands it satisfies and what barriers are faced.

Schmidt and Bannon (1992) argue cooperative work is not simply another name for group-work but refers to people working interdependently. During our studies we have noticed that the work of systems testing is cooperative in this way; to use Schmidt and Bannon's (1992) terms, testing is done by transient ensembles of people, with shifting priorities, and within and beyond organisational boundaries. That testing is a form of cooperative work rather than a mere technical procedure is evidenced by the myriad ways in which it involves various kinds of orientation to planning and procedures, necessitates forms of distributed coordination work and requires some subtleties amongst workers in the form of awareness of the work of others. The various ways in which awareness is developed, in which work is made public and available to others, are essential ingredients in doing the work, as part of a socially distributed division of labour. Such a focus on cooperative work in testing has rarely been taken, and never beyond the boundaries of user testing; this point is underlined in the next section of this paper where we review the human and organisational literature on testing. Following that, in section three we outline our (ethnographic) method, in section four we discuss our fieldsites. In section five we summarise the features of testing covered by the examples, and in section six argue that these features occur because testing is a thoroughly context oriented activity.

2. Background

In the early formalisations of systems engineering (eg. see Royce (1970)) testing was a project phase that followed the completion of the program and took place before its installation. Since that time, a great deal of effort has been put into mitigating problems earlier. Unit testing and integration testing have situated testing within programming rather than after it, and formal methods and test driven development attempt to shift testing ahead of programming. Testing now also goes well beyond mitigating defects in programs, looking at issues such as their performance, reliability, internationalisation, and security. Testing also no longer simply focuses on the technology alone but on socio-technical issues such as acceptability, usability and fitness for purpose. Testing is also increasingly focused on systems rather than software in isolation. Labour issues are also changing. We still find it common for programmers to test their own program, particularly in smaller organisations, but it has for a long time been seen as better practice for someone other than the programmer to do the testing. Initially programmers were encouraged to test other programmers' work rather than their own because they may be unwilling to admit to their own faults, but the increasing scope and complexity of testing is now serving to make testing a profession in its own right. Many larger software teams include professional testers. Specialist consultancies are often used to supply expertise, particularly in specialist tasks such as performance testing. We are also seeing the emergence of software test centres to which testing can be outsourced. It is now also common to involve project stakeholders in testing. In particular, actual or proxy users are often involved in usability testing. Senior members of customer organisations are often involved in acceptance testing. We have also noticed that on large systems projects that project board members take great interest in testing, reinforcing the need for testers to be accountable and responsive to the organisation. In the shift from testing at the end to throughout the development process, there has been a corresponding shift from individual to cooperative work, that is, testing has become a CSCW issue.

Testing can appear a somewhat fuzzy concept, particularly when viewed in the light of real world projects and systems. So whilst Jorgensen (2002) suggests "Testing is obviously concerned with errors, faults, failures and incidents. A test is the act of exercising software with test cases. A test has two distinct goals: to find failures or to demonstrate correct execution" (p.4), Beizer (2003) argues "Testing is a process in which we create mental models of the environment, the program, human nature, and the tests themselves… the art of testing consists in creating selecting, exploring or revising models" (p.22). We are not attempting to create a new definition of testing in this paper, but rather to navigate this fuzzy concept by looking at what testing is and how it is meaningful from a practitioner point of view. We are taking a similar perspective to Martin et al (2007a) who discuss the achievement of dependability in the engineering of two healthcare technologies. By taking an ethnographic view, Martin et al are able to show the ways in which dependability is an outcome of the reasoning and argumentation approaches stakeholders engage in. Martin et al's interest is in the achievements made in design and testing, and they argue these arise through the ways in which design and testing are socially organised. They focus on user acceptance testing, and claim

> "Although acceptance testing offers a framework for accomplishing rigorous testing, realisation of that rigour depends to a large extent on applying local expertise in devising both appropriate tests and test criteria. The status then of a user acceptance test as a guarantor of fitness for purpose is not something that rests on its formal character per se, but depends also on the specific details of its implementation in practice." (Martin et al 2007a, p481)

Woolgar (1991) and Sharrock and Anderson (1994) have also undertaken ethnographic studies focusing upon usability trials. Woolgar looks at how the user is "defined, enabled and constrained", and Sharrock and Anderson at how the user is "typified" during the course of technical discussions. Winter et al (2008), in their ethnographic study around a usability testing package, document the wide and different range of interests in testing, from designers and system architects to management, marketing and clients. Looking beyond usability testing, Carstensen and Sørensen (1995) look at the coordination of testers and suggest mechanisms for supporting them. Other papers by Sharrock (Button and Sharrock 1992, 1996, 1998; Sharrock and Anderson 1993) mention testing but take a general focus on project work in software engineering rather than a particular one on testing. In general, ethnographic studies of testing are rare and predominantly focus on usability testing.

There are also a number of studies of testers that explore cognitive factors and team dynamics. Unlike the ethnographic studies, these are predominantly laboratory or questionnaire based (see for example, Da Cunha and Greathead (2007); Capretz (2003); Miller and Zhichao (2004)). The underlying concern seems to be finding the type of people who are best suited to doing a testing job, and the ways in which teams should be constructed. Our contention is that such studies are often done with little regard to what the work of testing actually involves and software projects rarely have the luxury of selecting afresh who will do the testing and of affording them the ideal time and resources to get that testing done. These studies have undeniably offered useful insights, but we think it is important to attend to testing in context.

Authors such as Kaner et al (2002) identify themselves as of the 'context-driven school' of software testing, arguing that testing must be good enough (Kaner et al 2002; Bach 1998) for the system it is for rather than tested against any immutable criteria. The 'context driven school' has a pragmatic approach and provide lessons based upon real world experiences in testing. There are many other practical guides on how to do meaningful tests and manage tests in a competent way, including Whittaker's (2002), who argues that there is enough on testing theory but not on actually how to do it. A general complaint amongst such pragmatic books is about the gap between the academic discussion on testing (both in research and in text books) and the realities. We therefore feel a strong affiliation with these authors, but of course these works are focused on 'how to do it' rather than 'how it is done' and look to best practices rather than attempt to explicate ordinary action.

One person who is concerned with ordinary action; how personal, organisational and procedural problems spiral and blend within the overall problem of software testing is Evans (1984). But he gives a "fictional account of a real testing environment" although "based on several actual project situations in which the planning was poorly done" (p187). To find detailed discussions of actual practice, it is often necessary to look beyond the testing of software systems: Collins (1988) discusses for the transportation industries, the relationship between 'trying' (extensively testing the various limits of a technology) and 'showing' (providing a demonstration of the technology passing particular tests for a particular audience); Pinch (1993) claims testing in any situation involves projection of performance; Downer (2007) discusses the problems of standardisation in the resilience testing of jet engines; Mackenzie (1990) discusses the co-development of a weapons technology and the criteria by which it will be tested. Mackenzie (2001) has also written an insightful study of the development of formal verification, a movement that has had some limited success but has never replaced software testing as some imagined it might. Ethnographic studies of evaluation also cross the auspices of this paper, in particular Blythin et al's (1997) study of what counted as success and failure for two groupware systems. This paper also holds resemblances to other studies of 'diagnostic work' (see Buscher et al 2009).

3. Fieldwork

We have undertaken fieldwork within several organizations, examining the work of software testing. Our approach to fieldwork and analysis follows that discussed by Randall et al (2007). Our approach is ethnographic, in that we have observed, questioned, discussed, video taped and written down what it is testers do in their work. We have then transcribed our data and analysed it by looking closely at how work is made and kept orderly by those doing it. The examples in this paper have all been analysed at one or more data session, each of which has had multidisciplinary participation (including Computer Science, Sociology and several other disciplines).

Our approach is not just to list the kinds of tests that are done and the technologies used (what Button (2000) calls scenic ethnography) but to take an interest in how testers carry out their work, what they talk about, how problems arise and are settled, etc. To use the jargon, our interest is in the 'ethnomethodology' of testing; as such our studies are akin to the ethnomethodological studies of Software Engineers by Button and Sharrock (1992), Rönkkö et al (2005), Martin et al (2007a), and others. We have taken a deliberately naïve, anthropological perspective, sidelining our preconceived, academic ideas of what testing ought to be and concentrating on what testing actually involves. We try not to begin with preconceived questions, but take an inductive approach to discovering what is interesting in any particular place. This approach is organized (see Harper (2000)) but is by no means formal or prescriptive and therefore does not do the same work as other investigative approaches that are used in empirical software engineering research (see Randall et al (2007) for a discussion). We say this because some of the social studies of testing mentioned in this paper have differences to our own and because we are not taking the experimental approach often demanded in empirical software engineering. For contrast, Runeson (2006) has done a comparative analysis of unit testing; work that inevitably has to ask the same questions of each site. This kind of research is certainly very useful and can lead to new insights (see Juristo et al 2006b), but in doing comparative analysis one must skirt over the unique features of different projects and the specific (and often unexpected) ways in which testing relates to the particular work of a particular organization.

Our fieldwork has taken us into several sites, four of which we report on in this paper. None of the work we focus on is extraordinary, and none of it is safety critical. This is in line with recent work that has started to consider the more mundane but overwhelmingly common technologies for which "undependabilities are frequent and are dealt with in a routine manner" (Martin et al 2007a, p468). All data has been anonymised, which includes changing names and omitting unnecessary detail.

3.1 Testing in a Small, Agile, Software Product Company

Site one is a small software company that, at the time of our fieldwork employed four programmers, a technical director (who spent a significant amount of time programming), a customer relationship manager and several others in administrative roles. The company was founded in 2002 by the technical director and customer relationship director, who had been working together as programmers in another software company. The company was initially funded on venture capital, and made its first software release in 2004. The company continues to expand its customer base, has moved to larger offices and is now regularly hiring new people. This company produces a development environment for the rapid construction of software for mobile devices, their market being a small but growing number of businesses with mobile workers. This company used, to a certain extent, the agile method 'eXtreme Programming'. Agile methods rely on customer involvement, however in the case of software product development such involvement can only be achieved in roundabout ways (Lippert et al 2002, Martin et al 2007b), and so it should be no surprise that this method was only partially implemented. This company therefore is not a 'good example' of using an agile method, but on the contrary is one of many small companies we are aware of who are selectively drawing from (some might say misusing) agile methods. This misuse we believe is "for good organizational reasons" (Martin et al 2007c), small companies often have to take shortcuts and be dynamic in order to keep the money coming in and hopefully to grow.

There was at the time no professional tester at this company; instead the programmers were doing their own testing. This included writing unit tests as the code was written, and running regular regression tests. There was also a regular testing phase at the end of each of what they called an "iteration" (a development phase lasting several weeks). The company at which the directors had previously worked had taken testing very seriously, too seriously they thought, spending too much time writing unit tests and running the time-consuming regression tests. The testing phase at the end of each iteration would be around one quarter of the iteration, typically one or two weeks. The onus of work during this phase was on integrating code and putting the resulting software to use against various informal scenarios. Individual programming tasks undertaken in each iteration were written on task cards, and during the testing phase these cards would be distributed amongst the team to inspect. The programmers would usually not test their own code, and would often complain if assigned to test something they had written. In addition to the task cards, which would be pinned to a notice board, the programmers had three whiteboards. At the start of the testing phase they would discuss what they ought to check and validate and would write up a list of these on a whiteboard and tick them off as they went. We would also see them do things like sketch the architecture of the software they had implemented during the iteration, which seemed to help in deciding on what to test. The testing phase was also when the user manual was written. When we undertook our study they warned us that the testing phase was "boring", and whilst we did not think this was the case it became clear that the testing phase was the least enjoyable part of their work. Complaints and sarcastic comments were common and tempers would sometimes fray. Acceptance tests were also supposed to happen, with the Customer Relationship Manager making a decision on these, but this seemed little more than a formality.

Testing then at this company occupied a substantial amount of time, even though efforts were made to not spend "too much time" on it. Tests were figured out in the light of (rather than in advance of) what had been coded, and this was done individually or by talking about it as a group. There were no real usability or acceptance tests to speak of. However whether their software was usable or acceptable to customers was a pervasive concern, as we will show. The fact that this was a product company, attempting to bring something new to the market, and in many ways to create that market underlay the shape of testing. Getting the product to market before their capital ran out was a prime concern and a good reason for not doing 'full' testing. That their customers did not yet exist also meant that any final say on usability or acceptability was impossible. We now wish to give two vignettes characterising testing at this company.

3.1.1. Example: Performance Testing

In this example the programmers at site one are testing something called "the push server." This push server should send a single message to multiple mobile devices. The push server was required by a customer that the programmers estimated (through informed guesswork) 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 this push server. The test is carried out by each of the four programmers and 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 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. 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. With this being a product company, during testing the user manual is written. The tests serve not just as tests but as exemplars. The exemplary 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 (1976) and many others recommend, but sometimes only after the software is shown to work, something that Myers specifically does not recommend. Performance testing in this example is, from a technical point of view, rather slap-dash, but it should be clear that it uses organisational resources and strives to meet organisational demands.

3.1.2 Example: Failure

Failing a test usually lead to steps to resolve that failure, be it an immediate fix or a decision about when to do a fix. Sometimes such delayed fixes were scheduled (for the next day, or the next project iteration) and sometimes these were put off until some indeterminate time in the future. This is an example of putting off a fix indefinitely. Following a test, 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 (a virtual machine is a computer simulated on another, e.g. a Windows PC might be run as a virtual machine on a Mac). The programmers talked through and explored a variety of possible causes and solutions (it seems programmers do not always need to know the cause of a problem to find a solution to it) 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 was only after exhausting the possibility of a quick or obvious fix that the value of actually fixing it came to light. Deciding not to fix this problem but 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 (particularly things about their users) and is not the subject of any numeric calculation (see Alby and Zucchermaglio (2009) for a related example). The programmers 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, that decisions can turn out to be wrong and changed later.

3.2 Testing in Open Source Development

To discuss testing in open source projects, we will discuss two sites. The first concerns the development of a reasonably popular software product that is being developed by a very limited number of people with a seemingly overriding concern to build their user base. The second concerns a much larger, more distributed, more technology focused group.

Site two is a life-sciences research centre at a large, English university in the prestigious 'Russell Group' (note: this is neither of the authors' institutions). We have studied several aspects of work in this research centre, and in this paper concentrate on the work surrounding the development of an extension to software originally developed at a North American university. The software supports the creation of formal representations of 'domain information', and the extension supports a new method of formal representation. Many of the users or intended users of this extension are in the life-sciences, but the software is also used by researchers in many other areas, by librarians, and others with an interest in formal representations. The original software and the extension to it are open source software products. The software has over 100,000 registered users (registration is optional), and the mailing list for the extension worked on by the professional programmer is just shy of 2000 members. As with many open source projects these were not, as is often mistakenly thought (Feller and Fitzgerald 2001), created by thousands of programmers around the world, but were created by a handful of people who knew each other; the original software by a team in North America, and the extension by the UK programmer who travelled regularly to work with those developers. Other programmers could, and sometimes did, contribute code to this project; many of these programmers would be project students within the two institutions. Program testing on this project was done by the programmer, was done alone and seemed to be answerable to no one. However user acceptance testing was a major part of this work, with the programmer's superior (a professor) and a co-worker (a researcher in Bio-informatics) both acting as proxy users. An overriding concern on this project seemed to be getting the software out and building a user base, much more so technical correctness. Care was taken however that the software would work, error free, for the teaching work in which the software was used. So again this is not a 'good example' to be aspired to, but rather this programmer's practices are representative of many open source projects (see Feller and Fitzgerald 2001) and of software development in science (Segal 2005).

Site three is a large distributed open source development project developing a Java Virtual Machine, this being a low level technology that allows code written in Java to be executed on different types of machine. This project has a high number of academic researchers participating as this infrastructure affords useful opportunities to experiment with techniques for optimization and novel features; a large number of publications and dissertations have been written concerning changes to this technology. A large technology company also has several employees working on this system. The project has a hierarchical organisation, similar to many other open source projects that began as closed source, industry projects (Feller and Fitzgerald 2001). Of particular note, there is a 'core team' who have ‘write access’ to the code base and so are in effect gate keepers, deciding when contributed code is good enough for inclusion in the system. Core team members must be frequent contributors to the project, and are elected by other core team members. There are over 250 named users of this product (the real number will be much higher), there are around 20 core members and 3 steering committee members. We have spent time examining the mailing lists, code base etc. and have done face-to-face interviews with a member of the core team. Testing on this project includes nightly functional tests and performance tests. The performance tests are used to show how the system compares to its competitors, and so are detailed and a topic of interest for many on the project. The quality standards on this project seem to us to be set high by the core team, meaning that the technology is extensively tested but also meaning that contributors often do enough to get their paper published but not the extra necessary for their code to be accepted into the code base. Another issue is that testing must involve running the software on a set list of different machines, some of which are fairly obscure giving the owners of those machines certain powers and responsibilities in testing.

3.2.1. Example: User Acceptance Testing

This example is from site two, where a substantial concern is the acceptability of the system to users. The example concerns the programmer doing user acceptance testing. User acceptance testing on this project is done either by a particular bioinformatician, or by the professor for whom the programmer works. These tests are usually to show these people the software and to ask them if it is what they meant when they discussed new features to implement. In this example, the programmer (Jon) and the bioinformatician (Bob) discuss a newly implemented feature that allows items to be imported into the system. They are discussing the ways in which imported items are represented on screen (currently they are represented in bold typeface).

> *Jon says "Why do you not like the bold? Do you not like the bold?" Bob 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, It it could be, it could be a lot of different things. And, also because, imports for me is kind of a, a tertiary function, that you learn to use a bit later on."*

In this example we see Bob, 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 when they talk about their users, for example they will question whether the requirements of one user is representative of all users. As we see in this example, participant users also situate themselves. Although ideas of what the user wants and will do are very important in usability and acceptance testing, “knowing the user” (as Woolgar 1991 puts it) is problematic. The user is not always an on-call person, possibly because it is unknown exactly who a user will be, and partly through general issues of availability but also because on-call users must be listened to in the context of others who can’t be there. As we saw in section 3.1.2, the user is important in settling decisions, 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.

3.2.2. Example: Failure

This example is taken from site three. It is from an interview with a core-team developer working on the distributed, open source project. In this interview the programmer is discussing regression testing. The developer shows the interviewer the list of passed and failed tests that are emailed to anyone signed up to the ‘regressions mailing list’.

> *“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 seventy seven 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 interview 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, and thus we find some irony in being told the thing that helps the development community is something to help overlook long-term failures. This requirement to overlook failures and the very means with which failures are presented (listed in an email) are in most respects dependent on the fact this project is widely distributed. A fix is not always urgent, not everybody's problem, and is often best left to or coordinated with others. The regressions mailing list is as much to demonstrate the way the project is going (are errors building up?) than it is to alert members to immediate problems.

3.3 Testing on a Large, In-House Development

Site four is a large Scottish University (again not one of the authors' institutions). We report on the configuration of a COTS (Commercial Off The Shelf) system within that institution for handling student information. Such information covered what they termed the 'complete student lifecycle', from enquiries and applications, right through to final results. This is a large institution (over 25000 students) and so implementing this system is a major project. The project employs around 50 people, involves regular participation by two consultants, is overseen by an external project board, and is accountable to several stakeholder groups. The technology is intended to support major organisational change, securing the global standing of the institution in an increasingly competitive market. The project was scheduled to last 3 years, but schedules have now slipped by at least 1 year. The project team includes four full time testers, with other team members often being seconded to testing. There is also a training team who coordinate extensively with the testers (and in fact face regular problems in separating testing from training, which is seen on the project as undesirable but sometimes unavoidable). The University is decentralised and geographically distributed around a city, and so multiple parties at multiple sites will use the system. There will also be systems interfaces accessible to external parties (such as applicants). Testing, and in particular performance and user testing therefore is a wide, and logistical problem. This also reinforces the reliance on documentation and scheduled meetings to coordinate project work. The system clearly is business critical and testing is being taken extremely seriously. The project board we have noticed spend a great deal of time discussing testing. However, as with many projects of this size, we have seen that as the project has overruns, testing is substantially squeezed in order to meet deadlines. These deadlines are difficult to move, partly as a result of the academic cycle (a delay can easily become a year's delay) and partly because some of the systems being replaced are unreliable and do not meet current requirements. This institution is experiencing many of the troubles documented by Cornford and Pollock (2003) in their study of the computerisation of four British universities, and so whilst it is possible to criticise their practices they are indicative of software engineering in this sector.

3.3.1 Example: User Acceptance Testing

Issues such as the responsibilities of testers, what tests could realistically be done in the allotted time, whether testing and training should be done simultaneously, and whether testing should be done with real or simulated tasks repeatedly arose. The following example is of dialogue between two users, taken from a user acceptance test at site four:

> *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 scope might not be met. Planning and estimation of what can be achieved is an error prone art, and additionally 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 has more stamina than the other. The possibility of reducing the scope of what is achieved during the scheduled test by moving some of that scope outside is discussed. This re-scoping, as was the case with the initial scoping is done with reference to some practical organizational considerations: who, what, where, how etc..

3.3.2. Example: Performance Testing

Given our ethnographic research strategy, we are not interested in commenting upon what we think the testers ought to have done. This does not preclude us however from discussing various tests and types of testing that the testers did not do. This is because the testers often discussed what tests could be done and decided which were best done given the resources available. This example concerns a decision not to load test.

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 end deciding 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 site four. Such documents are not standalone pieces of information but are relevant within an array of other documents and constituent in ongoing discussions and decision making. A number of discussions preceded the writing of the proposal mentioned above. 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 3.1.2 we saw the developers at site one 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.

4. Features of Testing In the Wild

In the previous section we discussed examples from four of our fieldwork sites. We have selected these examples to characterise testing as it is done 'in the wild'. There are of course many techniques for doing testing and many things that can be tested; six examples cannot possible cover every aspect of this field. Rather we have chosen short examples that are illuminative of the orderliness of testing as an organisational activity. We have not focused on newsworthy achievements or experiences in testing and have purposefully not discussed safety critical testing. We are certainly making no claims that the examples represent best practice. What we are trying to achieve is a characterisation of run-of-the-mill testing, one that can supply insights into the kind of work that is currently done in many organisations on a mundane basis. This kind of testing is not usually safety critical but is often project or business critical. It is common in the testing literature to use stories of good and bad practice, stories of the kind of work testers should aspire to or avoid at all costs. We are following a different path and using examples of the kinds of work that we hope will be recognisable to anyone with experience in real world systems development. In this section we wish to draw themes between the examples.

The first theme is *plans are followed dynamically and remade as testing progresses.* We saw questions of what was achievable given the time and resources available being regularly dealt with. A key element of this, particularly at sites one and four was the formation of test plans. At site 1, a test plan was formulated at the start of each testing phase, and at site four a fairly comprehensive project plan was developed. As coordinating mechanisms within socially organised 'real world, real time' work activities, the point of such plans is to co-ordinate the work of people in order that separate work activities and tasks come to have a coherence and thereby meet goals efficiently. Plans do not simply execute themselves nor is the relationship between the plan and the action it directs a mechanical one (Suchman (1987), Dant and Francis (1998), Rönkkö et al (2005)). Firstly, following the plan often involves far more than can reasonably be specified within it. It is clear from the examples that testing is more creative than simply following a set series of steps. Secondly the workers need to be able to respond, to adapt the plan in the face of changing and frequently urgent organisational demands. For example at site one we saw the programmers indefinitely delaying non-urgent work until "calls come in". Thirdly the extent to which the tests were fulfilled depended on how well testing progressed within the allotted time frame and as testing developed the team had to make a series of judgments about what constituted sufficient testing. The examples from site four both show tests being reformulated as what was achievable became clear. Therefore we can identify several ways in which planning systems and plans are part of continuing action in testing work, and that the successful accomplishment of a plan is dependent on the practical understandings about what the plan specifies in *these* circumstances, using *these* resources, *these* people, and so on. Although plans may be presented as abstractions, as manuals, as statements of procedures, just what it takes to realise them is a practical matter of making the plan work through all the various and inevitable contingencies that arise. It is such activities that maintain the plan by dealing with those things which arise, the things not planned for, the things which suddenly come up, so that even deviations from the plan can be accommodated to sustain its spirit.

The second theme is *testing involves work to stay organised*. This work is not limited to the planning noted above, but also to tracking, coordinating, handing over, giving help, articulation, staying aware, making accountable and so on. As Button and Sharrock (1996) have claimed, this work to stay organized is a constituent part of any systems engineering. We note the ways by which such organisation is achieved varies from site to site. At site one there was informal conversation and writing on shared displays such as white boards and cards. At sites two and four meetings were more extensively relied on, and site 4 also relied extensively on documentation. Sites two, three, and four also relied heavily on emailing and the web. Testers must not only supply effort into cooperating with other testers but also with workers in and beyond the wider organization. At sites two and four testers coordinated in various ways with 'users' and with their superiors. A common problem facing testers, that there just isn't enough time, seems to stem from the fact that testing is situated within other work. Development work seems almost inevitably to overrun and slip into the time allocated for testing, and we have noticed that testing is done in parallel with other activities: for example 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. Another aspect of staying organised is managing emotion, be it the management of boredom at site one or the management of stress and users' 'worries' at site four.

The third theme we notice is that *time is of constant relevance*. It is widely reported that not enough time is given to testing, but (from our anthropological perspective) our results only show site four struggling to do what they saw as sufficient testing in the time available. However our studies do confirm that time is regularly a consideration at all sites and, as with all cooperative work (Reddy et al 2006), a significant factor in the way testing is organised. Time was often seen as a limited and valuable resource. Even in the iterative development of site one or the open ended projects at sites two and three the question of whether something is the best use of the time available was fairly constant. Another characterising feature of the work of testing at sites one and three was timing work to fit with deadlines. Such timing issues are easily reframed as a classic example of Garfinkel's (1967) 'administrator's problem' since decisions on whether the time and effort are justified are essentially and contingently organisational. Drawing upon Garfinkel's description of the administrators problem, Sharrock and Anderson (1993) write:

"'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." (p.161)

The fourth thing we notice is *there is congruence between organizational structure, organisational priorities and the way tests are performed*. For example at site one, a group of five people does load testing in multiples of five. At site two testing is done with users close to hand, and tests are often done in anticipation of teaching courses within that institution. At site three the distributed nature of the project entails not everyone can do every test, perhaps because they do not own the correct equipment, or because they are disjoint from the contributors of certain code and so are told not to worry about certain failures in that code. At site four testing is shaped by who is available, what rooms are available and where, and what equipment is available. Ahonen et al (2004) discuss the effects of organisational structure on the testing process and results, finding it has a significant impact. There is clearly a resemblance between our findings and those of Ahonen et al, although we are emphasising different aspects of organisation. Their orientation to organisational structure is broader and less concrete than our own, but at this level they are able to demonstrate benefits and problems associated with the ways teams and resources for testing are organised. Our work does not directly extend Ahonen et al's findings, but there is resemblance and we find ourselves in agreement that the problems of testing are not just to do with "test stages, tools and technologies… a common assumption in the literature" (p276) but also to do with organisational issues.

The fifth theme is *tests are attributed significances*. Not all possible tests are undertaken, we have seen testers choose which are the most significant to do given the time available. For example, a load test was not of significance during 'phase one' of the work at site four. A successful test can also be attributed a limited significance, for example a successful test of '5000 connections' at site one did not have any significance regarding the limits of how many connections could be handled. We have also seen testers debate the significance of a failure, particularly whether anyone would notice that failure. At site one we noted that dependent on timing (whether it is a first test, and whether the user manual needs an example) tests can attempt to break the system, but can also be to show and exemplify it working. We also notice that failure must be managed, noticing a fault or failure does not automatically lead to its fixing. At site one a failure was not fixed because it seemed as if no one would notice, at site 3 failures were actively overlooked as they were not everyone's responsibility. A key feature of attributing significance seems to be through conversation. Conversation is a key part of the work of testing, between developers, between board members, between users etc. Conversations cover such issues as "who will notice" "who will care" "should we do this?" etc, which in effect is a discursive form of risk management. This does not replace formal risk management (at site four it took place alongside a formal approach to tracking risks). We also notice through the course of these conversations tests are rarely dismissed but rather they are indefinitely delayed.

Finally, we notice *testing involves reasoning and speculation about practices and situations of use*. Our studies confirm Pinch's (1993) and Woolgar's (1991) points that a substantive part of a systems project involves reasoning about what users might do with the system. At sites one and four, performance testing is done with reference to how many users might reasonably use the system at any one time. At sites one and two we have seen discussion of what users might do in settling decisions on whether a failure is significant, and whether a user interface passes a usability test. Therefore the practical sociological reasoning (Sharrock and Anderson 1993) of testers is not limited to how to coordinate during the course of testing, but is central to deciding what it is a reasonable test to set.

5. Discussion

We began this paper with a quote from Whittaker (2000) on the complexity of testing. Whittaker's argument is that as software is increasingly complex, testing that software is also increasingly complex. Our position is that such complexity is not just along technical lines (although we are certainly not claiming that software testing does not face technical challenges) but is also along human and organisational lines. To explore this, we have introduced six examples from four sites and, drawing from these, discussed six themes.

Are the six themes simply symptomatic of bad practice? Whilst arguably none of our examples portray technical sophistication, each portrays an organisationally meaningful activity. The overwhelming feature of the mundane realities of testing seems to be how to deploy resources to find faults or design problems that might inhibit the system from meeting customer or user needs. The themes are in-fact common to CSCW. Such perennial issues are, through the bitter experience of many in CSCW, tameable but not resolvable by new methods and new technologies. The examples display the massive orderliness of testing as an organisational activity, characteristic of, indeed definitive of 'cooperative work'. Our interest has not been to prove that testing is cooperative work but to suggest that by treating testing as cooperative work, by pointing to some general features, we can begin to suggest ways in which the work of testing might be better supported. Such support can come through software (Carstensen, & Sørensen 1995), but it is clear that the problems of testing need not just a technical solution but ones that also focus on issues of human and cooperative practice.

For some time, very little attention has been paid by the academic community to what exactly is involved in real world testing and the reasons why testing is done in particular ways. However we should note that foundational arguments in Testing have some provenance in human studies. Dijkstra's much quoted statement on how program testing is inferior to formal verification, for example, goes as follows:

> "Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence" (Dijkstra 1972, p.864).

This quote is often used as an argument against relying on humans in testing: but often overlooked are Dijkstra's comments in the same paper that the means of verifying software should be useable, that they should be "intellectually manageable" so as not to increase the programmer's burden. Dijkstra's paper is infact overwhelmingly concerned with issues of practicality, achievability and the psychology of programming. Myers, in work that is also of enduring relevance to the field of testing (see Whittaker 2000), also shows these kinds of concerns:

> "Although one can discuss the subject of testing from several technical points of view, it appears that the most important considerations in software testing are issues of economics and human psychology. In other words, such considerations as the feasibility of "completely" testing a program, knowing who should test a program and adopting the appropriate frame of mind towards testing appear to contribute more toward successful testing then do the purely technical considerations." (Myers 1976, p.4)

Myers, and Dijkstra before him, mention issues to do with psychology and intellectual manageability, economics and discipline. They both made a strong case that the determinants of successful software testing have little to do with purely technical considerations. But despite the influence of these authors, their non-technical concerns have been left to fade. We think this is a problem, although in the light of our fieldwork, we would reframe their arguments and suggest contemporary problems in testing are primarily organisational in character and consequently they may be best understood, if not resolved, with respect to work practices. To borrow Simon's terms, in the face of real world complexity, the tester becomes a satisficer, "a person who accepts 'good enough' alternatives, not because less is preferred to more but because there is no choice." (Simon 1969, p.28).

6. Conclusion

Testing in industry is structured and meaningful along organisational and economic lines and so improvements to current practices, often seen as unsophisticated and inadequate, cannot be brought about purely through technically driven innovation. Rather the problem of improving software testing should be seen as a socio-technical challenge. This paper makes no suggestions for improvements to testing but is intended to set a backdrop for the ways in which improvements can sensibly be made. Technical innovation needs to be paired with an understanding of the cooperative nature of testing. Many of the insights into work practice made by the CSCW community are relevant to testing, and we suggest therefore: 1) The field of software testing ought to take more notice of CSCW; 2) CSCW can reasonably and usefully look beyond user testing to other features of testing; 3) The literature on the human and social aspects of software development (which is already drawing insights from CSCW) could sensibly integrate software testing into the discussion rather than ignoring it or treating it as a separate issue; 4) To be meaningful, empirical and experimental research into testing needs to take account in one way or another the satisficing nature of testing; And finally, 5) whilst there is a pedagogical role for descriptions of 'good' and 'bad' practice, studies of 'ordinary' practice are necessary in orienting academic research to the interests of software testing 'in the wild'.

7. References

Ahonen, J., T. Junttila, and M. Sakkinen (2004): Impacts of the Organizational Model on Testing: Three Industrial Cases. Empirical Software. Engineering, vol. 9, no.4, pp. 275-296.

Alby, F., and C. Zucchermaglio (2009): Time, Narratives and Participation Frameworks in Software Troubleshooting. Computer Supported Cooperative Work, vol. 18, nos. 2-3, pp. 129-146.

Bach, J. (1998): A Framework for Good Enough Testing. IEEE Computer, vol. 31, no. 10, pp. 124 – 126.

Beizer, B. (2003): Software Testing Techniques. India: Wiley.

Blythin, S., J. Hughes, S. Kristoffersen, T. Rodden, and M. Rouncefield (1997): Recognising ‘Success’ and ‘Failure’: Evaluating Groupware in a Commercial Context. In Proceedings of Group’97, The ACM SIGGROUP Conference on Supporting Group Work, Phoenix, USA, November 16-19, 1997, pp. 39-46.

Brooks, F. (1975): The Mythical Man Month. Essays on Software Engineering. Boston: Addison-Wesley.

Büscher, M., J. O’Neill, and J. Rooksby (2009): Designing for Diagnosing: Introduction to the Special Issue on Diagnostic Work. Computer Supported Cooperative Work, vol. 18, nos. 2-3, pp. 109-128.

Button, G. (2000): The Ethnographic Tradition and Design. Design Studies, vol. 21, no. 4, pp. 319-332.

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

Button, G. and W. Sharrock (1996): Project Work: The Organisation of Collaborative Design and Development in Software Engineering. Computer Supported Cooperative Work, vol. 5, no. 4, pp. 369-386.

Button, G. and W. Sharrock (1998): The Organisational Accountability of Technological Work. Social Studies of Science, vol. 28, no. 1, pp. 78-102.

Capretz, L. (2003): Personality types in software engineering, International Journal of Human-Computer Studies, vol. 58 no. 2, pp. 207-214.

Carstensen, P., and C Sørensen (1995): 'Let's Talk About Bugs!'. Scandinavian Journal of Information Systems, vol. 7, no. 1, pp. 33-54.

Collins, H. (1988): Public Experiments and Displays of Virtuosity: The Core Set Revisited. Social Studies of Science, vol. 18, no.4, pp. 725-748.

Cornford, J. and N. Pollock (2003): Putting the University Online. Information Technology and Organisational Change. Maidenhead: Open University Press.

Da Cunha, A., and D. Greathead (2007): Does personality matter? An analysis of code-review ability. Communications of the ACM, vol. 50, no. 5, pp. 109-112.

Dant, T., and D. Francis (1998): Planning In Organisations: Rational Control or Contingent Activity? Sociological Research Online, vol. 3. no. 2, <http://www.socresonline.org.uk/socresonline/3/2/4.html>.

Dijkstra, E. (1972): The Humble Programmer. Communications of the ACM, vol. 15, no. 10, pp. 859–866.

Downer, J. (2007): When the Chick Hits the Fan: Representativeness and Reproducibility in Technological Tests. Social Studies of Science, vol. 37, no. 1, pp. 7-26.

Evans, M. (1984): Productive Software Test Management. New York: John Wiley & Sons.

Feller, J. and B. Fitzgerald, (2001): Understanding Open Source Software Development. Boston: Addison-Wesley.

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

Harper, R. (2000): The Organisation in Ethnography. Computer Supported Cooperative Work, vol. 9, no. 2, pp. 239 – 264.

Home Office Identity and Passport Service (2007): Report on key projects implemented in 2007. http://www.ips.gov.uk/passport/downloads/IPS-report-on-key-projects-implemented-2007.pdf (retrieved 20th September 2008)

House of Commons Transport Committee (2008): The Opening of Heathrow Terminal 5. Twelfth Report of Session 2007-08. London: The Stationary Office Limited.

Jorgensen, P. (2002): Software Testing A Craftsman's Approach. Boca Raton: CRC Press.

Juristo, N., A. Moreno, and W. Strigel (2006a): Guest Editors' Introduction: Software Testing Practices in Industry. IEEE Software, vol. 23, no. 4, pp. 19-21.

Juristo, N., A Moreno, S. Vegas, and M. Solari (2006b): In Search of What We Experimentally Know about Unit Testing. IEEE Software, vol. 23, no. 6, pp. 72-80.

Kaner, C., J. Bach, and B. Pettichord (2002): Lessons Learned in Software Testing. New York: John Wiley & Sons.

Lippert, M., S. Roock, and H. Wolf (2002): Extreme Programming in Action. Practical Examples from Real World Projects New York: John Wiley & Sons.

Mackenzie, D. (1990): Inventing Accuracy: A Historical Sociology of Nuclear Missile Guidance. Cambridge: MIT Press.

Mackenzie, D. (2001): Mechanizing Proof. Computing, Risk and Trust. Cambridge: MIT Press.

Martin, D., M. Hartswood, R. Slack, and A. Voss (2007a): Achieving Dependability in the Configuration, Integration and Testing of Healthcare Technologies. Computer Supported Cooperative Work, vol. 15, nos. 5-6, pp. 467-499.

Martin, D., J. Rooksby, and M. Rouncefield (2007b): Users as Contextual Features of Software Product Development and Testing. In Proceedings of Group'07, pp. 301-310.

Martin, D., J. Rooksby, M. Rouncefield, and I. Sommerville (2007c): ’Good' Organisational Reasons for 'Bad' Software Testing: An Ethnographic Study of Testing in a Small Software Company. In Proceedings of ICSE’07, pp. 602-611.

Miller, J., Y. Zhichao (2004): A Cognitive-Based Mechanism for Constructing Software Inspection Teams. IEEE Transactions on Software Engineering, vol. 30, no 11, pp. 811- 825.

Myers, G. (1976): The Art of Software Testing. New York: John Wiley & Sons.

Patton, R. (2006): Software Testing. Indianapolis: Sams Publishing.

Pinch, T. (1993): “Testing – One, Two ,Three… Testing!”: Towards a Sociology of Testing. Science Technology & Human Values, vol. 18, no. 1, pp. 25-41.

Randall, D., R. Harper, and M. Rouncefield (2007): Fieldwork for Design: Theory and Practice. London: Springer Verlag.

Reddy, M., P. Dourish, and W. Pratt (2006): Temporality in Medical Work: Time Also Matters. Computer Supported cooperative work, vol. 15, no. 1, pp. 29-53.

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

Royce, W. (1970): Managing the Development of Large Software Systems. In Proceedings of WESTCON, August 1970, reprinted in Proceedings of ICSE ‘87 the 9th International Conference on Software Engineering, Monterey, USA. pp. 328-338.

Runeson, P. (2006): A Survey of Unit Testing Practices. IEEE Software, July/August 2006, pp. 22-29.

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

Segal J. (2005): When Software Engineers Met Research Scientists: A Case Study. Empirical Software Engineering, vol. 10, no. 4, pp. 517-536.

Sharrock, W. and B. Anderson (1993): Working Towards Agreement. In Button, G. (ed) Technology in Working Order. London: Routledge, pp. 149-161.

Sharrock, W. and B. Anderson (1994): The User as a Scenic Feature of Design Space. Design Studies, vol. 15, no. 1, pp. 5-18.

Simon, H. (1969): The Sciences of the Artificial. Cambridge: MIT Press.

Suchman, L. (1987): Plans and Situated Action: The Problem of the Human−Machine Communication. Cambridge: Cambridge University Press.

Tassey, G. (2002): The Economic Impacts of Inadequate Infrastructure for Software Testing. National Institute of Standards and Technology, US Department of Commerce Technology Administration. RTI Project Number 7007.011.

Whittaker, J. (2000): What is software testing? And why is it so hard? IEEE Software, vol. 17, no. 1, pp. 70-79.

Whittaker, J. (2002): How to Break Software: A Practical Guide to Testing. Boston: Addison-Wesley.

Winter, J., K. Rönkkö, M. Ahlberg, and J. Hotchkiss (2008): Meeting Organisational Needs and Quality Assurance through Balancing Agile & Formal Usability Testing Results. In Proceedings of CEE-SET the 3rd IFIP TC2 Central and East European Conference on Software Engineering Techniques, Brno, Czech Republic, Oct. 13-15.

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. London: Routledge, pp. 58-100.