Patching the gaps in the code race:

An Explication of Defect Resolution Practices in Software Development

Submitted in fulfilment of the requirements for a degree of Master of Science
August 2010
Daniel K. Sullivan

Supervisor: Prof. Liam Bannon

B.Eng. (University of Limerick) 1991

Department of Computer Science and Information Systems, University of Limerick,
August 2010.
Abstract

This thesis examines how software professionals, working on a real software product, are aided by their work practices, on a daily basis, in dealing with the practical and local consequences of global distribution. We present an examination of the phase in project orientated software development work when defects are uncovered, documented, investigated and resolved. This is a period of intense, stressful, even frenetic activity, with frequent interaction between the Test, Development and Management teams who form a temporary, but highly focused community of interest. Their Work Practices require the daily use of artefacts such as defect reports and software builds and are reflected, and even come to be embodied, in the artefacts; artefacts and work practice in turn shaping one another.

This examination is done by means of a field study where a crucial part of the software work has been distributed. Through this study we find revealed to us the local nature of the work practices and their criticality to all those dealing with those blocking defects. Blocking defects are those defects in functionality that prevents the Testing team from examining areas of the software in particular features that have been implemented by the development team. Artefacts, such as defect tracking reports or packaged builds of the software, generated as a matter of routine, assist in maintaining the rhythm and visibility of work cycles, knitting together and underpinning the fluidity of work from day to day, to sustain the work practices over the many months of a commercial software project.

These Artefacts which straddle the boundaries of the work worlds of the various professionals need to exhibit contrasting flexibility and relative stability; flexible enough to allow workarounds to present themselves when faced with blockages to the progress of work of members of the project group, yet stable enough that work performed on them successfully builds upon what has gone before.

Resolving the blocking defects involves extending and augmenting technology based boundary objects such defect reports and software builds. This is done through the locally negotiated modification of the artefacts to create work-arounds to these blocking defects incorporating work practice and technological change. These locally devised practices, which involved Fitting, Augmenting, and Working Around software artefacts allows the work to shift between the need for stable or very slowly changing artefacts and the more elastic, rapidly evolving, contingent pressures placed on boundary objects. The ease or difficulty of making alterations or augmentations to artefacts straddling the boundaries of the Testing, development and management groups either allows or impedes the group members in working around blocking defects. Understanding these dynamics through the theoretical bases of boundary objects and work arounds allows us to better appreciate the role of software artefacts and work practice both routine and ad hoc.
Acknowledgements

There are a lot of people who have to be mentioned for their contributions to the production of this work. The order in which they appear does not reflect their respective importance to this endeavour.

My supervisor Prof. Liam Bannon, whose support, encouragement, perseverance and patience over the last number of years has shaped what I have written in this thesis. Kjeld Schmidt and Chris Exton who thoroughly and fairly assessed my work providing valuable and constructive feedback.

All the people in and associated with the Interaction Design Centre; in particular my colleagues and friends from the socGSD project, Anders, Anne, Gabriela and Gamel who served as a great support and inspiration. All the people in the various institutions that have funded and kindly supported me during my time as a postgraduate student, particularly LERO (The Irish Software Engineering Research Institute) and the Department of Computer Science & Information Systems at the University of Limerick.

The people at the field study site for making me welcome during my field studies, for being so generous with their time and contributing so greatly to the insights in this thesis.

My thanks and love to my sister Breda and parents Sheila and Daniel T. Sullivan.

Different aspects of this work have been a long time coming and a lot of people have provided inspiration and encouragement.

I have relied upon, used and abused my social circle for distraction, discussion, books, glorious beer, lunch and generally nonsense. So an extra special thanks to the denizens of the informal Catherine St. coffeehouse and conflict resolution institute Michael Mulcahy, John Browne, Neville Gawley, Eoin Brazil and Michael Barry for listening, ignoring, probing and arguing when needed. There are many others, from far and wide, too numerous to mention by name, who have supported me, in ways small and large. My sincerest thanks to you each and every one.

And finally, Anne B. Hallahan without whom, I’d still be wondering if it was worth it at all.
...night
Table of contents:

CHAPTER 1 - INTRODUCTION | 1
1.1. INTRODUCTION | 2
1.2. THE WORLD OF SOFTWARE DEVELOPMENT | 2
1.3. THE MOTIVATION | 5
1.4. THE RESEARCH QUESTION | 8
1.5. KEY CONCEPTS | 9
1.6. THE ENVIRONMENT IN WHICH THE STUDY WAS CONDUCTED | 11
1.7. THESIS ORGANISATION | 12
1.8. CHAPTER SUMMARY | 13

CHAPTER 2 – EVERYDAY SOFTWARE WORK | 14
2.1. INTRODUCTION | 14
2.2. SOFTWARE WORKPLACE STUDIES | 15
2.3. RHYTHMS AND CYCLES OF REAL WORK | 18
2.4. SOFTWARE DEVELOPMENT WORK IN PUBLIC VIEW | 20
2.5. DEALING WITH THE SHIFTING SANDS OF CONTINGENCES | 22
2.6. GLOBAL SOFTWARE DEVELOPMENT | 25
2.7. WORK PRACTICE ACROSS BOUNDARIES | 27
2.8. LOOKING BEYOND COLLOCATED PRACTICE | 29
2.9. THE EFFECT OF WORK AROUNDS ON BLOCKAGES IN KEEPING WORK FLOWING | 30
2.10. VISIBILITY IN THE DAILY WORK WORLD | 32
2.11. CHAPTER SUMMARY | 33

CHAPTER 3 – STORIES FROM THE CODE FACE | 35
3.1. INTRODUCTION | 35
3.2. STUDYING DAY TO DAY PRACTICES | 35
3.3. THE STORIES WE WERE HUNTING FOR | 36
3.4. HOW WE MIGHT FIND SUCH TALES | 39
3.5. THE FIELD WORK – THE STORY WE FOUND | 44
3.6. HOW THE FIELD STUDY WAS CONDUCTED | 49
3.1. CHAPTER SUMMARY | 52

CHAPTER 4 – THE STORY OF THE FIELD STUDY | 54
4.1. INTRODUCTION | 54
4.2. THE FIELD SITE OUTLINE DETAIL | 54
4.3. SCHEDULING AND DEALING WITH TIME | 58
4.4. A BREAKDOWN OF TEAM WORK AREAS | 59
4.5. THE ITERATION GAP | 62
4.6. THE VIGNETTE: THE DAILY BLOCKERS’ MEETING | 64
4.7. SOFTWARE BUILDS AT THIS FIELD SITE | 70
4.8. DEFECT REPORTS | 71
4.9. BLOCKERS | 78
4.10. BLOCKERS’ DOCUMENT | 80
4.11. THE DAILY MEETING | 85
4.12. THE PATCHING PROCESS | 87
4.13. CHAPTER SUMMARY | 91

CHAPTER 5 – DISCUSSION | 92
5.1. INTRODUCTION | 92
5.2. DEALING WITH THE DISTRIBUTED REALITY OF SOFTWARE DEVELOPMENT WORK | 92
5.3. MAINTAINING PRACTICE THROUGH VISIBILITY | 101
5.4. CIRCUMVENTING THE ITERATION GAP AND ‘WINNING’ THE CODE RACE | 107
5.1. SUMMARY | 114

CHAPTER 6 - CONCLUSIONS | 115
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>INTRODUCTION</td>
<td>115</td>
</tr>
<tr>
<td>6.2</td>
<td>CORE CONTRIBUTIONS</td>
<td>115</td>
</tr>
<tr>
<td>6.3</td>
<td>EMERGENCE AND MAINTENANCE OF PRACTICES AND TECHNOLOGY</td>
<td>116</td>
</tr>
<tr>
<td>6.4</td>
<td>LIMITATIONS OF THE STUDY</td>
<td>118</td>
</tr>
<tr>
<td>6.5</td>
<td>AVENUES OF FUTURE RESEARCH</td>
<td>118</td>
</tr>
<tr>
<td>6.6</td>
<td>SUMMATION</td>
<td>119</td>
</tr>
</tbody>
</table>
List of Figures

Figure 1: A Simplified model of Software Development Work .............................................. 7
Figure 2: A Representation of the Distributed Development Team ...................................... 46
Figure 3: Example of DTR entry in blockers’ document ......................................................... 67
Figure 4: History of entries for DTR - LEPU as of May 31st .............................................. 69
Figure 5: History of entries for DTR – HFB6 as of May 31st .............................................. 70
Figure 6: Defect Resolution Process Mundane repetitive detail ......................................... 73
Figure 7: History of state changes to the document and reasons for making them .......... 75
Figure 8: The relationship between builds and defect reports ........................................... 76
Figure 9: Screenshots of the Steps necessary to reproduce the defect .............................. 77
Figure 10: Confirmation of the Resolution of the DTR ....................................................... 78
Figure 11: The collapsed blockers’ document ..................................................................... 82
Figure 12: Red entry from blockers’ document = as yet unresolved ............................... 83
Figure 13: Yellow entry in blockers’ document – patch verified as fixed, yet to be verified in build ................................................................................................................. 84
Figure 14: Green entry in blockers’ document – verified as ‘Not a Bug’ no Development action needed ............................................................................................................. 84
Figure 15: Green entry in blockers’ document – patch verified as fixed in build after code submission ......................................................................................................................... 84
Figure 16: New Patches as they appear when embedded in the DTR ............................. 88
Figure 17: Older Patches as they appear embedded in the DTR ..................................... 89
Figure 18: the Patch’s place in the development Process .................................................. 91
Figure 19: Interaction between the build, fix and defect report ........................................ 91
Figure 20: Interaction between the build and patch, fix and defect report .................... 97
Figure 21: The Path from a Defect to a Fix ........................................................................ 103
Figure 22: The Comment section of a DTR showing the time line for state changes .... 104
Chapter 1 - Introduction

This thesis examines the local work practices that emerge as people try to resolve blocking defects in a large enterprise scale software development project. The phase in project orientated software development work when defects are uncovered, documented, investigated and resolved is a period of intense, stressful, even frenetic activity, with frequent interaction between those involved in the Test, management and development groups. This is also the case moreover when the element of distribution of a portion of the work activity is added to the work environment.

In focusing on the practices of defect resolution in an environment, where part of the work has been distributed we find revealed both the local nature of the work practices and their criticality to all those workers. The effects of this distribution, such as a significant gap in the time between the availability of builds, are open to being adversely affected by blocking defects in those builds. Blocking defects are those defects in functionality which prevent the Testing team from examining areas of the software product. In particular those feature areas which have been newly implemented by the Development team for that release of the software product or project. These are Work Practices that pull together mundane day to day activities and integrate less tangible software artefacts into the concrete building blocks of software work. This is achieved by means of online discussions, documentation, conference meetings to manage the defect reports, and as we will see the circulation of software patches. These work practices assist in the maintenance of the rhythms of daily, weekly work cycles and in sustaining this work over the long course of an entire software project.

The practices involved in resolving these blocking defects involve extending and augmenting technology based boundary objects including defect reports and software builds through the negotiated modification of the artefacts to create temporary local, work-arnouds to these blocking defects incorporating work practice and technological change. These locally devised practices, involving the Fitting, Augmenting, and Working Around (Gasser 1986) of software artefacts allow for a shift between the need for stable or very slowly changing artefacts and the more elastic, rapidly evolving, contingent pressures placed on boundary objects (Star and Griesemer 1989). The ease or difficulty of making alterations or augmentations to artefacts straddling the boundaries of the Testing, Development and Management groups can either aid or impede the group members in coping with the
blocking defects. Understanding this dynamic setting through the theoretical bases of boundary objects and work arounds allows us to better appreciate the role played by software artefacts in software practices both routine and ad hoc.

1.1. Introduction

Well understood, long established, and locally-derived work practices when used by experienced and highly capable people working in the extremely time pressed world of software development are a ready means to overcome the consequences of distribution that manifest locally. Such practices at a specific phase in software development also serve to knit together some of the intersecting work of the Test, Management and Development groups as they work through what is a critical juncture: the defect resolution phase of a software project.

Our research question is how do the practices of software professionals, working on a real software product, aid them in dealing with the practical and local consequences of global distribution on a daily basis. To answer this question we need also to consider just what are some of those local consequences and how are those practices maintained.

The research approach we have taken stresses the value of examining the situated activity in detail by immersing oneself in it, entering into the situation with a focus on the local in order to then talk about the larger issues of the dispersal of work across globally distributed locales. The research conclusions presented later in this thesis are thus primarily based on a qualitative case study and the subsequent analysis of the data gathered therein.

This research focuses on work practices, which we will expand on in the next section, in particular the patching of installed software builds and the use of a defect tracking system as part of the process of continuing to make steady, predictable progress over the course of a commercial software project. These work practices assist Testing and Development software workers when they are faced with the sort of blocking problems that can and do arise during the normal course of a software development project. Such blockages as we have found for consideration in this thesis mainly consist of defects in functionality which prevents the Testing team from examining areas of the software features that have been implemented by the Development team. These problems similarly block the Development team members from moving their work on completely from this functional area, as they are uncertain whether their work there is finished or not.

1.2. The world of software development

Software development involves a broad variety of phases: requirements gathering, specification, prototyping, coding, testing, defect resolution, system verification through to post release and
maintenance. Each of these phases links with and will have elements of the other phases woven through it. So even while studying one single phase we are still able to see the broader picture of software development work. This thesis is primarily concerned with a single particular phase out of the many that exist in software development. That phase is the defect resolution phase when the bulk of the coding and testing work takes place. We chose to concentrate on this phase because it is a comparatively neglected area of research, compared with other areas such as the design and specification stage or the broader strategic considerations of software development. This is despite consisting of much of the day to day reality of software work and involving the usage of shared concrete artefacts by Testing, Development and Management.

During the defect resolution phase, defects in the software are daily being uncovered, documented, verified, investigated and resolved. This is a period of intense even frenetic activity, with a heightened level of interaction between the Test, management and Development functions within a software project. Within this phase there is intricate and exacting work on display, as ways are found to overcome the defects and problems uncovered by routine testing: work which is further exacerbated by distribution. This work is a necessary part of what leads eventually to the resolution of the everyday defects that arise all too readily during the course of a software development project. By focusing on defect resolution work practices we open out for examination and discussion this crucial juncture point, this nexus in the work of testers, coders and managers. These work practices as we will see later encompass the documentation of the failed and successful attempts at defect resolution and the synchronised manipulation of defect tracking reports and software product builds. This is as part of the work of both Testing and Development, taking step by incremental step towards their mutual goal of a completed quality software product.

1.2.1 Practice in Software work

In approaching this research effort and before proceeding further we must comment on our understanding of “practice” and how the concept will be used throughout this thesis. We acknowledge that the notion of practice is problematic. The concept of practice may be spoken about in various ways and with a number of meanings. We recognise that practice can be considered as referring to

- What people believe or imagine they are doing
- What is prescribed as something that people must do or
- What has been proscribed as what people must not do
- What actually takes place from day to day as matter of routine

To summarise we will be using the latter form of practice; for us practice is what people are routinely doing to deal with the everyday problems of getting work done.
We appreciate that care needs to be paid to the understanding of “practice”. We share with our colleague (Sigfridsson 2010) a consciousness that thoughts on practice have the potential to conflict with one another and that any in depth discussion about practice has the potential to be divisive. Our team’s research focus on “real work” means that we chose to use practice as an observational lens and not as a topic of analysis in itself. The research perspective that our group drew on meant that our use of practice is in respect of “the coordinated activities of individuals and groups doing their ‘real work’ as it is informed by a particular organizational or group context” (Cook and Brown 1999 pp. 386-387).

Work practice is tied into the detail of the situation in which it takes place. In the context of the topic of this thesis, the software professionals have clearly understood work to perform in a particular setting. The nature of the project they were working on, who they were working with, what tools were at their disposal, etcetera all combine to influence and shape the agreed practice for doing their tasks. What is of interest for us as observers are their actual day to day activities, the “real work” making it all happen. What we are talking about is not one off events or singular activities, but the routine and repeated habitual actions of the quotidian or recurring performance.

When looking at work practice due attention needs to be paid to those individuals performing the work, the communities they belong to, and how they make use of artefacts in common to them to satisfy their different priorities at a crunch point in the software development cycle. Key to software development practice are those artefacts that are created, destroyed, mislaid, modified and maintained, and which often transition through iterations until the finalised version of the software is deemed complete. Practice involving artefacts spanning the Testing and Development groups can become embodied or reflected in the artefacts.

With our study of software work practice we are covering one area of work, and within that work phase we are concerned with one set of practices. This thesis focuses on explicating and uncovering for a wider audience particular work practices of the ‘patching’ of software builds which help in the articulation of the work of developers and testers as they tackle the resolution of specific blocking defects in the course of developing software.

1.2.2 Artefacts in work practice

We will present a number of artefacts, in the main electronically based and while we only briefly set out their place in the development cycle here, we will be looking at their use in considerable depth in Chapter Four. The work practices which are central to our discussion involve working with three artefacts:
1) Defect Tracking Reports (DTR)
2) An electronic document of blocking defects known as the ‘blockers’ document’
3) Software product builds

We will have cause to make reference to other tools and artefacts in use but these three serve as our primary focus. Defect tracking reports, ‘blockers’ document’ and the packaged builds of the software in which defects are uncovered, knit the day to day work throughout the course of the overall project. This is particularly important when so much else about the work environment is fluid, changing as it does from day to day.

The Defect Tracking Reports, or DTRs, are structured electronic documents contained within a Lotus Notes based database application. These Reports track in detail the circumstances of how the software defects were found. They also detail what steps are necessary to recreate or reproduce the defect or defective behaviour, what attempts are made to correct the defect and whether or not the defect is ultimately resolved. It also records the various state changes that occur to the Defect Tracking Report and also may include various file attachments including screenshots, data files used to produce the defect, log files outputted and code files that might fix the problem.

The ‘blockers’ document’ is an electronic document contained in a different Lotus Notes based database application, which lists and links to, by electronic means, those particular Defect Tracking Reports of problems that are significantly blocking the progress of work. The threshold of significance typically but not exclusively relates to the amount of testing work that is blocked from being performed with 4 hours of work blocked being the normal threshold.

Software product builds are the software code packaged in a format suitable for installation and normal use and testing. The software product build is a level of abstraction from the actual code that the software developers have written. Builds are automatically produced on a daily basis at a fixed time. Individual builds are commonly referred to by the date on which they are produced. From time to time, and for internal discussion, some builds may be labelled for milestones such as for a Beta release or Feature or User Interface Freeze.

1.3. The motivation

This research has taken place within the socGSD project\(^1\), which focused on social, organisational, and cultural aspects of globally distributed software development. The research we did represents an

---

\(^1\) “Social, Organizational, and Cultural aspects of Global Software Development”, a joint project between LERO (Irish Software Engineering Research Institute) and the Interaction Design Centre at the University of Limerick, Ireland. socGSD was one of the LERO cluster projects funded under PI grant 03/IN3/1408C by the Science Foundation of Ireland (SFI).
individual strand from the socGSD project, but it was undertaken very much within the overall frame of the project and with the support of the research group. Within this context, a field study has been conducted for this thesis, utilizing participant observation as a core method. The study is of the project team of a globally distributed software project in a multinational organisation across a period of 4 months.

1.3.1 Author’s professional background

Our own professional background is as a computer engineering graduate with more than a decade of having worked in the domain of software development. This background had motivated us to seek out and present to a wider audience those areas of software work that we considered had been for too long neglected as too mundane and transient for deeper consideration. We have over a decade’s experience various types of software work, from working with custom machine automation to shrink wrap desktop software products to enterprise deployments, and working in a variety of locales. We worked in Japan, programming logic controllers for a car components company and the United States as a build engineer and integrator and international liaison. In addition we had worked for a time as a project manager in the organisation that the study was conducted in. Understanding the existing relationship with the organisation will permeate all aspects of the study. This not alone cannot be avoided, but proves useful in expediting the process of becoming part of the work place. As the researcher has previously been part of this environment and has been the primarily instrument for gathering data we seek to make our report of this study as rich as possible in order to allow others to draw from our experience and learn from it.

1.3.2 Software builds in Software development process

Software product builds are being produced and defects are being resolved every single day. Yet much of this apparent drudgery of software is seen within the industry as newspapers that become tomorrow’s chip wrapping. They are overlooked for more eye catching work as the source of profound insight. Yet, each build produced is a block in the overall process of constructing the final product. The defects found in the build and corrected in successive builds ensure that the movement is forward even if this is, at times, only by small increments.

Software product builds result from the packaging of the compiled software and are delivered daily to the test group for use in their work. These ‘builds’ may be packaged in a number of ways, e.g. in the form of a self contained installer that is a program to be run by the user and which involves the selection of options to configure it or the simple unzipping of the compressed files from an archive file to a specific location on a computer locally or on a server. Accompanying the build may be further documentation detailing what new features have been implemented in this build, known problems or limitations, and data sets for use by Testing. In our field study the distribution of one particular
activity that of building the software to another location has affected other activities undertaken as part of the project that remained collocated. Software development in which some aspect of the work has been distributed (even a relatively discrete function such as building the software) presents specific challenges of its own.

In the simplified process model depicted below in Figure 1, we see the place occupied by the electronic artefacts of software product builds and defect tracking reports.

![Simplified model of Software Development Work](image)

**Figure 1: A Simplified model of Software Development Work**

Figure 1 shows the electronic artefacts of software product builds and defect tracking reports in the work world of the test and Development groups of the software product team. The Development team produces the Software Builds as a result of their coding work; these builds are the subject of testing work performed by the Test Team. If defects are found in the Builds then this leads to the creation of Defect Tracking Reports as part of the defecting tracking work. These defects must be worked on by the Development team as part of their defect resolution work and if they find a means to solve the problem it leads to changes in the code which in turns forms part of the next Software build.

In contrast to most people’s expectations, it is software builds not software code that are the currency of the quotidian work of software. Code repositories are designed so that the software code is stored so as to allow incremental changes to be to it made and, if necessary those changes can be rolled back if found to be unsuccessful. Software product builds are generated from the code continued in these repositories. However, code is not normally accessible to those outside the Development group including those who are testing the product. It is not code that is tested by the Testing team; rather, it is the behaviour of builds generated from this code that is being assessed.

### 1.3.3 Blocking defects in Software Development Process

*Blocking defects* are a particular subset of general software defects and are those defects, primarily in functionality, which prevent the Testing team from examining areas of the software build as per
their testing plans. This may often involve those features that have been just recently implemented by the Development team. In order to overcome these blockages our team has deployed patches, made up of a number of modified code files, to alter an installed build so that the defect is eliminated as quickly as possible in that particular build allowing testing to resume. The patch is made available in a controlled manner as the developer hands it over to the Deployment team by means of the defect tracking report. The Deployment team then copies these patched files onto the installed configuration. Once the patch has been tested and verified as unblocking the defect in the functionality by the Testing team it is checked into the code repository by the developer. Doing this ensures that it subsequently becomes part of the code stream and the next build where is it once again verified (as an ordinary fix is). Meanwhile the blocked testing work has already resumed its progress as it is no longer encumbered by the blockage.

1.4. The research question

1.4.1 Developing Software as part of Global project

There has been increased attention paid to software development in recent times, with a large proportion of this research being into the emerging area of Global Software Development (GSD). However, it has largely been conducted with a focus on the strategic management concerns of GSD (Sahay, Nicholson et al. 2003) (Krishna, Sahay et al. 2004) (Karolak 1999; Kobitzsch, Rombach et al. 2001). In the main this has been in the form of an overview of high level management and business concerns. Even where this literature has been self-labelled as being ‘tactical’ such as that of Carmel (Carmel and Agarwal 2001), it consists of, in truth, tactics and techniques for managers operating at a senior level. Often it results in instructions or guidelines or advice for best practice on how the work of those workers lower down the food chain should be organised, offering little if anything to the individual working on the code floor in such a distributed environment. Gasser (1986, pp 206) noted in his studies that a “detailed focus led us to findings that sometimes contradict the assumptions of those who overlook or assume away the importance of the micro level social dynamics of computing”. This suggests that the work ecology within organizations could benefit from a more narrow focus which would lead to more particular and useful generalizations.

There is a paucity of field work which given the highly diverse nature of software work means that the resultant literature is not necessarily representative or reflective of the current state of the art. We will redress the gap this presents within the literature through the course of our observations of the work environment of the local code floor. Our research addresses this gap by means of documenting in detail the practices of the everyday routine and redirects the focus of research away from the strategic issues of GSD and back towards the practical day to day concerns. “Work is a process, not just an event, and so it has a temporal organization. Some work occurs cyclically, while other work is unique
in form and function”, Gasser (1986, pp 209). The minutiae of this routine, which those doing it can see as dull and mundane, makes their work truly compelling and of abiding interest for us.

1.4.1 Research Question

We have chosen to take an approach which focuses on the day to day level of what individuals working in software development need to do to accomplish their work wherever they are located in the organisational structure. The focus of our interest takes in the effect of distribution on their local practice and how, in response, the use of electronic tools in software development evolves. We answer our research question by revealing and describing

How software professionals, working on a real software product, are aided by their work practices in dealing with the practical and local consequences of global distribution on a daily basis.

1.5. Key Concepts

Through the course of answering the research question in this thesis we discuss and touch on a range of associated topics. In examining practices and artefacts such as regular daily meetings, software builds, blockers’ documents and defect reports, we require some conceptual frames in order to make sense of what we see. These include how team members cope with their differing work priorities and the boundaries between their work worlds, how work practices are maintained and the role of visible and invisible work in that process, and how temporary work arounds are accommodated within a larger process. Our most significant frames in this thesis are that of boundary objects as a means to understand and discuss the use of the electronic artefacts and work arounds as a means to look at the work practices.

1.5.1 Boundary Objects

Divergent boundaries of the workers’ respective work worlds are thrown up between the Development team who are primarily focused on producing code, the Testing team who are focused on examining the builds produced from that code, and Management who are focused on ensuring that progress continues at a steady pace. Yet in terms of the work objects they are dealing with and the relationships between them there is much in common. They are all involved in the subtle changes to individual instances of the artefacts, both builds and defect report that allow them to continue to function usefully through the course of the project.

Boundary Objects is a term first coined in a seminal work by Star and Griesemer (Star and Griesemer 1989) who were studying paper based artefacts used by multiple groups or individuals for
different purposes but without impeding the use others could make of the same artefact. An artefact that was a Boundary Object, allowed them to communicate their needs as suited to their distinctly separate social and professional worlds. While referring to the same object it was recognised they have separate and perhaps quite divergent interest in the object. Such Boundary Objects facilitate people in making use of the same object with different intents.

Such objects are not without their limitations. Changes, whether unilateral or agreed, to a boundary object could result in the artefact ceasing to have those properties that allowed it to serve as a boundary object in the first place. Yet, these objects have to be in use and open to change, not static or isolated; for without human intercession they have no boundaries. We will in the course of this thesis be illustrating Boundary Objects in action.

Artefacts that straddle the boundaries of these work groups need to exhibit both flexibility and relative stability if they are to be of ongoing use to all groups during the course of the project. Artefacts generated as a matter of course as part of software work practices, such as defect tracking reports or the packaged builds of the software in which defects are uncovered, can knit the day to day work through the course of the overall project. This is moreover the case when so much else about the work environment is fluid. We are making use of Boundary Objects as a means to conceptualise how the defect reports and builds, with sufficient flexibility in both, allow workarounds to present themselves as part of the existing practices that allow for blocking defects to be unblocked. Both the content and use of the artefacts is extended and augmented to facilitate a successful working around of the defect that is blocking the testing work prior to that defect’s resolution. In our case study the boundary objects are altered as part of a process of working around the blocking defects.

1.5.2 Work Arouneds

Gasser (1986) found that when the functioning of the computing resources that are supporting work does not fit with the real world expectations such as say the turnaround time required to install new software build, then alterations to the practice are necessary. He also suggested a number of aspects which come into play when seeking to work around problems thrown up when the computer resources do not fit with the work practice. This contrasts to our study in which the computing work is the primary work, yet in both situations real world behaviour that was not anticipated in the design of the computing resources forms the focus of attention.

In addition to the problems, potential solutions to overcome any misfit can be present in the same system, providing people are willing to adapt their practices. By changing both the practice and software artefacts, these enhancements can facilitate finding a work around to the blocking problem.
Gasser’s (1986) framework sought to assist in analyzing computing and routine work and to be of use both in representing and reasoning about activity in multiactor systems. The aim was to further a better understanding of how to integrate the organisation of people and computers in which work was coordinated.

1.5.3 Emergence and Evolution of changes in work practice

We are looking at an ongoing project and are interested as Gasser (ibid, pp 206) was in “how circumstances persist and evolve, rather than why they exist in the first place.” The computing system used can continue to function more or less as designed in the face of much change. Yet the when work world no longer conforms to that for which the system was designed for, then change must come to the practice that was evolved around it also.

Echoing the work on boundary objects is the assertion that those artefacts, systems or tools need the intercession of human contact and shared understanding in order to have meaning - “meaning is created through the interactions among people and groups and during the process of performing work. The meaning of events and artifacts is not static or constant across participants or groups.” Gasser (1986, pp 209) The artefacts that have derived their form from the defect report and builds as boundary objects change day to day as they are altered to maintain their usefulness and relevance to the work that is happening. We believe that the practices in evidence at this site can benefit from being looked at through the dual lens of boundary objects and work arounds.

1.6. The environment in which the study was conducted

The field work for this thesis was conducted in a multinational organisation that has a sizeable subsidiary in Ireland employing several hundred people in its local software operation. The team who were the subject of our study were producing software that is an extension of a portal platform. This Portal software is produced by the same company at a number of sites across the globe. There were, also, with a number of individual free standing developers, supplying components to all the portal team and our team, working outside the site at locations in North America and the Middle East.

We had spent some time conducting an exploratory study prior to commencing the field study for this thesis. This allowed us to gain an initial understanding of the systems in use in situ, and helped us in focusing the study towards the practices of a team as they were working through their defect reports and on individual code fixes. We draw only on the results from this latter more substantive qualitative study of that portion of the software development project. We used a number of data gathering techniques, such as
Chapter Title - Introduction

- Semi-structured interviews with participants
- Shadowing testers, developers for periods of time
- Document analysis
- Participatory observations of meetings

The software product which we will refer to as Portal Learning Extension (PLE) 2.7 is a portal-based learning solution that allows any group manage their training programs and associated tasks more efficiently. Students can access a "just-in-time" online education environment that can be fully personalised. It is a standards-based learning system which improves the availability of learning resources, helping students to improve their skills in a flexible manner. The portal technology is also a product of the same organisation and it runs as an infrastructural platform between the operating system and the PLE.

This study of work practice centres on the use of builds and a defect tracking system by a software team comprising developers, testers and managers who are engaged in a software development project and forms the core of this thesis. While the majority of the team members of the PLE software component that forms the focus of the study are collocated, their work forms part of a larger product and the associated Portal team is distributed across a number of global locations.

1.7. Thesis Organisation

In Chapter Two, we will review some prior published work that we believe is relevant to our work and which has shaped and informed our thinking and approach, in respect of the nature of the domain and the nature of the workplace. We discuss how this literature influenced the way in which we later discuss the issues that arose at the field site. We also describe in general terms the setting and the concepts we draw on later in the thesis.

In Chapter Three, we detail what we look for in a field site for it to satisfy our interest and adequately answer our question, what we expected to find in such a setting and the methodological approach that would be best suited to such a site. We then discuss what criteria and practical considerations we had to take into account in selecting our site and how we revised our criteria. We provide a detailed description of how the study was conducted, how we gathered our field data and what would later inform how we would seek to present it.

In Chapter Four we present the data about the practices we encountered and observed partially by means of a narrative account of a meeting in which the artefacts were actively used by the team members. The field work as presented in this chapter specifically examines local work practices which
feature the ongoing and repeated manipulation of artefacts such as defect tracking reports and software product builds so as to overcome blockages in work progress. These are work practices that had been developed locally by the members of the software team in response to problems and restrictions resulting from the remote distribution of the software packaging and building process. These practices are reflective of the conduct of work on this project and we uncover in some considerable detail their documentation and the use of tools to discover, investigate and eliminate defects as part of software development.

In Chapter Five we tease apart these data and explore what it has to say about work practices, the reality of the emerging area of Global Software Development (GSD) and some of the common artefacts in software development. We discuss the maintenance of work practices through visible and less visible means. How when even small discrete aspects of work are distributed to the other side of the globe it can have local consequences that must be dealt with locally. And this must be done without impeding the work of others. This reflection is conducted in the light of the existing literature which we reviewed in Chapter Two.

Finally, in Chapter Six, we conclude the thesis by offering some conclusions from our work and potential avenues of future work. We present the potential of investigations to create a better understanding of the practices in the domain of software development as it is conducted today. We reflect on how this understanding may better inform the design of future generations of tools that are used to support such work, and how work practices can be allowed to evolve.

1.8. Chapter summary

In this introductory chapter, we have set out the background to the study, the gap in existing research we were seeking to fill, and what our motivation in conducting this research was. We gave some background on the researcher’s own relationship with the field site. We do this in order to provide a context for the overall theme of this thesis through a thorough exposition of the research question. The intention of the research from the outset was to examine the local work practices at a critical juncture in a software development project. After completing a preliminary analysis of the field site selected we chose to focus on the work required to resolve test blocking defects through the temporary modification of installed software builds by use of patched software files. Through this detailed immersive study we uncover a number of defect resolution practices in day-to-day use in software development when portions of the development work have been distributed.
Chapter 2 Everyday software work

2.1. Introduction

In most disciplines, knowledge is advanced through a recursive process involving observation, formation of theories based on the data so gathered and then experimentation or renewed study of the real world to validate those theories (Basili 1996). This journey to uncover the truth through research can be undertaken by many routes: experimentation, imaginative speculation, playing with new imagined or reproduced situations, or simply by observing an existing situation and documenting it. Following on from this with the formulation of theory we can create and communicate our understanding of the discipline to a wider audience. However, research orientated primarily towards theory generation from the outset can miss what surfaces unexpectedly in the field. A fixed mindset at the outset can blinker the researcher when an opportunity arises as something unexpectedly interesting crops up.

Over the course of this chapter we will put in context our work by means of existing, relevant, published literature that has informed our research, how we gathered our data, how we will later present and analyse it. Concerned as we are with the practices of software professionals using technological artefacts as a matter of routine, we must start by adequately situating our work through an acknowledgment of the extensive recent rise in interest in the varied aspects of global work in software development. We see in this interest a renewed and compellingly practical necessity for a fuller detailed description of the shifting environment of the working though and overcoming of wicked problems posed by distribution in its many forms, and the practices involved in helping blocked work to progress.

In the rest of this chapter we are studying software work in general, before progressing to the particular circumstances and phases of software development work. We then consider the usefulness of defect tracking reports and associated objects for analysis. Finally we introduce the concepts of boundaries between people’s work worlds and what role everyday objects can play in helping them to work around problems they encounter.
2.2. **Software Workplace studies**

This work follows in the steps of workplace studies which have a long tradition in the world of CSCW (Dittrich, John et al. 2007). The interest in studies from a social science perspective, employing more qualitative methods of enquiry, such as those by (Grinter 1996), (Dittrich, John et al. 2007), (Boden, Nett et al. 2008) and (de Souza and Redmiles 2008) has steadily increased in recent years. Our concern in this thesis is to ensure that scholarly treatments of software work in GSD be influenced by the extant body of conceptual and empirical work within CSCW. In particular we are drawn to its concern for understanding the details of work practice and how, in the light of this understanding, we should think about “the support requirements for cooperative work” (Bannon and Schmidt 1991).

Capturing all of this detail, motivation, and the context indeed the very essence of what is done at the very time it is being done requires that one be present when it is done and in a position to be able to capture it quickly and accurately. The work of Software development is not so readily amenable to examination by experimentation. While observation offers a better course for investigation it is not without its problems. Routine work practice can be difficult to capture faithfully as it is what is being done that is of interest, not what is written down as being done or that what has been written as should be done or what comes to be written after the fact as to what was done. For practitioners to inform the researcher in their own terms about what it is that they are doing and for them to do so as part of their normal work pattern enables that fidelity that we are searching for. The work of Orr (1986) highlighted the value of such an approach to the understanding of actual work in situ as it is related by practitioners.

There has been a trend toward more empirically based research of software development processes that has gained acceptance in the last few years in both software engineering and GSD. However, in software engineering, this tradition of going out into the field and dirtying one’s hands with the detail of work is significantly less well developed (Sim, Singer et al. 2001), (Basili 1996). For the most part these studies have involved either small-scale experiments or large-scale surveys of software development procedures and practices. While such approaches do have their merits, they also tend to fail to elucidate fully the actual software development practices in the field.

Our work advances the notion that the CSCW tradition of workplace studies can profitably contribute to the study of the varied workplaces and novel work practices of global software development both conceptually and methodologically. Hence, this is a CSCW-informed empirical study drawing on an ethnographically informed methodology, paying attention to the complex and
artful ways in which people manage their work both to complete their work tasks and simultaneously satisfy various organizational and technological constraints in the process of doing so.

2.2.1 Go global locally

An explicit engagement with these CSCW-perspectives is yet to be developed in the GSD literature. Given its fundamental concerns for the organization of distributed work, GSD should form a natural ‘home’ for many of the CSCW-orientated studies of work practice. We believe that there are a number of ways in which a CSCW perspective can illuminate problems within the emerging GSD field. These range from providing an analytic critique of various concepts being imported into the GSD field from the social and organizational sciences, to undertaking more qualitatively-driven empirical research studies.

For example, the very notion of ‘Global Software Development’ as a well-formed object of study can be held up for analysis. To what extent do the methods that already exist for studying distributed work practice need to be enhanced, revised, jumbled together, or discarded entirely when it comes to ‘Global Software Development’ (GSD) or can they be applied as they are? In what sense does a study satisfy some qualification for being deemed relevant to GSD? Is it a prerequisite to have work observable at sites around the world in order to rigorously study GSD? Must the structure of software project team involve roughly equal distribution of workers across the globe, or would having a few individuals scattered around the periphery while the vast bulk are collocated suffice?

In our response to these, we argue that ‘the global is in the local’, and thus we can investigate many of the issues relevant for GSD through a careful attention to the work practices of specifically local settings. Within the CSCW community, several studies have provided informed accounts of how software units set up and coordinate their work (Grinter 1995; Grinter 1996; Singer, Lethbridge et al. 1997; Grinter, Herbsleb et al. 1999; Herbsleb, Mockus et al. 2000). At the same time, various authors in CSCW have explored concepts which may assist in understanding the complex ways in which people share information and coordinate their activities — boundary objects (Star and Griesemer 1989; Lutters and Ackerman 2002) organizational memories (Kuutti and Bannon 1996) , common information spaces (Bannon and Bodker 1997) inter alia.

There are still voices in the software engineering community that criticize ‘studies presented from an ‘inside’ perspective(...) as revealing ‘bad practice’, i.e. bad application of existing methods” (Rönkkö, Dittrich et al. 2005). In part this may be due to bad practice attracting more attention, while good practice runs smoothly and without attracting much comment. We argue that empirical studies of software development teams using extensive field study methods allow for all practices to be exposed. And through an in-depth understanding being arrived at of what software development teams actually
do in practice this could inform software process improvement efforts. We have set out to study the challenges of global software development through the eyes of a development team who practice software development on a global, distributed basis in their daily work rhythms and routines. We examined the global dependencies through the perspective of an Irish team, aiming to understand their practices, and noting the kinds of support tools these people use in their work. Our time spent in the field has demonstrated that flexibility, creativity and commitment are vital at a local level in order to keep the work flowing, while tools, norms and procedures as articulation mechanisms are important for coordinating the work done locally with the work performed in other locations.

In this thesis, we are specifically interested in how field material highlights the importance for the engineering teams involved of ensuring that the work continues to ‘flow’ even in the most pressing of situations. Note that this is very distinct from arguing for a ‘workflow system’ as it is the human agents within specific settings that determine how to make the work flow, and indeed, in certain instances innovate, patch and workaround in order to ensure that there is an appropriate continuous flow of work. Our particular focus in this thesis concerns how globally distributed teams resolve software defects in the face of the consequence of distribution. The process of performing and tracking software tests and defect tracking, ‘bug tracking’ as it is termed, is critical to software engineering. It encompasses the documentation of the test results, reproduction of the defect by development and testing and its resolution and verification of the resolution. It brings together people with often distinct interests and is an activity in which the organization of cooperative work in a project becomes conspicuous.

We do not claim that this observation and perspective on work practice is original, as it has become well-accepted within the CSCW field (Bowers, Button et al. 1995). However, we do assert that our field study shows up this issue in the domain of software development in a way not dealt with in any great depth in either the CSCW or Software Engineering literature to date. As we shall see, software engineers commonly work with a variety of technologies picked from the heterogeneity surrounding them which they both ‘work with and around’ in varied ways to keep the work flowing. In recent years, several studies have emphasized the role of social, cultural and organizational problems in working with distributed software teams (Carmel and Agarwal 2001; Herbsleb and Moitra 2001; Sahay, Nicholson et al. 2003; Krishna, Sahay et al. 2004).

Our contention is that such problems given their human centric and practical nature, need to be informed empirically. What the social, the cultural and the organizational mean for each of the participants in project work, how these affairs affect work and the success or otherwise of projects, are empirical topics. While software engineering, more generally, has been studied empirically from a social scientific point of view for some time (consider, for example, the work of (Button and Sharrock 1996), rather less attention has been devoted to the particular nuances of the numerous forms globally
distributed development teams may take on. Even if these nuances exist only temporarily in response to a set of circumstances that are not repeated or which cease over time to be valid.

Our intention, as we have stated is to contribute to the recent studies on globally distributed software development that are emerging within CSCW (Halverson, Ellis et al. 2006; Holmström, Fitzgerald et al. 2006) to deepen the influence CSCW perspectives can have on GSD. In addition to adding to the corpus of CSCW studies on software engineering, our work seeks to deepen the benefits GSD can obtain from CSCW-derived perspectives on distributed work. Part of our story here is of how the mundane work activities involved in defect or ‘bug’ tracking and defect resolution occasionally requires innovative responses from the teams. This happens even as these local innovations are viewed locally as just forms of ‘getting the job done’, simply doing what is required to keep the overall work activity on track.

2.3. Rhythms and cycles of Real Work

We are primarily interested in the conduct of the real everyday work of software development, in preference to the performance of work under fixed experimental conditions with students or the simulation of work settings, or surveys that rely solely on those with time on their hands to complete them.

People working together have not just their own individual work to deal with, but their collective work as part of a team, the work of others combined with theirs, and cumulatively, their own prior work combined with their work done today and their work planned for tomorrow. The interdependent nature of their work, shackled though it is to a larger shared goal still allows each of them to approach that work in subtly different ways, as fits their respective role whether testing, coding, design, management or other. The local organisation of work comes to be orchestrated through how such complex forms of organisation are sustained or altered over time.

The work, of the team and the individual, moves forward in cycles, from one day to the next, from one instance of the product to another. The rhythms of this cycle and how the work practices that are attuned to those rhythms may act to help or hinder the maintenance of these rhythms, so in turn shaping the tools may act to help or hinder the maintenance of these rhythms. These kinds of efforts that people make to locally organise their work can’t be ‘full on’ all the time whether working with the assistance of technology or having to struggle against it. There will too, be naturally occurring ebbs and flows over the course of the day, over the course of weeks, over the course of work cycles. There is a considerable temporality involved in how practice unfolds from day to day.

When there are high levels of staff turnover in the project team there is an increased need to document how everything is ordered because people lack the tacit contextual knowledge “that this is
the way we do it round here” (Walsham 2001). Alternatively when there is a steady core of people who have been working together for quite some time, the potential for this problem to exist may not be realised as they will know what to do next. That knowledge is often tied up in the practices that are common across product group to product group, even where the product itself has changed. Just how such practices come to be instantiated and maintained thereafter in such situations is a core concern for us.

With so many rhythms being beaten out in parallel, from day to day, week to week, build to build it is necessary that some practical means is found to keep work synchronous and act as a metronome for all involved. Meetings held at a regular time, with a clearly acknowledged purpose and in a consistent setting do this, with documentation that effectively records and fixes the events and outcomes of the meeting in time while linking it to both prior and future meetings. Such documentation will form the core of the next document and is built upon past documents. Like a guide rope that assists the climbing party. The defect found in this build is in all the other builds until shown not to be but it remains linked to the one in which it is first found.

This is work practice as Grinter (1995) speaks of it providing for its own modification when faced with blockages “…, these procedures do not work when the contingencies of their work case them to deviate from the model…In this case the system provides an acceptable work around for its own limitations. Sometimes developers deviate from the model because it does not support their work“. Grinter goes on to describe how both the software work practice and the technological supports for the software work activities must either bend to accommodate one another. They must be continually adjusting to what is happening on the ground or risk one or other of them being sidelined, pushed to the margins, undermined and ultimately cast aside as broken by those using it. This work to accommodate daily changes in circumstances, to ensure that the practice and the tools continues ‘to fit’ one another is reflected in how workers cope with the daily ebbs and flows of their work. This takes place against the backdrop of the general difficulties of managing software work for reasons as elucidated by Grinter (1995) “First, developers can easily change code. Second, the modifications can affect the behavior of the entire system because of the interdependencies among modules. Third, because teams develop software the changes one person makes often impact the work of others. Configuration management procedures focus on controlling developers’ abilities to alter code.”

2.3.1 Rhythms between phases

Software development work involves many phases. In seeking to understand in depth what it is that people do day to day we need look at an intense time critical phase, in this case, the testing phase after specification and design work has largely been completed and before the final regression testing takes place prior to the release of the finished product. Such a phase comes in many different formats
but some aspects are common; testing work will be ongoing while coding work continues to implement features from the design.

In examining one phase we are conscious that no single phase exists in isolation from another and that the events of the previous phases and the requirements of future phases impinge upon it. Members of the project team need to communicate amongst themselves and also with those involved in both the previous and subsequent stages in the process. Such work, when undertaken by people in groups, and when organised into mutual dependent stages, means that the team members are interdependent both within and across the phases. Along with the Testing and Development phases, there are milestones and feature completion stages as each build is produced and tested.

Software work does not necessarily proceed at a steady pace; rather, individual tasks may proceed in a stop/start fashion. The work may be done by shift assignments and also be held in abeyance for many weeks, or even months. Periods of more intense activity, with many people making changes in a single week are followed by fallow periods of month or more when nothing occurs in that work area: design, testing or coding. It can seem from the tracking system that is being used to monitor the progress of the project work that the defect is not a focus of anyone’s efforts. This happens because any work conducted away from the defect report is invisible until such time as it is placed in the defect report, thus making it public.

The time that may elapse in a large software project between a particular defect being found by testing and the associated defect report being created or ‘raised’ and being first tackled by a developer has an impact on the defect resolution process. It is on the informational sufficiency of the initial report that it is revisited and made real once more. The reproduction and reliving of the defect is part of a process that testers and developers are engaged in, perhaps daily at some points, by which the defect is renewed each time.

2.4. Software Development work in public view

Software work is not the solitary activity it can often appear on the surface to be; though software development can certainly appear to the casual observer to be quite solitary. We are familiar with the image of an individual alone at their desk, typing periodically at a keyboard, all the while looking intently at a screen, oblivious to the world around them. The reality of the development of almost all software as currently organised is of a high interdependent work environment. The person who seems so alone at their desk is actually interacting and highly dependent on many other co-workers, both collocated and at other sites, interaction that is needed in order for their own work to be productive.

Developing software for corporate environments is now simply too large and complex an undertaking to even consider having individuals work alone on these projects. It involves the creation of millions of lines of code, which have to be deployed on an ever-increasing array of hardware
configurations. Ever present are the commercial pressures to deliver the product in a timely manner. Teams comprising of even more than a few dozen will, by default, involve the physical dispersal of people such that they must move about in order to engage in face-to-face discussion of issues that arise. The overall character of such large-scale software development work involves an explicit and implicit division of labour, with the attendant increased specialisation of expertise in testing, coding, specification, installation, configuration, and overall management of the project. Working like this leads to specialisation which leads to an increased distance between co-workers and hence the further distribution of work.

Due to larger sized teams there is increased specialisation and division of labour. This means that each team member is not able, nor are they equally equipped, to focus on the same aspects of a problem in exactly the same way. However, they will also share some tasks in common such as the need to be able to reproduce a defect. The developer or developers who work on fixing defects require certain information so that they can reproduce the defect. The tester or other members of the test team will need to confirm that the fix proposed by the developer has resolved the problem, by recreating the circumstances again at a later point. With many scripts to be run and, in some cases, the passage of many weeks or even months between the time the defect was uncovered and when it is resolved, the person who creates the defect report may not remember the circumstances in detail. They require support in recreating the circumstances in which the defect occurred. In effect, a defect report serves not alone as a message to other people but to the person themselves at some as yet unknown later point. Such communication across time between future and past points is not unusual in many work settings including software development.

While developers can often appear to conduct their work of coding in relative isolation, the work is not conducted in an isolated silo. When actions that they perform or the outcomes of those actions, that are might otherwise be occurring in relative privacy, are performed more publicly, it is assisting in the development of a cohesive team. The value of acting in public has been demonstrated by research whereby actions, such as the modifications of electronic documents or the addition or removal of file content embedded in such documents is conducted in a more accessible and transparent fashion (Gutwin, Penner et al. 2004). This allows others to more readily know, in detail, the work that one is doing without requiring additional effort on the part of the person performing the action separate to the actual conduct of the original action. The value of others especially co-workers knowing who is doing precisely what and when they are doing it, in software development was further highlighted by (Perry, Staudenmayer et al. 1994; Ehrlich and Chang 2007). Their work shows that such acting in public serves in both establishing and thereafter continuing to maintain a common knowledge of what has been done to date, what is currently being done right now and what it is that it planned to be done over an upcoming period. This assists in creating and supporting the sense of unity or ‘teamness’ that is necessary for group work to be successful.
When software development is distributed, having a defect tracking application that is (a) equally accessible to all team members irrespective of their role and (b) is shared across all sites, is extremely important for a number of reasons. In recent research (Casey and Richardson 2003) those reasons have been highlighted as being even more pertinent in the global environment. The sharing of information through such common systems also serves as a means initially to introduce and later to further inculcate common standards across an organisation which is engaged in collaborative work (Kobitzsch, Rombach et al. 2001).

Such a reliance on their interdependent interactions paradoxically results in an increased requirement for them to occur between distributed groups just as it comes more difficult to achieve. To alleviate the detrimental impact of this dependency on the success of the distribution (Carmel and Agarwal 2001; Mockus and Weiss 2001; Moitra 2008) sought to identify particularly highly dependent activity and associated practices. Then they firstly suggested isolating them from other practices and activities and then, if possible, finding the means for their complete removal from the situation. However, if it is not possible to remove them entirely, their prescriptive solution is that attempts should be made to reduce the degree of dependency present in the situation; others have proposed strategies to reduce the distances involved or the impact of the distance. These research efforts such as (Herbsleb, Mockus et al. 2000; Carmel and Agarwal 2001; Mockus and Weiss 2001) have sought to focus on a consideration of the impact of this increased inter-reliance on members of remote groups and working towards a reduction in the dependency.

2.5. Dealing with the shifting sands of Contingences

With such a high degree of interdependence amongst the various individual workers and their work functions there comes a considerable vulnerability to shifts and changes in the state of the material product being worked with. Progress is often contingent on the stability of the work product itself and the particulars of the associated set of circumstances surrounding it. This stability is something that is uncommon in the world of software development.

Ongoing meetings between the clients and those designing the software, or amongst those same designers themselves, can lead to significant changes in specifications, in deadlines, or in the available resources: even, ultimately in whether the project is to continue to completion. In order to ensure that work does not become blocked and continues to progress it is necessary that additional articulation work is done to coordinate the core work involved in design, coding, testing etc. This work frequently involves the use of more static documents including but not confined to; specification documents, flow charts, regular meeting agendas, visual models based on the Unified Modelling Language (UML) or defect tracking reports which allow the work to be represented in a visually readily accessible manner. Everyone working with the team will deal with these documents to varying degrees. As the project
progresses, those with managerial responsibility have the most dealings with these documents, many dozens of times per day, in order to track the status and progress of the entire project.

Enveloped as the work of the project team is in this maelstrom of change and instability, it is not unexpected that unanticipated blockages can arise. Such defects and blockages form part of the contingencies of real work as envisaged by Gerson and Star (1986), contingencies that are not always foreseeable in detail, given that by their nature they are unknown at the outset. When work becomes blocked as a result of unforeseen circumstances, it is frequently necessary that the people with various functional responsibilities from Development, Test and Management must come together in order to discuss and tease out in great detail what the revised situation is. In seeking a solution to a problem, it is necessary first to agree what the situation is now, then to detail the nature of the problem before proceeding to consider what potential action might be taken. This equally holds true for the selective deployment of any solutions in that their impact needs to be particular and confined to the defect, and does not affect other areas such that they must be revisited. This process of drilling down through the information to form a mutually agreed understanding of the nature of the problem is assisted by all those involved being simultaneously able to view the same information irrespective of their location. This is the case whether that information is in a defect report or other electronic document. This serves to secure stability in the midst of considerable flux by means of this agreed frame of reference, knowing the problem in detail in turn leads to knowing the solution.

2.5.1 Propagating practice

One of the questions arising from Star and Griesemer (1989) was –does what forms the input to or cause of a problem lead to a working around being potentially developed? Can the characteristics peculiar to problem lead to its own resolution being more visible or inaccessible to those seeking to resolve it. Grinter (1995) separately describes a software work situation where the contingent natures of tasks that have previously been successfully addressed are encountered anew. The defect tracking application facilitates people in learning from and re-implementing the past resolutions approach insofar as it had been documented. This, of course, required that the defect report documents had captured all the work involved in fixing the problem in a satisfactory fashion. Also, they found that it was necessary that the defect tracking application allowed for the reports to be organised in such a fashion that the reports could be easily retrieved, and that supporting work was organised appropriately.

Such usage shows that successful solutions can propagate themselves over time through the history of an organisation, acting to reinforce the effectiveness of good practice, while bad outcomes make barriers for themselves. In this way work practice can support or form barriers to efforts to conduct work in an alternative manner beyond the immediate project to which they are first applied.
As a mechanism for the instantiation of practice as highlighted by Grinter (1995), it allows certain work practices to be perpetuated while preventing others from being adopted in the first place. “When the processes do not match the realities of the work then two results may occur, developers may bend the model to work with it or reject it entirely” (Grinter 1995). Their study, (of a development team that had created its own configuration management system to be used in its own development) had certain unique qualities given that the technology being used was open and subject to ongoing revision. In our study, the organisation of which the team was part had also created its own defect tracking system and had considerable latitude in modifying the defect tracking system, provided it was agreeable to all those other teams using it.

2.5.2 The fidelity of defect tracking

In examining what people engaged in software development work actually do Lethbridge and Singer et al (2003) through their accounts of what practitioners related to them of how software work unfolds indicated that developers view defect tracking applications as important repositories of historical information. A key finding of theirs was that software developers, whether consciously or subconsciously, make a value judgement placing defect tracking applications in a primary role of documentation which is worthwhile for them to maintain. They see it as worthwhile both for their own work as developers and for the work of others. Their research, based primarily on interviews with developers, found that their practice was to work to ensure that defect tracking reports in preference to the other documentation such as the specification documents contained the most up to date information. The documentation in the defect tracking applications is not alone recognised as more detailed and relevant to their work but expected to be updated in a significantly more timely fashion than other forms of documentation. The defect tracking reports thus form an accurate and up to date account of all the work both successful and unsuccessful that has been performed to date in an attempt to resolve defects.

This fidelity by the software professionals in the defect reports to current state of affairs stands in marked contrast to the lack of maintenance of other documentary records of the software project be they specification documents, prototypes or user assistance guides. This occurs, as Lethbridge et al (Lethbridge, Singer et al. 2003) noted, not because software developers will not create documentation but rather, because they prioritise the work of creating and maintaining that documentation which they believe will be of benefit to them directly. The reports are used by them to record what they have done and how it had come about, they could also see in the defect report what the testers had done in association with this defect. Defect tracking reports prove themselves on a daily basis to be a form of documentation that is vital to software developers.
2.6. **Global Software Development**

This increase in distributed international software development has precipitated the emergence of a new arena for research within the fields of software engineering and information systems: that of Global Software Development (GSD). GSD has become a stable, defined sub-field within the software engineering community, with its own conferences, workshops and journal special issues (Battin, Crocker et al. 2001; Carmel and Agarwal 2001; Ebert and Never 2001; Herbsleb and Moitra 2001; Kobitzsch, Rombach et al. 2001; Mockus and Weiss 2001). It has a characteristic emphasis on the global aspect of the development work, and increasing concern over issues of control, coordination and communication among a dispersed global workforce,

In the course of our work we raise some significant questions specifically concerning the nature of the field that has come to be termed “Global Software Development” (GSD). The label is seen as denoting a new field of enquiry, with its own unique set of concepts and methods, but there is room for some debate starting with the label of “Global Software Development”. Our contention is that one must be careful in positing, or making the assumption, that the extension of a distributed practice from local sites to a global level necessarily implies that a whole new set of concepts is required. Thus one of our messages is that, the global is readily apparent in the local, in the sense that methods and concepts developed to study the local can, in virtually all instances, also be applied to the global.

2.6.1 **Pressures behind GSD**

Software development is a complex work activity that is carried out in a variety of settings around the world. Despite this heterogeneity, software development, as a form of work, shares with many other forms of work a series of pressures. Examples of these pressures are the need for increased flexibility, rapid response, effective coordination and distribution of global resources, and the need for ad-hoc re-organization of teams. These pressures come from the large-scale organizational re-structuring of recent years, as organizations adapt their internal structures and functions to an increasingly demanding - and at times chaotic- global environment. The need to harness the global resources of companies has increased the necessity of organising and deploying effective distributed teams of workers, across large sites and even countries. The increasing power and flexibility of modern communications technology has provided important infrastructure to support these distributed teams. Thus, at one level, the increasingly distributed nature of software development work can be viewed as a part of this larger organizational re-structuring going on in many sectors. According to Moitra (2008) the key motivators for organizations to distribute their software development include,
• economic (cost-saving) motives
• leveraging knowledge diversity
• exploiting existing capabilities and developing new ones
• supporting cross-location interactions and synergy
• harnessing the capability of partners

2.6.2 Limitations of GSD Research

There have been a number of self imposed limitations in the type of software work considered noteworthy for prior research. Of considerable interest to researchers, perhaps as a result of difficulties in gaining access to workers on the coal face or what we might term the “code face”, has been the strategic management concerns of GSD (Sahay, Nicholson et al. 2003) (Krishna, Sahay et al. 2004) (Karolak 1999; Kobitzsch, Rombach et al. 2001). This stands in marked contrast with taking the approach of focusing on the more mundane day to day level of what it is individuals working in software development need to do to accomplish their work.

Research studies undertaken on Global Software Development have mainly focused on “robust models, methods, processes and tools” to make the organization and execution of global software work more effective and efficient (Damian and Moitra 2006). However, other studies on GSD have chosen to focus on more person-centric issues such as communication, coordination and control, which are seen as major challenges (Carmel and Agarwal 2001; Holmström, Fitzgerald et al. 2006). This perspective has changed during recent years, and the role of human actors in actually accomplishing work and implementing work processes is now widely acknowledged. It has been argued that in order to understand and overcome the problems faced in software engineering today and how they are addressed, we need to further understand “the behaviour of software engineers, development teams, and organizations” (Herbsleb 2007).

In a global environment, the work of software development may end up distributed in a number of ways using the various phases in the development process as natural breakpoints; the work may be located in different places on this basis. A simple and common example of this approach is the outsourcing of a parallel phase of the work such as maintenance, testing or coding work (Mockus and Weiss 2001).

2.6.3 GSD summary

The forces driving the growth in GSD are varied and numerous. Financial pressures have resulted in organisations seeking lower cost bases, from a desire to increase market share, and from difficulties
encountered in accessing specialised professional expertise (Carmel and Agarwal 2001; Herbsleb and Moitra 2001). This new context of Global Software Development places even greater reliance on the interactions between the members of the team engaged in the work. Such distribution across global boundaries has been found to present a number of problems (Herbsleb, Mockus et al. 2000; Ebert and Never 2001; Heeks, Krishna et al. 2001; Kobitzsch, Rombach et al. 2001), such as ensuring that people have up-to-date information available to them, that work can be broken into manageable chunks and that common tools are in use to support an element of community. Such problems mean that people engaged in global software development find it difficult to coordinate their work. Considerable additional effort is required to stay up-to-date with the work of others and, to effectively communicate the current state of their own work. This is in addition to the existing challenges of software work.

### 2.7. Work practice across Boundaries

#### 2.7.1 Specialisation in Software work

The field of CSCW has long concerned itself with the arrangements necessary for such cooperative work to take place. Bannon and Schmidt’s early definition of CSCW was that “CSCW should be conceived of as an endeavour to understand the nature and requirements of cooperative work with the objective of designing computer-based technologies for cooperative work arrangements”. Cooperation inherently involves multiple participants. In software work there are divisions of labour between those who design, test and manage. In some cases the same individual undertakes this work at different times: the designers of the architecture may code some of the product, the test manager may when time is short running some test scripts, a member of the test team may chair a status meeting. Yet even this overlapping where it occurs merely serves to emphasis the distinct roles that exist.

With the considerable specialisation necessary in software development work, from the skills required for writing test cases, documenting them and various technical competences, to those needed to code the product and manage flow of the work; the workday world of each person involved can diverge quite readily. As the work they do differs so does the unfolding of their work day and world. Within such a specialised environment with distribution as a given the boundaries may necessarily only be crossed and re-crossed in those documentary artefacts. In their use of these documents the divergence of their work worlds is at once highlighted and accommodated. In our case common to most software organisations, we have three communities: the Testing community, the developers and the managers overseeing the work of both.


2.7.2 Boundaries of Work Worlds

Boundary Objects, as first coined in the seminal work by Star and Griesemer (Star and Griesemer 1989) who were studying how paper-based artefacts were used by multiple groups or individuals, can be used for different purposes but without impeding the use others could make of the same artefact. Boundary Objects suggest themselves to us as a means to allow us to conceptualise documentation, artefacts and other electronic currency from software work. The specification and design documents, defect reports, software code file and builds are mutually dependent, defects are found in builds and builds comprise of code written for feature work, code that is intended to address those defects as documented in the defect reports. Defects exist and are found in specific builds, and the accounts of their journey from uncovering to resolution in the defect reports are in turn tied tightly to those same builds the defects are found in. The defect reports are a documentary instantiation of the defect that exists in the build, and the code in the build is part of the documentary instantiation of the work done to resolve the defect. This documentary instantiation also encompasses the defect report and as we will show in some cases the code can come to reside there too.

The boundary object’s design must allow for successful alteration in its use to occur that will accommodate the situated needs of testers, developers and managers just as it did for the 19th and early 20th century hunters and museum curators (Star and Griesemer 1989). Doing so meant that others who came long after them were still able to make use of the forms describing the captured and catalogued flora and fauna. There was sufficient malleability in the forms that let them be modified without becoming unusable by others. These objects that work across the boundaries of the work worlds of a number of groups given us a concrete fix in what is a very ephemeral world.

These boundary objects, in this case documentary forms, allow those using them to communicate their own needs suited to their distinctly separate social and professional worlds though it was recognised that they have separate and perhaps temporarily quite divergent interests in the object. Such objects are not without their limitations; changes, especially when made unilaterally can alter the object’s ability to be used by others, whereas changes that are agreed and commonly understood can reduce the risk of a boundary object ceasing to have those properties that allowed it to function adequately in the first place. Yet they cannot be immune from change. These objects have to be in use, not static or isolated or on their own. In the absence of any human intercession the objects have no work worlds to form boundaries. The nature of the boundaries themselves is reflective of what is bound by them they are social, work-world related and temporal not merely physical.

We are examining a coupled set of artefacts, builds and defect reports, altered over the course of the group’s efforts to resolve the blocking defects. We are especially focused on their respective alteration by means of the additional deployment of patches via addition to the DTR as well as to the
builds once they’ve been installed and made available to the Testing team. We have to be clear about what form that the software takes is more important for us, software code or software builds? We are interesting in software builds in part due to the lack of prior research attending to them but also as they are concrete objects in daily use by Testing and Development. The builds in turn result in the production or propagation of other types of artefacts such as defect reports that we are looking at. Without the software builds running on machines there would be no defects to find.

### 2.8. Looking Beyond Collocated Practice

In looking at work practice, we must reiterate that we are looking at what people do as a matter of routine. Whether that routine is mandated from outside the group or what they have established themselves as a result of experience of prior work of a similar nature. They continue to choose to use these practices because they hold them to be of value. Mandated practice that is not followed for whatever reason is not of interest to us.

In examining GSD work in close detail, we must be careful to avoid too narrow an adherence to the prescriptive notion of there being a single best way to do something. There exists a tendency to view a collocated situation as being the norm for all work including software development work. Thus the collocated situation is the optimum work situation to be aimed for and the reference point against which all other forms are to be measured (Hollan and Stornetta 1992). However, they also highlighted such a view as problematic and ultimately self limiting in that it obscures the potential of means of communication that are not possible when face to face is available. Researchers in global software development (Battin, Crocker et al. 2001; Carmel and Agarwal 2001; Ebert and Never 2001; Heeks, Krishna et al. 2001; Herbsleb and Moitra 2001; Kobitzsch, Rombach et al. 2001; Mockus and Weiss 2001) have looked at the consequences of how distributed situation diverges from being collocated. They then seek to remove these divergences so as to mirror or as closely as possible replicate the collocated situation. The intention is to return to the norm of collocation, yet this is not always either possible or even desirable.

While the norm is to seek out means to mimic or reproduce collocated actions when distribution is part of the work environment, it is subject to the constraints explored by (Hollan and Stornetta 1992; Storey, Čubranić et al. 2005) that some of the benefits which a newly distributed environment and the associated technology can bring are missed out on. In a project which is globally distributed face-to-face interactions do not so readily exist and, consequentially, communications by technological means or as part of the practice of particular tool usage take on even more importance. When that usage extends beyond the confines of the solely distributed to include the collocated software development team members in combination with their face-to-face communication; we are presented with new opportunities to alter work practices. In the distributed global software development situation with the
near complete absence of the physical face-to-face contact the reliance on these tools is increased. They change from being a potential alternative to face-to-face contact to being the sole means of communication available, and their use is, as such, essential.

2.9. The effect of work arounds on blockages in keeping work flowing

When impeded at work one has to find a means to deal with the impediment whether it is the result of a) a limitation in the design that needs to be worked around or b) an unexpected blockage that arises because of changed circumstances. However the impediment to be worked around isn’t always the root cause of the work around, much as a fallen tree is to be worked around but the storm or whatever felled it is the cause.

The defect report shows the progressive alterations that the contributors have made to it. This allows the reader to build a sense of the history of the defect, its ups and downs and also of the technical competencies of those involved. This extends their personal work network beyond their collocated colleagues by means of the ad-hoc temporary groups and subgroups that form around a particular work item such as a defect tracking report to work in common (Ehrlich and Chang 2007). By means of the data that accumulates in the logs of changes made to the defects tracking reports such personal networks showing the team stability, visibility of the work for the team members (Horn, Finholt et al. 2004) may be traced out by later readers of the DTRs.

When the functioning of the computing resources that are supporting work does not fit with the real world expectations such as say the turnaround time required to install a new software build, Gasser (Gasser 1986) found that alterations to the practice are necessary. Gasser’s (1986) framework sought to assist in analyzing computing and routine work and to be of use both in representing and reasoning about activity in multiactor systems with the aim of furthering a better understanding of how to integrate the organisation of people and computers in which work was coordinated.

Gasser (1986) was looking at a work setting where the computing work was considered separate to and rather supportive of the core work activity which was non-computing in nature. This compares to our field study situation where computing activity is central to the work being undertaken. Gasser (1986) suggested a number of aspects that come into play when seeking to work around problems thrown up when the computer resources do not fit with the work practice. In contrast to our study in which the computing work is the primary work, yet in both situations real world behaviour that was not anticipated in the design of the computing resources forms the focus of attention. In addition to the problems, potential solutions to overcome any misfit can be present in the same system, providing people are willing to adapt their practices.
In contrast the domain for our study is where computing work is the primary work, yet in both situations real world behaviour that was not anticipated in the design of the computing resources forms the focus of our attention. In addition to the problems, the potential solutions to overcome any misfit or blockage can be present in the same system providing people are willing to adapt their practices to accommodation these work arounds. Change can be made in both the practice and software artefacts, these enhancements can help the workers to find a work around to the blocking problem.

In looking at an ongoing project we are similarly interested in “how circumstances persist and evolve, rather than why they exist in the first place.” Gasser (Gasser 1986) The system can function as designed but the world no longer conforms to that which the system was designed for nor the practice that was evolved around it. Gasser (1986) noted in his studies that the work ecology within an organization could yield up perhaps more narrow rather but also more particular and useful generalizations. Similar to Gasser, we are intent studying the persistence and evolution of the work environment of which the work practice is a significant part. We seek answers by studying the discrete, fine grain interactions as people try to retain control in the face of rapidly changing work demands and evolving technologies. However, we extend and build on the earlier work by looking at an environment where the computing work is not taking place separately or distinctly from the real work.

Gasser (1986) also echoes the work on Boundary objects by comments on those artefacts, systems or tools which require the intervention of human contact and shared understanding in order to have meaning “...meaning is created through the interactions among people and groups and during the process of performing work. The meaning of events and artifacts is not static or constant across participants or groups.”

The notions of fitting, augmentation and working around the blockage by means of altering the computing support for work is dependent on maintaining mutual agreement on the part of all participants in order to progress (Gasser 1986). The nature of the alterations is part of work as a contingent process where it is impracticable, nigh on impossible to predict the broad spectrum of contingencies every team member must deal with each and every day. The world of work is not nor should it be expected to be limited to comply with the original design of the computing resources in use.

It must not be forgotten that the work around functions as a means to an end with the overall goal remaining central to its use “...the concept is used to explain how one actor is able to adjust a technology to meet his or her particular needs or goals” (Pollock 2005). While sharing the same overall project goal and vision, that of delivering a function software product, each individual worker
has their own more immediate goals from day to day and build to build. A work around involves more that simply altering a tool, in tandem the considerable work to be performed in maintaining changes in work practices.

2.10. **Visibility in the Daily work world**

This is a fluid and dynamic work environment and we have many practices rubbing up against one another. Some work practices such as that involved in completing testing work may be impeded as a consequence of another practice such as centralised builds and code integration. A practice for example centralised builds that has many positive effects such as reducing build breakages and code integration problems may also impose limitations on local behaviour in the form of deadlines, or restrictions in when code may be altered. These restrictions can affect the turn-around time between code changes made and the code being available to Testing in the form of a build.

Gerson and Star (1986) noted that while there is heterogeneity in the range of software work involved and temporary, but necessary, divergence in short term goals and horizons, the larger common team goal and longer term horizon always looms large over their endeavour. In contrast with the position of Boundary Objects in Star and Griesemer (Star and Griesemer 1989) where that the difference arises when their worlds intersect, here it is when their world’s diverge temporarily but in a repeated and predictable manner that the boundaries are uncovered.

As Star and Griesemer (1989) state “*because these new objects and methods mean different things in different worlds, actors are faced with the task of reconciling these meanings if they wish to cooperate. This reconciliation requires substantial labour on everyone's part.*” (ibid) The labour involved in this daily reconciliation is frequently invisible to those working in other areas as well as to those seeking to study software work. Inconsistency and code conflicts are a frequent by-product of distributed software development, particular a problem to be surmounted and the distribution crosses time zones leading to problems in resolving the conflicts due to mismatched work patterns when one person is sleeping when the other is awake and working. Reducing or shifting this immediate dependency does not necessarily eliminate it but contains it within a more confined space.

Work that is invisible to others can be made more visible by relatively simple means. This can be assisted by the adoption of a classification system that ensures that the differences between one course of action and another are made clear. (Bowker and Star 1999) Identifying one type of defect as being different from others by the use of a name such as for example “blocker” makes clear that the work practices associated with it will be different. It may be that special attention will need to be paid in aspects of the resolution practices or that certain practices will be tweaked.
The individual activities of design and coding, testing and higher level management are accessible and attractive to study. Where all these activities come together it is more difficult to tease them apart. However this underappreciated work is deeply reflective of those characteristics of software work day to day. Star and Griesemer (1989) referred to such labour in establishing the criticality of “the flow of objects and concepts through the network of participating allies and social worlds”. Just the right sort of environment (physical & organisational setting) needs to be in place in order for practices to foster such a flow to emerge. It requires regular, planned and frequent contact in a suitable form in order for them to, “translate, negotiate, debate, triangulate and simplify in order to work together.” (ibid)

Also of interest is the broader non-computing world of work where Gerson-Star (Gerson and Star 1986) talk about the nature of contingency and the role played by differing individual viewpoints in the effort to reconcile them within a single static representational object that is useful to those different viewpoints. Whether by doctors, customers, patients, medical reviewers, claim adjusters, or administrators, much work must be done in order for the procedures and categorisation to work effectively for all. This is work which is commonly hidden from each grouping. The strategy used has its own interim aims of administration, which are temporary and locally applied in order to reconcile them into its single representation.

Yet, one can only reconcile with what is known and this means these representations must in turn be visible, and of necessity built upon and drawn from, the historical experience and local circumstances that people find each day. In this way ‘specialised quirks’ (ibid) are utilised to ensure that work continues progresses and any contingences that have arisen are dealt with. Much of the historical experience is tacit and difficult to capture, or report or indeed transfer outside the particular local setting. Through acting more explicitly via electronic systems the actions are captured naturally as part and parcel of that work and made explicit. This externalisation has the added benefit of making access to empirical data by researchers easier.

2.11. Chapter summary

In this chapter we have considered literature of relevance to this thesis. We have considered the complexities of the distributed work-world, the visibility of work and work practices, and the topic of boundaries and boundary objects as it applies to a highly contingent work setting. We believe that the practices in evidence at our field site can benefit from being looked at through the dual lens of boundary objects and work arounds.
As we noted findings by Grinter that the development model and practices used by software professionals can be embodied in the tools they use. Lethbridge and Singer highlighted that defect tracking systems are readily maintained by software professionals because they perceive them to contain information of value which it is readily accessible to them.

We have highlighted that in a workplace with multiple work arounds being operated at the same time, means must be found for these work arounds to accommodate themselves to future changes in each other. We plan to expose how the practice of controlled alterations of the builds and defect reports supports the work of these boundary objects so that they continue to be valid. The boundary objects shift to accommodate the changes that occur in the boundaries.
Chapter 3 Stories from the code face

3.1. Introduction

In this chapter we will present the type of work environment we sought in order to conduct our field study. We outline how we might gather data from a site of the type we were interested in studying. We detail how we came to select the particular field site we conducted our study in, and how we went about gathering the data there in the light of the practical considerations we had to be mindful of.

3.2. Studying day to day practices

What has driven our research focus is our core interest in looking at the everyday and routine which can be termed the “work-a-day world” (Moran and Anderson 1990). A world composed of those activities that workers see as mere unremarkable matters of dull, mundane routine; such activities comprise for us their work practices. As Lethbridge, Singer et al note in their 2003 study “Software engineering is a human task, and as such we must study what software engineers do and think. Understanding the normative practice of software engineering is the first step toward developing realistic solutions to better facilitate the engineering process.”

Of the existing literature covering software development, (Information Systems etc) a considerable body looks at work practice from the top down view, starting with a set, preformed hypothesis and seeking to test this theory against what they find in the field. This thesis is not an examination from the strategic overview of software development as favoured by others (Carmel and Agarwal 2001; Carmel and Eisenberg 2005) which often involves taking a management perspective on what the actual workers on the ground are or should be doing. In doing so such researchers arrive at the field site looking for a match for an academic perspective that they possess and this mindset can all too easily reshape what is found at the site. In the two decades since the extensive qualitative work of (Curtis, Krasner et al. 1988), literature looking at software development work has been aware of the potential of qualitative research to inform. Yet it has also perhaps been restrained by a consciousness of the
commitment in resources required for it to be done correctly and with sufficiently thorough
rigour. In seeking repeatable and reproducible empirical data there is the inclination towards
sources of hard data. This data is often derived from using quantitative metrics and hypothesis
testing, as well as through laboratory experiments, conducted for the most part by using
students.

Having chosen to study work practice, we need to consider what characteristics are
prerequisites in a field site for it to suit our interest and make a study there worthwhile. The
criteria that the site should satisfy a) extended periods of time can be spent with people as
they go about their daily work, in their regular work environment and b) allow us to have
access so that we’re living and breathing amongst them, and are sharing in their joys and
sorrows. The means of gathering data is informed by and reflective of work which afforded an
opportunity to workers to tell their tales and war stories while they worked (Orr 1986; Moran
and Anderson 1990).

In order to benefit most directly we must embed ourselves and be their associates to hear
the tales told, and to avoid missing the unplanned opportunity that might arise, the unbidden
problem that otherwise cries out silently for our attention. By doing this, we expected to be
able to seize upon a practice particular to this setting which provided the means to see the
workers moving in concerted action, but not specifically what that practice might entail. The
detail of the specific practice we would examine was for us at the outset a ‘known unknown’.

3.3. The stories we were hunting for

We have chosen to look at all the professionals involved software development, (testers,
developers, managers and those in other support functions), rather than using specifically or
exclusively one role, be that of manager or developer as the entry point. Also, we seek to look
at their work, a) common place and recurring, not individual isolated incidents b) when they
are together not apart, c) working collaboratively as members of the software team (testers,
managers and developers) along with d) those who support the team but who are not
necessarily part of it.

The day to day work of software development may be described as a collection of
activities by a group of people attempting to produce a software artefact or service, or to
modify a complex system or process. The development of software by means of a global
software endeavour presents problems for the conduct of work which are intrinsic to software
development, whether sites are separated by a) borders, b) cultures and c) increasingly by
Stories from the code face

time zones even if they are still within national boundaries (Abbott and Jones 2004; Abbott and Carmel 2005). Parallel problems are encountered when conducting research into such work situations. In addition to the traditional challenges to research in software work, such as a) making initial contact with and arranging to meet people in different locales, b) ensuring sufficient and equivalent levels of access to work sites and data sources, c) negotiating appropriate and timely access to relevant people, and d) obtaining the use of artefacts. Both of the latter two may be restricted by reasons of understandable sensitivities and concerns of commercial intellectual property rights. Further to this, with so many competing issues needing attention, it can be difficult to pick out which issues are specific to Global Software Development (GSD) and more generally applicable and not simply part of work. A better understanding gained through this form of research, can serve as the basis for improvement in the execution of the methodological approach.

3.3.1 Phases and activities of interest

The activities in day to day software development, global or local, are frequently organised by means of development models which place more focus on the completion of some activities, others may omit certain activities completely. Irrespective of the development process model being followed, it is a highly complex undertaking and at least some of the activities below will form part the effort:

- the gathering of requirements
- the analysis of those requirements to produce specifications
- the use of those specifications to produce design documents and test plans
- the design of the architecture of the software
- the creation of prototypes and joint creation with users of use cases
- the creation of test scripts and procedures to ensure that the resultant outcome matches as closely as possible the original specifications
- the coding of the software
- the running through of the test plans, scripts and procedures on incremental versions of the software until reaching a stage when validation can be performed
- sign-off and delivery or deployment of the finalised software

To look at each of these phases in the depth it demands is too great a challenge, even for a team of people, and can lead to the temptation of taking too high-level a view of the work in order to talk about it in its entirety.
Instead, we propose to look at just a single phase, a phase during which all team members need to come together in order for their work to be accomplished. From the detail, in this one phase we expect to bring some of the broader issues that can also arise in other phases to light for more general discussion. The work of testing written software code, i.e. the effective capture and resolution of defects forms an intrinsic part of any software development project. This is the primary focus of our research observations; the phase when the test team are working with software builds. The developers work primarily with the code which is in turn used to generate these builds or versions of the software product; whereas the testers are primarily working with the built software product to uncover problems in the code use to create the builds.

Within each stage of the software project, there are a number of distinct, discrete yet interdependent roles discernible within a software team; roles focused on development, testing, Deployment, performance, and management. The software architect, development leads, test leads, and project manager in the team play roles similar to those discussed by (Brooks 1995) for software development. The software architect is the keeper of the vision of what the final product should be. The project manager works to ensure that the necessary resources are available at the correct time, negotiating with other groups within and outside of the organisation to realise the end result.

The work of a member of the software development team at the testing stage encompasses many tasks. Members of the Testing team are writing and running through the test plans, scripts and scenarios to reconcile the software produced with the design specifications. ‘Use cases’ and other descriptions customer product use scenarios are made available from the software designer, who may well be the same developers who will code the software. The tester will create many test scripts to run through these scenarios. After completing the execution of the test scripts, accounts of their successful outcomes are documented (in the form of test logs) and the unsuccessful outcomes (in the form of defect reports).

Developers find that addressing these defects form a key part of their tasks even though the work on individual defects arises unpredictably As individual defects are not planned for or are unintended (or so we hope), the consequent work is not part of the original specification process and, hence, cannot be specifically scheduled. However, effective planning means that time is normally allocated within the overall plan for the resolution of general defects. Such allocations are typically made by estimation based on previous experience and may simply be extrapolations based on the data contained with the existing
defect tracking applications for other similar projects. This uses the experience and knowledge of specific team members and also the data captured over the course of previous projects. Hence, defects when taken together are significant to the project planning process. The goal of everyone using a defect tracking application is to try and close off the defect reports; thereby eliminating them from their work queue.

These are people who are subject to real world delivery pressures while coping with real world problems that unexpectedly impinge on their work, people getting married and members are moving in and out of the team. We are looking at work practices at a crucial phase in software development where defects are uncovered and resolved and where the defect tracking system forms one of the core foci of activity. Through our examination, we bring to rude life the dynamic environment in which the software team operates.

In summary, we view work practice as what people do, as a matter of routine and common recourse. We are interested in what a range of people within a software project team do together, and not the stated policy or documented formalisation. This is not a laboratory environment or classroom study group.

3.4. **How we might find such tales**

In going about our field study, and as outlined in chapter two we draw from the workplace studies tradition which forms a core thread through much Computer Supported Collaborative Work (CSCW) research. Through this tradition workplaces studies in CSCW has exhibited a clear preference for broad ranging, in-depth, immersive forms of field study, drawn from the tradition of ethnographical studies, resulting in accounts of a qualitative and narrative nature. These studies necessarily involve direct local observation, of a process of being embedded and becoming part of the work team as it collaborates. As indicated by Dittrich, John et al. (2007) qualitative research has its main strength in exploring and illuminating the locally situated practice of software engineering. The research approach we adopted stresses the value of examining a situated focus, of seeing the global in the context of local work, and the potentially broader picture in the specific evidence of the work that is performed.

However, on some projects, a portion of the activity is distributed outside the immediate site and is not so easily directly observed. Extensive travel and use of multiple observers at each site are means to achieve this but, on this occasion, not available to us. One ready means to bridge the access gap to non-local activity is by extensive use of more quantitative methods. We do not, in principle, argue against the use of quantitative methods in software
development research. Indeed we would wish to highlight the complementary nature of quantitative work to the qualitative. It serves to complete the picture and provide re-emphasis through an illustration of the distinct benefit of a qualitative focus in uncovering particular practices such as “patching”. There are clear differences between quantitative and qualitative which does not rest solely in the nature of the data collected but in the overall approach to data collection. We do not start with a hypothesis to prove or test; a qualitative approach is thus beneficial to us as it assists us in refining our research question.

3.4.1 A qualitative approach

Facets of qualitative research methods have been used in isolation as part of an experimental artifice (Bos, Shami et al. 2004) to consider the temporary introduction of an application, or through combining observation with later examination of files ‘logging activity’, which are generated in the course of the observed tasks being performed (Brignull, Izadi et al. 2004). Other studies use students or teams drawn from other domains, who try to behave like software professionals (Clear 1999; Clear and Daniels 2001) or work colleagues who do belong to the software development domain (Churchill and Bly 1999; Churchill, Trevor et al. 2000). We contend that this work, useful though it is in complementing other work, may, especially in looking to gain an understanding of issues that actual professionals encounter, be overly artificial and necessarily limiting in its applicability.

In order that qualitative empirical studies might be best able to inform future design work, it is required of researchers that he/she be open to the messy, human work situation as they find it, and to the data that offers itself up for consideration. During the course of our initial exploratory study rather than predetermine the questions to be answered prior to experiencing the environment, we let the environment reveal to us what was significant and worthy of further investigation. In this way, the richly descriptive data is gathered opportunistically, even fortuitously, with the focus being reflectively redirected and reinforced by means of repeated observation, and contemporaneous interviewing. By this approach we can see the sense making that occurs as people progress through their work day; i.e. the form that their communication takes, the local terminology, the shorthand phrases used to refer to concepts that are readily understood by those natives. What can seem ‘strange’ to the outsider is part and parcel of their work day.
3.4.2 Narrative inspiration

Our intention is of following in the traditions of workplace studies, and looking for a method that allows us to be living and breathing amongst the workers, sharing in their joys and sorrows, affording the opportunity to workers to tell their tales and war stories while they work. We draw from Julian Orr an approach of allowing practitioners to inform the researcher in their own terms about what it is that they were doing, and to do so as part of their normal work pattern enabling greater fidelity to the work (Orr 1986). Orr’s work focus was on photocopier repair workers and their ambulatory work as they moved from office to office, fixing broken machines. Accompanied by Orr the practitioners talked at their own pace, in their own time and in their own terms about what they were doing, as they did it. It was designed as a method to facilitate the gathering of data as part of their normal work pattern, relying on oral story telling as events unfold, making for more naturalistic data that was supportive and not disruptive of the work the people were doing.

We draw heavily on Orr’s narrative inspiration to assist us in gathering data and in the presentation of it. We lay bare the everyday routine by means of the exemplar of a daily meeting that shows pertinent and interesting aspects of the various coping practices that had evolved to deal with delay as a consequence of distribution. Thus we detail in a rich documentary style what is done day in day out telling stories as a means to bring the situation and the work being done to vivid life.

3.4.3 Ethnography

Ethnography has become popular in the social sciences and has been used in a variety of research domains particularly where people are involved in group activity. Ethnography thus rather obviously suggests itself as a methodology for us to obtain a deep, well-founded understanding of our field site. An ethnographic study could to be undertaken by full immersion in the environment, perhaps working as a tester or developer for the group.

Ethnographic immersion can be undertaken by long term means as in Harper’s 1998 study at the International Monetary Fund (Harper 1998) or more comparatively short term as in (Harper and Carter 1994) study. In both extremes of observation the researchers and those being observed have discovered that a commonplace practice exists serving a particular specific purpose in the performance of their work process, separating the professional and social work world of distinct phases, architect and engineer, design and implementation.
Observing actions as they happen, while complementing those observations with interviews with the actors reflecting on what has just occurred.

3.4.4 Ethnographic alternatives

However, we recognized early on that it would not be possible to immerse oneself fully for a sufficient duration or, extend the considerable effort in time and other resources required to do a sufficient rigorous study to be faithful to the ethnographic tradition. Further taking our inspiration from this CSCW tradition, our field work has been conducted in a manner termed “ethnographically informed” (Rodden 1991). Ethnographically informed research is where one draws on the strengths of participatory observation and the need for time spent in the field, while acknowledging that the researcher does not go completely native. This means our field work does not consist of a fully immersive ethnography study in the complete sense of the term. We did seek to leverage as fully as possible the access we had to the site, the people, their technology, and their dining habits. We read email on their system, viewed the defect reports, attended meetings, received the blocker’s meeting minutes all as they did so, being immersed in their day to day affairs so far as the circumstances allowed. Whilst we do not believe this sufficient for a full ethnographic study, with the attendant complete immersion into the environment, we feel it combines the approach with our constraints.

Other modified approaches have sought to apply another flavour of the ethnographic approach, that which is termed ‘Rapid Ethnography’ (Millen 2000). This approach can be applied to environments which are less hospitable to immersion, as there is no physical environment in which to be immersed. This approach, whereby electronic data as entered by people is viewed as substituting for them, is an extreme we do not go to, but we were aware, in crafting our approach of the temptation to be overly short and nasty. We sought to steer a middle course, between full blown ethnography and simply dipping in and out of the work environment.

Prior research on the impact of technology on practice (and vice versa) has used the approach of immersive field studies in studying the introduction of a defect tracking system and its opportunistic extension. (Orlikowski 1996) saw it used to assist in training and later the introduction to a setting of a similar ‘Lotus Notes’ based issue tracking system which jarred with the existing practice (Orlikowski 1992). Both studies showed that problems and opportunities inherent in the practice can emerge and are observable as the technology is in use. Examining in detail the progress of work and the restrictions it must be put through to
Stories from the code face

reach an outcome can yield up interesting insights into the documented process and how the practice of it actually occurs (Gerson and Star 1986).

3.4.5 Groups meetings

Working as we are with groups, it is necessary to see them when they are together to be truly able to take the pulse of their work practice. Meetings are an obvious choice for this and, as used successfully by others (Mark, Grudin et al. 1999; Mark 2001) they form a lens through which to observe practice when it is at its most raw and exposed, whether for one-off purposes, such as intense design collaborations, or mundane weekly project management. By attending meetings in person, listening in on conference calls, and taking detailed contemporaneous notes, our use of the detritus of meeting minutes and outcomes as threads to be followed opens up routes through their work for examination.

3.4.6 Compatible data gathering methods

Other mechanisms for gathering concrete data in field site studies are possible.

- shadowing of individual developers for extended periods or shadowing a group of developers for a shorter period,
- questionnaires distributed in person or made accessible by the web
- surveys distributed in person or made accessible by the web
- internal data gathering done by the organisation itself
- tool usage statistics gathered by the organisation itself to form a picture of the scale and scope of the project work

Questionnaires while they can often reach more people than a researcher could speak with in person are also self limiting in that they fail to anticipate all possibilities and can, from the outset, unnecessarily guide the answers in a particular direction. Herbsleb, Mockus et al (2000) used the data already resident in the defect tracking system. Specifically they combined the data on when defects reports had been entered and when they were responded to, with the location of the entrants and responders. This was as a means to elucidate responses from participants on the distance between participants and how this correlates to delays in rendering assistance to co-workers.
3.5. **The Field Work – the story we found**

The field study was conducted at the subsidiary software development operation of a large multi-national organisation. This site had been selected after an engagement process involving many visits to a number of companies that offered the opportunity to conduct field sites. We were in retrospect fortunate with the potential field sites to which we were able to gain access to even if, in the end, we did not use them as extensively as we envisaged.

3.5.1 Field Study Duration

Contact was first made with the company site in January 2006 expressing our interest in studying concrete practices in software product development. Over the course of the summer of 2006 we familiarised ourselves with the organisation by attending various project meetings and interviewing a number of members of a software team. Our study of the software development team began in March 2007 and was completed in the autumn of that year. Our study of this specific software development team began in March 2007 and completed in the autumn of that year, with our on-site observations encompassing the release of the BETA\(^2\) build. The main phase of the project release concluded at the end of September 2007 with the shipping of the product. We monitored and had ongoing access to the meeting minutes until this natural end point.

More than 23 whole days were spent in the field spread over a three month period, a period stretching from May 16\(^{th}\) to July 13\(^{th}\). The time was spent observing the activity of the team in its own work environment, participating in meetings and group activities and occupying a desk in the open plan office space next to the team’s area. The periods of time spent in the field site varied between four consecutive days when the team was approaching an important milestone or release, to one day a week in order to maintain contact and awareness. We continued to follow the meetings and development work remotely for several months until the final release date.

---

\(^2\) A beta build release is a common occurrence for many software projects whereby a stable build of the product is released to a beta audience who have registered to use it on a trial basis and test it with their own work environment.
3.5.2 The software product of the team being studied

The project team was tasked with producing the Portal Learning Extension (PLE) software product, a portal-based learning solution. The underlying portal technology is a product of the same multinational organisation and it runs as an infrastructural platform between the operating system and the PLE. The portal is enterprise level software which consolidates services or applications into a single point of access, usually through an internet browser. This serves to unify the authentication of users, availability of portal resources, profiling, and management of hundreds of potential applications.

The software product - Portal Learning Extension (PLE) 2.7 - is a content management system for use in supporting the delivery of learning solutions within a corporate environment. It is a portal-based learning solution that allows any group to manage their training programs and associated tasks more efficiently. Students can access a customised "just-in-time" online education environment which can be fully personalised. It is a standards-based learning system which improves the availability of learning resources, helping students to improve their skills in a flexible manner. The portal technology is also a product from the same organisation and it runs as an infrastructural platform between the operating system. The PLE

- uses customer-supplied job profile information to deliver roles-based learning resources right to the individual's desktop
- automatically assigns learning activities as part of a personalized employee development plan so users can work towards closing their skills gaps
- enables learning resources to be accessible at all times within the context of workers' day-to-day activities
- includes an authoring tool that can be used by subject-matter experts to design customized courseware on new products, business processes, regulations, etc., without learning complicated programming skills
- supports industry standards and allows customers to reuse their existing third-party courseware libraries
- integrates learning resources on the desktop as part of a comprehensive portal solution
- delivers learning to mobile and multilingual audiences, wherever they are located
- enables organizations to more cost-effectively manage their classroom-based and e-learning programs and content, and adapt them to changing organizational requirements
3.5.3 General detail of the field study

Our principal site was based in Ireland, near Dublin. The local organisation was part of a free standing software company that had been acquired by a larger organisation over a decade previously and was then a wholly owned subsidiary of its parent company - a large multinational player in the technology sector. At the time of the acquisition the company had a significant product portfolio which matched the needs of the parent which had lead to the buyout of the company. This acquisition left much of the local team personnel in situ and a large portion of the senior team members of the teams we looked at had join the company prior to the acquisition. It was a settled team with long experience of working together successfully. The team working at the site interact with a number of development locations on the eastern and western seaboard of the US, on continental Europe and the Middle East.

![Diagram of distributed development team]

**Figure 2: A Representation of the Distributed Development Team**

The organization of the team is depicted in the graphic in Figure 2, the bulk of those working in the project team were located in Ireland but a number of developers (represented by the blue dots) who contribute software components to the team were located in Jordan, Israel and the US. Communication between them was by email, Instant Messaging, phone and the document databases they shared.
3.5.4 The software project team being studied

The PLE team (Figure 2) we observed consisted of a locally-based project manager as well as more than a dozen developers, two dozen testers, and Deployment and Performance testers. Whilst the majority of the team was located in Ireland, two developers were based in the USA, and during the course of the study one local developer moved to Mexico from which he continued to work on the project. Two further developers who were based in Jordan contributed discrete components to be used in this and other products. In addition, office policies at the Irish site allowed employees to work from home 1 or 2 days per week. We followed the project team through the development process for several months up to Beta release point.

While the majority of the team members of the PLE component are collocated, the portal software that it is built upon was the result of collaboration between various sites located across the globe from the West Coast in San Diego to East Coast Boston and the Middle East in Jordan and Israel.

The PLE “team” we observed was actually a heterogeneous cooperative ensemble, consisting of developers led by two senior developers who headed two groups. One group was focusing on the User interface (UI) and Dataloader, the other group on Web Services. The project has its own dedicated Testing group dedicated to functional testing of this product, and was assisted by the Performance Testing group which serviced multiple software products simultaneously. There exists also a Deployment team, one specific member of which was assigned to work with the PLE team, which is tasked with downloading and installing the software builds on servers set up as they would be at customer sites. This individual deployment engineer assigned to the PLE was also responsible for supporting other local teams. There was a local project manager for the PLE team who ensured the coordination of all these local groups and their collaboration with the considerably larger and much more distributed portal product team.

In software development it is the norm that there will be a constant tension caused by the amount of time required to strictly adhere to the prescribed process and the pressure of meeting deadlines laid down well in advance as part of scheduling the entire project. The team we worked with had evolved various local practices in an effort to maintain a smooth flow of work throughout the intense defect resolution phase of software development.
Stories from the code face

As we have already mentioned, there were several distinct groups involved in the work of software defect resolution for this project. It is sensible that we describe these groups more fully

(i) Development – The Development team had three leads for UI, Dataloader and Web Services. We will explain later these areas of functionality in more detail. The bulk of the development effort was in Dublin though a number of the developers were located overseas on the East and West coasts of American while other functional components that the PLE team used but not exclusively were developed elsewhere in Israel and Jordan.

(ii) Testing or Quality Engineering (QE) – In this organisation the core of testing and quality engineering work of the software team was commonly referred to by the acronym - FVT - which stands for Functional Verification Testing. Though FVT is really merely one of the phases of testing that they are responsible for, it was often used in reference to all the testing team. The other forms of testing taking place were Build Verification Testing (BVT), System Verification Testing (SVT), Globalisation Verification Testing (GVT) and accessibility testing and Performance Testing (PT). As deployment and installation took more than a day it was not possible for every build created to be subject to the full gamut of tests. Build Verification Testing (BVT) took place for all builds that were going to be subjected to Functional Verification Testing (FVT). As in FVT itself the testing involved the team running a predefined set of test scripts or scenarios against the software build. The testers other than for Performance and SVT were all part of this particular project team.

(iii) Deployment – This is a separate group from the PLE team. There was one deployment member who had been assigned responsibility for this project. It was not necessarily the only software project they were working to support and from time to time someone else from the Deployment had to step in if the assigned individual was unavailable due to holidays or other circumstances

(iv) Performance – This is a group of software testers who had specific focus on the performance of software cross product lines on different hardware and operating system configurations. They had a very different background and focus to what aspects of the software product they were interested in. Their role was to stress test the product under load conditions similar to what it would encounter in the real work world. As this was a product for assisting learning, this meant simulating many thousands of user profiles,
many hundreds of simultaneous users, and many thousands of courses were installed on hardware typical of the anticipated work environment.

Management - There was one project manager, and one manager of the FVT team. The lead software developers take a role in organising the work of their own component team members but they are not typically considered as managers of their teams. Finally there is an overall development manager responsible for the progress of the entire project, which represents the project to more senior management.

The project team members ranged in experience from new employees with less than 12 months experience with the organisation to some with over a decade and a half. They often work from home and use remote access software to use their work machines from home. Meetings can be attended via teleconferencing something that is done even when at their desks at the workplace. The defect tracking system in use had been developed in-house and was based on a Lotus Notes database application. The individual defect tracking reports are referred to by an abbreviation but we will substitute the term Defect Tracking Reports in place of their term or more commonly DTRs in place of their 3 letter acronym. This will how the defects reports will be commonly referred to in the observations found in the remainder of this thesis.

3.6. How the field study was conducted

One of the challenges in such a commercial workplace environment is that much of the information is necessarily confidential so much of the content cannot be directly or too explicitly reproduced in this work. It is one of the compromises that must be made in order to gain access to this world; however we can use some extracts of it in isolation. In the main we used qualitative methods in our field research. The use of qualitative data in software engineering, as we noted already, is becoming more prominent (Dittrich, John et al. 2007).

Whilst we were open in our data gathering to what might possibly present itself, we were also conscious that we could not examine everything, the scale of such an undertaking would be simply too vast. With the very time pressured nature of the work it is not possible to pull people aside to chat about what they have just done. It is not possible to be everywhere and with everyone at once in a work environment, especially in software where so much of the work involved thinking about a problem before actually acting. It is all too easy to only see someone tapping away on a keyboard looking at a screen all day. What we needed was an opening into a tight problem, giving access to as wide a range of people as possible. With the
Stories from the code face

aim of finding a story that would thread us through the tapestry of their workday, an opening
where we would know to be present when the action would happen, even if we were unclear
what form that action might take. The daily blockers’ meeting was just one such opening.

Our data collection and analysis methods included the following

- observation
- shadowing
- participant observation
- document analysis
- in-context interviews
- sketching
- audio recording³
- focus groups
- workshops

The research team was granted access to the project document repository and its members
were added to the Development team’s mailing list. We were also able to install and use the
instant messaging system used by the developers. This enabled us to reach a better
understanding of the ongoing activities and to continue our observation even when we were
not present at the site.

We also conducted both formal interviews and informal discussions with various team
members and were allowed to observe various teleconferences. We maintained a diary and
took detailed notes for every day spent in the field. Minutes of meetings as circulated via
email were collected and reviewed for those meetings that we did not attended in person.
Face-to-face interviews were conducted with five members drawn from the Testing,
Development, and management teams: two developers, one member of the test team and the
project manager. The interviews were semi-structured and focused primarily on a practice of
‘patching’ that drew our interest during the initial phase of the observation. Prolonged
conversations were also had with various members of the development, Test, performance,
and management teams via instant messaging, email, in corridors and post-meetings.

In addition to the onsite presence the researchers were granted access to the defect
tracking system, test scripts, company intranet, and to the team’s mailing list. We were also

³ Video-recording and photographs were prohibited in the field site
Stories from the code face

allowed to make use of the instant messaging system, useful both as an awareness mechanism for when people were working be it at their desk or from home and as a communication channel with the members of the observed team.

It was common for team members to work from home, thus we were also granted the opportunity to participate in teleconferences which allowed us to observe directly the team members’ interactions with people in various other locations (Ireland-offsite, US, Jordan). The remote participants were made aware of our presence in the room on these occasions. This access allowed us to immerse ourselves and reach a fuller appreciation of the ongoing activities and to carry on our observation even when not present on site.

Meetings serve as a revealing environment to gain insight into other work that is going whether with a single defined purpose (Mark 2001) or as a means to understand work that it not so accessible to observation due to its distributed nature (Mark, Grudin et al. 1999). While the venue for the meeting and the participants even the timing might change the constant through all this was the blocker’s document and the defect reports to which it linked. The links in the blockers’ document open into the linked DTR, providing the foundation and common reference point, from day to day, week to week, build to build, for the collective efforts to resolve the defects. Structured meetings with well defined goals can be highly effective in allowing participants to arrive at solutions to complex problems while not tying up unnecessary resources.

With access to the data contained within the defect tracking system, the temptation exists to reproduce the approach of Herbsleb, Mockus et al (2000) and focus solely on the rich mine of information present in the defect tracking system. However we go beyond the raw data of mere individual defects to see the work practice as it happens, not individual defect stories, fascinating and compelling though they are, but rather, the story of how they are handled when the consequence is more than simply failure to function as intended.

This consequence is when a defect has a broader impact on the work of others; such as when it may mask the existence of other defects because testing cannot be performed in certain areas. The concern of the Testing team is that areas which have not yet been tested are unknown. The areas where testing has been done and defects found is a known quantity: untested areas are more of an unknown quantity.

The absence of defects is significant in that it does not mean that the work is progressing perfectly; in fact their absence may equally be a cause for concern. The Development team
and in particular the software architect can take this as an indication that the appropriate areas are not being sufficiently stressed or tested. Major blockages which prevent Testing from progressing to other component areas of the product thus have effects outside of themselves. In some sense, the presence of a certain amount of defects gives reassurance to those outside of the Testing team that adequate testing work is underway. The contrary view is that until such time as the defect tracking report database has started to be populated with defect reports, the core work of the project has not yet begun. Indeed, the progress of a project is often charted in terms of the change, over time, in the numbers of critical defects.

3.1. Chapter summary

In this chapter we discussed how we approached the gathering of data and the myriad of problems posed in doing so. We outlined the methods we chose and why we took this approach.

A combination of methods to gather data is not unusual. We looked at what others have done in similar settings comprised of similar people working towards a deadline on a product to be used in a real world commercial setting. Lethbridge, Singer et al (2003) used a number of methods: a web based questionnaire, the shadowing a single developer over a period of fourteen weeks, and the further shadowing of nine developers for periods of an hour. They made use of tool usage statistics that were generated in the course of the work being performed.

A combination of methods allows for different angles and aspects of the work to be uncovered though those methods in turn have their limitations. Questionnaires are limiting, in that they can fail to anticipate all possibilities and can unnecessarily guide the answers in a particular direction. Herbsleb et al have used the data resident in the defect tracking application they examined regarding when defects had been entered and when they were acted upon (Herbsleb, Mockus et al. 2000). This data was used as a means to elucidate responses from participants concerning their research focus of the effect of distance between participants and how this correlates to delays in rendering assistance to co-workers. We examine the project defect history to aid in detailing defects. We similarly looked to use the hard data in the defect report in a more open-ended fashion to draw out the story of the defect with semi-structured interviews. We chose to do this as it marries together both qualitative and quantitative data. Doing so enables us to capture the dynamics in a global project team and how the personal, group and corporate models of software development reach a working balance that allows work to be satisfactorily completed.
We outlined the environment we were dealing with and how this modified and also reinforced our choices with respect to data gathering. In the next chapter we present the actual data we gathered by means of an exemplar meeting, and then present elements of this data in isolation for more detailed examination.
Chapter 4 The story of the field study

4.1. Introduction

In this chapter we present data that shows the local practices that exist in our field study at a software company. In order to do this in sufficiently rich detail and to bring the workplace fully to life, we will draw heavily on Orr’s narrative inspiration in laying bare the everyday routine through telling stories as a means to bring the situation and the work being done to vibrant life. At the core is the exemplar of a daily meeting to show pertinent and interesting aspects of the coping practices involved in dealing with the delay that results in iteration due to a particular form of distribution.

We then present individual items and of note from the practices: those being

- a daily meeting
- the blockers’ document
- the defect reports

Through these we show how some of these defects find their eventual resolution by discussing them beyond the snapshot of a particular meeting that we use to illustrate their practice. We finish by discussing the practice of patching that is so much a part of the workplace.

4.2. The field site outline detail

During the course of our observations we followed the activities of the project team members in their efforts to resolve blocking defects. ‘Blocking Defects’ are defects which prevented them from making progress with development and testing work in certain functional areas of the software. The data we present was gathered by means of attending meetings, by interviewing members of the team, observation, conversation, and reviewing documents including Defect Reports and Blockers’ documents. We will provide more detail later in the chapter on each of these key artefacts of software builds, defect tracking reports and blockers’ document. We present a daily meeting for the moment now we will start by giving some background information necessary to understand what was happening in the meeting.
The story of the field study

Two artefacts commonly used in software development; software builds and defect reports, are intrinsic to the execution of the practice. A brief description of each is included below for clarity and to assist in understanding the following vignette. After this, we will go into each item in more detail. The characteristics of the artefacts provide a concretely visible manifestation of the coordination mechanisms used by the team in articulating their local work practice.

4.2.1 The daily blockers’ meeting

We sat in on the daily meetings which were devoted solely to dealing with the blocking Defect Tracking Reports (DTRs). These meetings had been in progress for a number of months prior to our arrival and were held each morning at 10am, the exception being on Thursdays, when they took place at 11am. The meetings took place at a point in the daily work cycle in which the latest overnight builds were in the process of being downloaded from the host server. This server was located on the east coast of America to which they had been automatically posted by the US build machine and before they were to be deployed on local servers. Holding the meeting earlier than 10am would have caused difficulties in ensuring appropriate attendance due to commuting problems. Many of the meeting participants would also work from home from time to time and call into the meeting via the teleconference facility. The decision whether to proceed with a new deployment that day, or to hold off until some particular code or feature was part of a build, was just one of the outcomes of this meeting.

4.2.2 Centralised Daily Builds

A new build is automatically generated nightly, and while not every build is deployed for use by the test team, the Deployment team preemptively acts to pull across every nightly build produced in case someone does wish to use it. Testing, or FVT as they were often called, move to a new build infrequently, typically, every seven to ten working days, due to the extensive work involved in changing from one build to another. FVT could typically work on a particular build for anything from one to two weeks during the early stages of the project but this would be increased to having two builds in use per week as they moved closer to the end release phase. There would be multiple builds deployed on different servers at any one time, for example build 05-21 might be on ‘snoozy’, while build 05-17 might be on ‘flipper’.
The story of the field study

There existed a degree of fluidity in the decision of which exact build would be used next by Testing. Dates for taking new builds will have been penciled into the schedule as part of the project planning at the outset but is under constant revision as the project progresses. The precise date to take a new build was usually chosen by the project manager, in consultation with the test and development leads, and was based on a combination of the new feature work that was definitely included and any defect fixes that had been recently committed to the source control and would be in the build. A final “go/no go” decision to take a particular build was taken at the daily blockers’ meeting, depending on the condition of the new build, and, also, how much testing had been completed on the builds currently deployed. If the testing work had been exhausted on the deployed build, it was considered that there wasn’t much point re-testing that which had already been tested. Therefore a tension existed between finishing a full test cycle on the current build with moving to new build to start new testing work on.

Meanwhile, the developers needed to be more up-to-date and most of them commonly made use of the daily builds in order to try out their attempted code fixes on the build on their local machines. This they would do in so far as this was possible prior to checking them into the source control. They will take ‘archives’ (these being the compressed version of the code files used to create that day’s build) to ensure they were working on the most recent version of the software, before checking the code changes into source control.

4.2.3 Blocking defects or ‘Blockers’

In the field study, the term ‘blocking defects’ or “blockers” as they were more colloquially termed by the PLE team referred to any defects that stand in the way of the progress of the testing effort and presented the team with a problem. This was a particular problem due to the regular means of eliminating them being removed as a consequence of the centralized build process and the long turn-around time between making code changes and those changes being available for testing in a locally installed build. These blockers are those defects that prevented the testing work from continuing as scheduled.

The impact of this delay was sufficient to result in the emergence of local practices to work around the prescribed process. Blocking defects are those defects in functionality that prevent the Testing team from fully examining areas of the software build. This type of problem prevents the testers from running their tests to completion in accordance with the test plan. Thereby leaving some areas of the product untested which has a knock-on effect on development efforts as developers are unsure if their work in an area is completed.

In the course of the study we noted the impact which the centralization of the software build process for the entire globally distributed Portal project had on local defect resolution.
The story of the field study

practices at the Irish site. The term software builds, commonly abbreviated to builds, was used in conversation and documentation to refer either to the process of converting the source code files into standalone software artefact(s) that can be run on a computer, ‘Running a build’ or the result of so doing, ‘downloading the build’. Source code is not normally accessible to those outside the Development group this even includes those who are testing the product. Thus, software product builds are generated from the content of source code repositories. The end result of a software build is a collection of files that generate a product in a distributable package (Maraia 2005). It is not therefore the software code per se that is tested by the Testing team; rather, it is the behaviour and performance of software builds generated from this code that is being assessed in the testing work.

4.2.4 The software project team

The team is consists of more than a dozen developers and two dozen testers working locally as part of a project team which is a component for a larger portal architecture. Two developers were based in San Jose, two others who were based in Jordan, Haifa in Israel contributed discrete components used in this and other products and one developer moved to Mexico during the course of our observations.

There is distribution in terms of some of the services the core team use from other teams, for example there was a team in Haifa that they had to spend considerable time trying to contact about some of the DTRs they have come across which related to the work of the Haifa team. They are not entirely self contained or independent because they have dependencies on functionality in other products within this overall portal product offering.

The software development team had, over a considerable period of time, built their own local practices to manage the problem of defects that blocked testing progress. This innovation involved an alteration in existing practices to overcome the impact, felt locally, of the build process being distributed to a central remote location in the US, over five time zones away. This change in local practice as it deviates from normal software development is particular to this set of circumstances but illustrates more broadly some of the challenges local sites face in collaborating in the midst of the effects of transnational distribution.. Whereas (Casey and Richardson 2003) showed having a common toolset across a distributed team as serves as a necessary first step in facilitating collaboration, it is not sufficient in itself.
4.3. **Scheduling and dealing with time**

As is the norm in software development, there is a constant dichotomy between the time required to adhere to the prescribed idealised process and the unrelenting pressure to meet imposed deadlines. When we were observing the team they were working towards delivering and certifying the PLE product on a number of platforms and database schema: Windows, Linux, AIX, HP-UX, Solaris, and DB2, SQL, Oracle. These releases were to take place starting the following September. This multicity of platforms only serves to compound the complexity of what is already a high complex undertaking. The management of this complexity is a very demanding and leads to those involved leaning heavily on the available computing-based supports.

At the outset of the project, various estimates of the effort required are made in order to schedule the work. While Defect work is unplanned in the specifics it is planned for more generally, time is allocated in the schedule not alone to run the various test scripts and scenarios, the actual uncovering and documentation of any defects found, but also to defect resolution, verification and final regression. This is done by drawing on prior experience of projects within the organisation and takes into account those measurable materials that can be usefully applied across projects.

According to the project manager they include the use of “*Metrics gathered from previous releases (usually similar sized releases are used). For example, we would compare bug fixing metrics for a maintenance release to a previous maintenance release but not compare a maintenance release with a feature release.*“

This is aided by the fact that software projects are typically iterations of previous releases. Hence, the developers do not typically start afresh with a blank slate. “*For each release, there are a portion of DTRs/bugs that are deferred from one release into another. Therefore when we begin a new release, we include this with our estimates in addition to our estimated incoming rate of DTRs*“— project manager A

The development group of the project team is also expected to scope out the required time and resource allocation estimates prior to the commencement of the project in particular with respect to what is known to be required for the next iteration. These estimates are factored into the schedule by the Project Manager especially in the case of new features that have to be designed and coded.
The story of the field study

“If there are known issues in specific areas we wish to address, we normally get Dev to analyse these areas and size the effort”. – Project manager A

The Testing team also has to review the previous releases and draw on prior work contained in the DTRs in order to estimate what will be required for the new release. The Project Manager “… noted that we have Quality Plans in place for each release. These contain metrics which include - incoming defect weekly rates, incoming defect week rates by severity and defect rates by Open, Pending, Resolved and Closed. All this data is reviewed at the end of a release and used for estimates for the next release.”

In this way, the overall effort required to be expended by all sides can be estimated at the beginning of the project based in part on the existing information in the DTR system. Similarly, the effort thus far expended at any particular point in the project as it progresses can be tracked by means of the DTR system, and where when necessary subsequently revised and re-estimated. Each week the number of overall defects, those opened, closed or pending is examined, and an estimate of the rate of resolution is reached. This is then compared to the planned rate of resolution and allows the manager, the developers, and testers, (in fact the entire team) to know how far behind or ahead of the plan they are in each area.

4.4. A breakdown of team work areas

Within such a complex software undertaking, there is a need for sub-division and specialisation in order to bring particular focus on each area. Yet those working in each of the sub-divisions need to stay in frequent contact in order to synchronise their highly interdependent work. In the following section we outline briefly the sub-groups, in which the team members of test and Development were working, and what their work involves.

4.4.1 Development coding sub-groups

In our study, developers were assigned features to implement at the outset of the project based on their existing expertise in areas. Taking one example, the ‘Dataloader’ involved a lot of work with different data storage mechanisms and also was a processor time critical area of the product’s functionality. Hence, a person who had extensive expertise in various database schema, code optimisation and performance was chosen to work in this area. Whilst it did not occur in this case, it is also the company’s policy to rotate team members between project teams and also areas of technical competence; if a person has expressed an interest in undertaking work in areas new to them in order, to ensure their work remains interesting. Whereas defect resolution work, in general, interfered with the work required for new feature
The story of the field study
development, those defects classified as blockers were given priority due to their impact on maintaining the flow of work within the overall team.

The people responsible for the ‘User Interface’ were responsible for the implementation of everything the user sees when accessing the software through their internet browser. Problems in this area commonly involved the failure to display the correct information accurately or even at all. It also affected the area of globalisation as it meant ensuring that it was possible to locate the text that was displayed and that the data schema supported the input of character sets from different locales.

The ‘Dataloader’, which handled the retrieval of data from a variety of storage mechanisms, generated the largest number of defects (161) of any of the product areas over the course of the project. This Dataloader functionality allows administrators or course owners to update learning information from another format such as a spreadsheet Comma separated variable (.csv) file or similar. Problems in this area commonly involved the failure of information stored in the csv files (and the templates from which they were created) to load data correctly into the PLE software. Details of a course would typically include the start date, end date, dates for registration, and details of each student such as name, location and time zone etcetera. All courses are created either through the data loader or manually though the UI. The data loader could also be configured to notify users at their email address upon the completion of automated upload.

Web Services (which proved to be the next busiest area with 108 defects) were devoted to the interaction between the client browser software and the server on which the PLE was installed, interaction with other PLE servers and other systems, for example supporting Instant Messaging engagement across the system. Web Services are implemented by code contained in a class file or package of class files and invoked or called by other code that is running in the client's browser. Web Services were quite a complex area to specify at the outset and to plan testing for, relying as it did on the running of automated unit test code. To assist in this the DTR system was used in the initial stages to ensure that these features would be implemented as specified and so DTRs were created by the developer who was responsible for designing the code, setting out in individual defect reports the behaviour and responses each unit test was expecting to see returned. This was in advance of any testing being done, so, in effect, these defects did not yet exist nor might they if the coding implemented the Web Services as it was intended that they be. Resolving these defects meant that the specifications had been implemented as set out in the DTR.
4.4.2 Testing sub-groups

The Testing group also had its own internal divisions, though they were more in the form of the sub phases involved in testing the builds. The Testing team was referred to as FVT (Functional Verification Testing) as the Testing team ran predefined set of test scripts against the software build to test functionality. This was the core of their work, but some tests were defined as being part of BVT (Build Verification Testing), SVT (System Verification Testing), and GVT (Globalisation Verification Testing). As deployment and installation took more than a day to complete, not every build could be subjected to the rigors of FVT. BVT was run on those builds that might be accepted as an FVT build. However, while dates for a build for FVT to work with were tentatively worked out in advance, as part of the overall project scheduling, this was a back and forth process. The dates were selected based on when the Development team believed they would have features implemented, but were subject to change in response to the condition of the builds and the functionality actually implemented. The Development & Test team leaders and project manager would go back and forth over those dates, even once agreed they were not written in stone, they continue to shift to fit the needs of the moment. Right up to the date came around, the managers might still decide to postpone for a day or two.

Members of the team we interviewed pointed up the fact that the sheer number of defects that one has created, pending or resolved per week or per build, is used by those higher up the management rungs as an indication of the progress being made so far. This is seen, too, as a measure of the effort thus far expended towards the goal of releasing a product with zero defects. However, a greater amount of work may have been expended in creating the defect report than in uncovering the defect, and the correlation between overall effort expended by the whole team and the overall number of defects is problematic. Still, it is viewed as being of some use as an indication that at least some work is being performed.

A points system is in use across the organisation and this serves as a means to express the effort required for testing. It is a system used across the company to allocate work amongst the test team members and estimate the overall Testing workload. 10 points equates to 10 minutes of a tester’s time to run through that particular test scenario or test script. This will include any set up time necessary to run the test case. However, it does not include the time required to document any defects uncovered in the process of running through the scenario. There was a default threshold of 80 pts for how much testing would have to be blocked before a defect causing this blockage was potentially flagged as a priority blocker.
The story of the field study

4.5. The Iteration gap

Blocking defects in themselves can cause in a delay in the testing process but this delay was further exacerbated due to the “iteration gap” in the build cycle. Because software builds were centralized and generated in the USA, there was a minimum of a 3 day delay from the time a code was submitted to when it was available for testing. This was from the point at which a code fix was submitted locally by a developer to the central code repository to the central build process generating a new build and being available to deployment to install and configure locally ready for testing. This gap has a negative impact on the Testing team being able to assess in a timely manner attempts made by the developers to resolve the defect. A blocker, once encountered was not so speedily resolved due in large part, to this gap. The full 3 day turnaround is illustrated step by step below:

Day One
- Code needs to be locally checked in before 3pm local time* in order to make the local pre-build
- pre-build is performed locally and completed by 5pm (local time). This is to allow problems that arise to be corrected or backed out before the next step
- Code passes the pre-build process
- Code is checked into the US source control (Clearcase)
- Central Build commences on the US East Coast at midnight (US time) - 5am (local time)

Day Two
- Build is posted on a US based server at 4am (US time) - 9am (local time)
- Build is pulled down to Dublin 5am (US time) - 10am (local time)
- Build deployment is completed by 4pm (US time) - 9pm (local time)

Day Three
- Build is subject to BVT and smoketests which take roughly half a day
- Defect can now start to be regressed after lunch almost 48 hours after the code change was made by the developer

*In this case local time refers to the time at the local site in Ireland.

This delay or gap in iterating might be viewed as manageable within existing practice were the first attempt to correct a defect always successful; it comes rapidly more unsustainable as each successive attempt to correct the ‘blocker’ fails.
The story of the field study

Consider the situation if the attempted fix fails, and it is possible that this failure would not be detected by the test team until after 3pm on Day 3. The developer is not notified as to the failure until after the 3pm deadline. This mean that work on the second attempt cannot be commenced prior to that deadline and hence won’t be checked in until sometime on day four at the earliest. At which point the cycle as above will start all over again. This means that the second attempt at a code fix, (the first attempt of which had first been checked into the source control on, for example, a Monday before 3pm) might not be available for testing in a build until the Tuesday of the following week. This gap of over a week in seeing a 2nd attempt is unacceptable in the pressurised schedule the team works under.

In the case of blocking defects it was readily apparent that this delay in being able to continue the test cycle was simply too long. This iteration gap, coupled with the concern about breaking the build (by checking in code that had been rushed), was a key motivator in the emergence of a local “patching” practice to maintain the smooth flow of work. Patching in itself is a common practice in software development, (though occurring more on the post release stage (Dadazie 2005; Gerace and Cavusoglu 2005) and more recently within the open source community). However, it is not as commonly known within the development community because it breaks the accepted process and the temptation to alter builds after installation undermines testing protocols. This is something acknowledged at the site by the developers in speaking about the decision to patch or not, even if the defect was not a blocker.

Dev A “I, eh, I would have been tempted now if it was fairly safe. I mean beforehand but probably not now there’s too much patching going on. If you’re fairly sure, if you see something, sometimes you just can’t check it on your PC. It’s something that happens on a, I mean I don’t want to start crimping a course into my local. Not the main blockers, the official...So if the DTR was something to do with that, it would be very difficult for me to verify. So yeah, in the early days I probably would have sent a patch to whomever...to Ernie. Not even for fixing, to have a few system outs to debug there...might get a few ideas there. But a lot of times now, we’d have a few issues there and it would be obvious, and we’d fix it and deliver and we may not even verify it...,that’s it’s fixed at all. I know that happens, I’ve done it myself. But that’s not the way to do it. Really, we should be very sure about what we’re delivering. But we’re actually more worried about stuff not building. That’s the worst of all because that affects everybody.”

From this we can see that patching i.e. the replacement of existing files in a build that the Testing team are currently working with, if it is to occur must do so under strict controls and clearly must be done in as plain transparent way as possible. Otherwise the test team will find it increasingly difficult to trust and rely on the build they are working with. This need for transparent action lead to the blockers meeting. Further constraints were in place by the
Project Manager’s controlling of which defect was deemed a blocker and the Deployment team’s control of what files were changed on any installed build that was being tested.

4.6. The Vignette: the daily blockers’ meeting

In this section, we present a short narrative to set the scene for the later discussion of the articulation mechanisms involved in the specific “patching” work practice that forms the focus of this thesis. This practice is not exclusive to our group, but is not widely used within the company because other groups are not faced with the same constraints.

Recap of Background: The PLE team met every day at 10am apart from Thursdays when they meet at 11am for the “blockers meeting” to review the latest information on blocking defects and thus reprioritize and decide on a course of action for the next 24-48 hours. One representative of each group was expected to attend the meeting, although they were not always the same people every day.

On this particular Thursday, May 31st, in the meeting room were the leads of the two Testing groups and one of the senior developers. Four others (including the project manager) joined the meeting via telephone conferencing.

At this point in the project the team hadn’t had a new working build for the last 10 days. This meant there was nothing new to which they could move their testing work over to. As the following Monday was to be a public holiday, they were confronted with the prospect of having the testing effort blocked in a number of areas for at least another 4-5 days. The meeting was chaired by the project manager over the phone. As happens for every daily meeting, he went through all the defects on his list sequentially as they appear on the blockers’ document. As the project manager went through the list, he read aloud the shorthand names of defects – referred to by the last 4 digits of the defect id e.g. PRAV6ZSJTY is referred to as SJTY - and waited for the other participants from the Development and Testing teams to share relevant information on the status of each specific defect.

The first category discussed were the Build Verification & Testing (BVT) defects which determined if a build was running at a minimum level of functionality that Testing deemed sufficient for their further work to continue with that build. Finding a defect as part of the BVT Testing didn’t necessarily mean that this build would not be accepted as an FVT build but defects found as part of BVT work were categorised as BVT defects. One of the defects in the BVT was 8HQ8, which was a bad one, in that it was appearing intermittently meaning it was hard to be sure that was really eliminated. They thought they had solved it previously but it popped again: it had initially appeared 41 days before, disappeared, then reappeared, a
temporary workaround had been found, and now finally they had a patch which the developer
had attached to the defect report. It was noted that "This was not a problem on the original
1004 deployment but since it was redeployed this has appeared."4 From then on it had been
under investigation until the 15/5 build when the developers could reproduce it themselves.
This led to a workaround in the test case behaviour, which was to “go into course details first
and course details will be available.” At this meeting it was said that the developer was now
targeting delivery for ‘Thurs 05/31’.

The Deployment specialist then announced his decision to “re-base” the build (re-install
all the components on the server from scratch) on Snoozy (one of the server machines the
build is installed on) to make sure everything will go well this time. The decision was greeted
with applause by the meeting participants. After that, he was going to re-apply all the
provided patches and someone else (because the Deployment engineer was actually working
from home) would have to copy the patched build archive to a 3.5 Mb USB key. They would
then physically provide it to every developer at their desk: a peripatetic form of file
distribution. They then moved onto discussing the other defects.

The table5 below shows the headings for the blockers’ document entries

<table>
<thead>
<tr>
<th>Build date</th>
<th>DTR#</th>
<th>State</th>
<th>Severity</th>
<th>CS</th>
<th>QA</th>
<th>Dev</th>
<th>Proj</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>this was</td>
<td>the</td>
<td>the full</td>
<td>Severity</td>
<td>The</td>
<td>The</td>
<td>The</td>
<td>A short</td>
<td></td>
</tr>
<tr>
<td>typically</td>
<td>DTR</td>
<td>ID of the</td>
<td>version number</td>
<td>Quality</td>
<td>developer</td>
<td>developer</td>
<td>description of the defect</td>
<td></td>
</tr>
<tr>
<td>not filled in</td>
<td></td>
<td>DTR</td>
<td>the defect was open, closed, resolved etc</td>
<td>Assurance</td>
<td>assigned to the defect</td>
<td>assigned to the defect</td>
<td>the defect</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>person assigned to the defect</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1 – an explanation of the Blocker’s document entry headings

---

4 Text in italics is quotations taken from contemporaneous notes of the discussion at the meeting

5 Figure 3, on the following page, shows the full content for a real defect as of May 31th.
The story of the field study

We’ve included the corresponding entries for the following defects from the blockers’ document itself, with one example included here for SJTY. Here we will concentrate on showing about what was said about some of the defects at this particular meeting and what actions were subsequently taken with respect of them. Each piece will start with the last 4 letters of the DTR as this is how the subject was started in the meeting itself.

As you can see in the most right-hand column of the following Figure 3 which is a direct extract from the blockers’ documents of the defect referred to as SJTY, it was not uncommon for patches to be admitted but which did not succeed.

<table>
<thead>
<tr>
<th>PRAV6ZSJTY</th>
<th>Open</th>
<th>1V6.0.1</th>
<th>Annette McGuire</th>
<th>Marjorie Portal</th>
<th>Learning Extension</th>
<th>Advanced search functionality not working</th>
<th>16/4 - No WS yet</th>
<th>Robin is working on it - hopefully tomorrow</th>
</tr>
</thead>
<tbody>
<tr>
<td>180 points</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>19/4 - Marjorie to investigate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20/4 - Patch should be available today</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>23/4 - need to patch on latest build</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>24/04-Patch tested, still an issue</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>26/4 - Terence investigating</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>30/4 - still investigating</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1/5 - Only course name search is broken - Terence says this should work in new deployment</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>02/05-Barack to investigate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3/5- still an issue on 0427</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4/5 - Barack investigating</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8/5 - Terence</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9/5 - Appears to be working on 0503 build - Barack</td>
<td></td>
</tr>
</tbody>
</table>
The story of the field study

**Figure 3: Example of DTR entry in blockers’ document**

<table>
<thead>
<tr>
<th>Date</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>15/5</td>
<td>Checked this on 12/05 build. Advanced Search screen appearing but search is not working</td>
</tr>
<tr>
<td>16/5</td>
<td>This issue is working fine now on Snoozy but functionality breaks when Cesc's patch is applied. Terence is currently investigating. DTR itself has been pended</td>
</tr>
<tr>
<td>16/5</td>
<td>FVT tested this &amp; it is still a problem, Terence asked this be reassigned to Marjorie as it is a UI issue.</td>
</tr>
<tr>
<td>17/5</td>
<td>UI to follow up with Terence</td>
</tr>
<tr>
<td>21/5</td>
<td>Being investigated today</td>
</tr>
<tr>
<td>22/5</td>
<td>Fix planned for end of this week</td>
</tr>
<tr>
<td>25/5</td>
<td>Still wip</td>
</tr>
<tr>
<td>29/5</td>
<td>Plan to have fix maybe later today</td>
</tr>
<tr>
<td>30/5</td>
<td>Patch planned for today</td>
</tr>
<tr>
<td>31/5</td>
<td>Patch expected tomorrow</td>
</tr>
</tbody>
</table>

The defect SJTY (in Figure 3) was related to the advanced search functionality of the PLE product. It had first appeared a month and a half before. Over the previous few days, various comments had been noted in the blockers’ document such as: “plan to have fix maybe later today”; “patch planned for today”. Now a senior developer reported that “patch expected tomorrow” and the project manager added the new target date and this variation on the standard comment to the blockers’ document. There was no argument about the repetition of attempts, everyone being aware of how unpredictable defect resolution can be. The patch
The story of the field study

would be found to be only intermittently successful over the course of the next week and further changes to the code had to be made to resolve it.

Defect SFUA was “a tricky one”. The problem had been identified as a communication issue between PLE and the underlying portal technology, and a patch was provided by Development. Now one of senior developers reported to the meeting that the patch had failed. It would be almost another week before another patch successfully addressed the problem. However, that patch also had the effect of breaking the advanced search function and had to be removed or “backed out” from the build. It would be a further week after that again when the problem was resolved completely and came to be removed from the daily blockers’ document.

Defect UE5Y was described by the manager as a “heavy” one, with blocking a whole functional area, in fact it “blocks 550 points - all of course collaboration & discussion threads”. It had been reassigned to a different developer three weeks previously. One week before, he had sent an update to the Project Manager by email indicating that he would need four weeks to tackle the problem as he had other work focused on feature implementation which took priority (and needed to be addressed first). With this in mind, the name of the defect was still read by the project manager even though he expected no further input. There were no comments made at all, and so they went on. It would be another month before this defect could be investigated and then it was found to be a duplicate of another defect which had been resolved in the meantime.

For Defect LEPU, the history of actions for which from the blockers’ document as on May 31st is shown in Figure 4, patching didn’t make sense. There were both too many files to be changed and the contents of those files had to be changed significantly. Too large a number of changes in files can have the potential to affect too many other functional areas. Such changes in these functional areas would require their retesting thus increasing the work associated with the application of the patch. The aim of patches is that they be specifically effective to the area that is causing a problem, without involving modification to other unaffected areas. In this case the fixed code files were instead committed in the more ordinary fashion to the code repository, and it was anticipated that the resultant changes would show up in the next build to be used by FVT. There was nothing else to do at this point but wait for the code changes to appear in a later build.
The story of the field study

<table>
<thead>
<tr>
<th>RSOL72LEPU</th>
<th>Open</th>
<th>V6.0.1</th>
<th>Raquel Sohoni</th>
<th>Jim Lan</th>
<th>Portal Learning Extension</th>
<th>Learning Version Number &amp; Build Date is not available anywhere from the UI</th>
</tr>
</thead>
<tbody>
<tr>
<td>10/5</td>
<td>Sinead to reassigned</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/5</td>
<td>reassigned to Jim. Currently WIP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16/5</td>
<td>Waiting on info. from Terence</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18/5</td>
<td>no update</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21/5</td>
<td>still WIP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22/5</td>
<td>Fix will be in next build 0523</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24/5</td>
<td>not in the 23/5 build.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24/5</td>
<td>To be followed up with Jim</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24/5</td>
<td>delivered it at 10:30 on the 23th of May, so should be in 2405 build or later</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 4: History of entries for DTR - LEPU as of May 31st**

In the case of defect AF72, a patch had been provided which seemed initially to work. The code from the patch was then subsequently committed to the source control and included in a previous build for verification, but now the testers reported that the problem still hadn’t been solved. The status of AF72 now turned red again (designating it as a current blocker), after having gone through a yellow phase (a resolved blocker pending confirmation). This issue also affected the performance group’s testing. Changes were made which addressed the performance issues but it remained in the blocker’s document table for the daily meeting until some minor aspects which didn’t affect performance had been resolved completely.

PGXN had been just discovered: and was of an API not displaying results. This was the first time it had been discussed at the meeting. When the API test was run the expected results were not being returned. As it was a ‘New’ blocker, there was nothing more to contribute at this meeting in terms of its status than to note its appearance in the blockers’ document. A fix would be attempted a week later and be unsuccessful, a later attempt one month after this meeting would be initially partially successful, before leading to the defect being re-opened, and it would be the end of July before it was confirmed as being resolved.

HFB6 (shown in Figure 5) proved to have a successful outcome. Three days previously it had been suspected of not being a defect, now this had been confirmed and the status finally turned green and the project manager noted in the comments column that it had been “Pended as NAB” (Not A Bug).
At this point the meeting concluded with the ending signal by a sign off over the phone from the project manager and with the people present in the meeting room dispersing and returning to their desks. Occasional conversations were struck up as they walk along. The meeting had lasted 30 minutes, 36 blockers were covered in that time, there were 8 changes in status, and 7 new blockers were reported. A few minutes after the conclusion of the meeting the minutes arrived in the email Inbox of those who attended and others along with a copy of the updated blockers’ document, and a new work day was underway.

There are a number of artefacts used in this meeting and the meeting itself is worthy of further detail. We will look at each item, the software builds, the defect tracking reports in turn to try and provide detail not present in this specific meeting

4.7. **Software Builds at this field site**

We have spoke about the term software builds, commonly abbreviated to *builds*, in general already. Noting that it can refer either to the process of converting source code files into standalone software artifact(s) that can be run on a computer *'doing a build'*, or the result of doing so *'a build'*. The end result is that a software build is a collection of files constituting a product in a distributable package. Code repositories, which people are more familiar with, are designed so that software code is stored in a manner allowing incremental changes to be made and rolled back if the changes are found to be unsuccessful. The code repository does not need to be on site. The software builds to be used by Testing were created nightly at a central location located on the East Coast of the USA and distributed or made available to all other project locations the following morning.

In the case of our field site, a local version of the code repository was maintained and a single build engineer at the local site checked in all the changed files for the day in order to reduce the number of people who had access to the central repository on the US East Coast. A pre-build was run by the local build engineer to flush out any code conflicts that might break the main build. A broken build arises when some problem occurs in the run of the build process that results in no completed build being produced. This could mean that the build failed some basic acceptances tests such as failing to install properly or start up once installed. Build breakages are very, very rare.

---

**Figure 5: History of entries for DTR – HFB6 as of May 31st**

<table>
<thead>
<tr>
<th>PCRX73HFB6</th>
<th>Open 1</th>
<th>V6.0.1</th>
<th>PARAG ANDERSON</th>
<th>Fintan Clancy</th>
<th>Portal Learning Extension</th>
<th>Trace service is not activable</th>
<th>mailed to Seol Young E.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>29/5 - Possibly not a bug</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>30/5 - Pended as NAB</td>
</tr>
</tbody>
</table>
Builds are automatically produced each day, though not every build that is produced ends up being used. Those builds that are used for FVT Testing are replaced on the test servers such as, Bert, Ernie, or Scooter roughly every week to ten working days. Meanwhile, developers are also working on builds on their local machines but these are more up to date builds taken from daily archives which the Deployment engineer produced for them.

While most builds were commonly referred to by the date on which it was produced, a build can also have a label. This naming of builds by date assists in recall the build when DTRs re-emerge and as a means to encapsulate the organisation memory as it is in the process of being created. Within the blockes’ document, build are identified is by means of a build number e.g. PLE6.0.1_20070821.0000. In the meeting the shorthand verbal reference is Month-Date e.g. 5-21

This is broken down as

<table>
<thead>
<tr>
<th>Section</th>
<th>The Product Acronym</th>
<th>Version Number</th>
<th>Date the Build was created on:</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g.</td>
<td>PLE</td>
<td>6.01</td>
<td>2007 08 21</td>
</tr>
</tbody>
</table>

**Table 2 – an explanation of the build numbering system**

A build may also be labeled at specific points in the schedule such as for milestones. Amongst those milestones are UI (User Interface) Freezes, Beta Freeze, and Feature Freezes, Code Freeze, Gold / Release Candidate Freezes.

Thus, software code is not normally made accessible to those outside the Development group in particular to those who are testing the software product. Instead Testing and eventually customers use software product builds, as generated from the content of these code repositories. So it is not software code that is being tested rather it is the behaviour of builds generated from this code.

**4.8. Defect Reports**

The defect tracking report (DTR) or ‘bug’ report is a single document referring to a specific defect in the defect tracking database. The defect tracking database is an implementation of a Lotus Notes application. Some characteristics of the defect report tracking application are that it is
The story of the field study

a) Lotus Notes based  
b) Has Views that show defect reports are centrally stored and accessible to all  
c) Embedded File Attachments are allowed i.e. Screenshots as files or other files.  
d) Stand alone application  
e) Mouse based navigation  
f) Free text entries  
g) Histories of changes to the defect report

4.8.1 Ideal form of a Defect Tracking Report

To be effective a defect report must encapsulate the relevant detail required to reproduce the defect, and record the steps leading to its resolution. As shown in Figure 6, the defect report ID was generated automatically based on the User ID of the person creating it, typically, but not exclusively, a member of the test team, e.g. DKSR45XLNY. The full defect ID is 10 characters long with the first 4 characters derived from the name of the person who has created the defect report, using their username for logging into the DTR system. The DTR ID stays the same from then on.

Those looking at the defect report can use the first 4 characters of the ID as a ready prompt to if they familiar with the team members and prior defect reports to suggest who created this report. Because of the ID being somewhat cumbersome for use at the daily blockers’ meeting a form of shorthand had emerged within the team that the last 4 characters are used in conversation to identify which defect was being discussed.

4.8.2 Actual format of the Defect Tracking Report

The format of the defect tracking reports is in large part highly structured allowing for the selection of options from pre-populated lists such as which Test Phase was the defect was found during, or the Severity of the problem – highlighted in Figure 6. This is combined with the facility for the embedding of text, graphics and other files in specific areas of the reports. Highly structured reports allow for significantly greater consistency across the defects as the same language and terminology is used. In practice there exists a great desire for lightweight reports from those who have to input the data as the practitioners regard entering the same information or data multiple times in different places as unnecessary waste and duplication. In Figure 6 you can see the top area of the DTR which has been populated from drop down lists of option.
The story of the field study

Figure 6: Defect Resolution Process Mundane repetitive detail

The opportunities that arise with defect tracking reports affords us more of an opportunity to gather accounts of the work associated with the defects as they are actually encountered. Resolving the defects includes the work of accounting for the resolution done while the work of resolution is taking place. Defect reports are concerned with how this work is performed, and how such accounts are created, made available to others, interpreted and used by other people. The other people may be novices or other experts with knowledge of the field.

It is entirely possible for such a report to contain more information than that listed above; however if it did not at minimum contain the details listed above, it would not be of much use. Other information typically contained in a defect report would be a) details of the environment obtaining on the machine or system at the time the defect occurred, b) the steps taken or task being performed when the defect occurred, c) sample data that was entered as part of this task, d) the test script used or a link to the more detailed description of the script.

---

6 Some of the details have been obfuscated to perverse the anonymity of the field site and the workers.
The story of the field study

or area of functionality that the test was intended to examine and e) any outputs from the system, error logs etc. A history of significant edits made to the defect report may be stored in sequence, as shown in Figure 7, to aid comprehension.

4.8.3 Tracking changes in the Defect Tracking Report

It is typical in software development that while one person will have found the defect many people will thereafter work on attempting to resolve it. Tracking their editing changes is a simple way to highlight this.

Defects are found in particular builds and resolved in specific later ones, the defect reports and the builds are each altered in turn and in a sense exist as interlocked objects, each changing in step with the other. A change is made to build to resolve a defect and a change is made to the defect report to reflect this, if the change is effective the defect report is updated accordingly and the change becomes a permanent part of the software product. If the change is not effective, then the report changes to reflect this and more work is done.

---

Customer Information

3rd Party Bug Information

SPR History: SRLXL, Clicking Status icon leads to error - Portal unavailable

29/03/2007 17:54 CAMDB04: [Validated - Open/Reproduced in build _20070323
Assigned to Dev:

29/03/2007 17:58:20: [categorized and assigned SPR to QA: Dev] for Project Area: UI - LMS Enrollment UI

23/04/2007 12:15:58: [categorized and assigned SPR to QA] for Project Area: UI - LMS Progress Management UI

23/04/2007 12:16 CAMDB04: [Reassigned to Dev:

27/04/2007 13:03 CAMDB04: [Verified on beta.com. When Status is clicked, gets the blank page as shown in the screen shot. See my comment

27/04/2007 13:04 CAMDB04: [Verified on beta.com. When Status is clicked, gets the blank page as shown in the screen shot. See my comment

11/05/2007 10:36 CAMDB04: [Reassigned to Dev:

24/05/2007 12:05 pm CAMDB04: [Dev Action - Fixed in build _20070630
Platform(s): All Platforms Assigned to QA:

27/09/2007 01:32 PM CAMDB04: [Verified - Fixed in build _20070615
Platform(s): tested: All Platforms

17/09/2007 21:37 [Local Replica]
An agent has updated this doc? "Regression Status has been set to "Needed"
The story of the field study

Figure 7: History of state changes to the document and reasons for making them

Defect tracking reports are in active daily use throughout the course of the development work of the software project and are used to assist in the prioritisation and coordination of tasks. Members of the Test team frequently create, read and edit documents within the defect tracking system. The defect tracking application is accessible to all team members irrespective of their location or role within the team. However, the actions they can perform on the DTR are dependent on their assigned role in the team and in the Access Control List, which is part of how Lotus Notes manages individual users’ access to databases.

For example, at the field site the workflow of the defect tracking application does not allow a person, typically a developer who has made a code change and ‘pended’ a defect as being fixed to set the status of that defect to a ‘Resolved’ state or ‘Close’ it.

These state changes can be seen in Figure 7 in the ‘comments’ section of the defect report documents showing what individuals have done, e.g. who edited the defect report and what changes they made to it in addition to tracking the status changes that the defect report underwent. In the defect tracking system agents monitor state changes in the reports and updates are sent via email as individual notices to the associated tester or developer when they occur.

4.8.4 How defect tracking documents are seen by team members

The Defect report documents are regarded differently by people who have differing concerns due to their part in the development process. Defect reports have different meanings to people who are performing different roles in the development process, be they members of the Test, Development or Project Management team. Much of the background information for changes in the code is contained in the defect tracking application rather than the source control which ultimately contains only successful solutions.

For testers, the defect tracking reports are an outcome of their work and indeed constitute some of the few pieces of documentary evidence of what they have achieved and the work they performed. This is very important to keep in mind. If the testing work passes off without finding any defects, there can be little concrete evidence of their considerable efforts.

For developers, defects are potentially a critique of their work, especially those that involve the failure of a function to perform as intended or expected. Experienced developers we spoke to recognised that defects will inevitably occur. Hence, the defect reports are not taken by them as a personal criticism of their work. The resolution of the defect is a joint
The story of the field study

undertaking by the developer and the test team. We outlined in the introductory chapter as shown in Figure 8 the place that defect reports occupy in our simplified representation of the development lifecycle. However, individual defects travel their own version of this cycle, some only traversing it once while other require many circuits of the cycle before reaching resolution.

![Diagram of defect lifecycle](image)

**Figure 8: The relationship between builds and defect reports**

In order more fully to understand the process and intention behind the patching process, we need to look more closely at the life cycle of the defect report itself. The defect goes through various states from the point when the defect is first uncovered and written up into a report, until the product is finished and released.

The process of finding and verifying a resolution of the report is a joint undertaking between the person who opened the report, usually from the test team, and the developer who is responsible for resolving it. Once created in the system the defect report may have multiple people assigned to it during the course of its life cycle. However, only one person can be designated at any one time as responsible for its resolution and for the confirmation of that resolution.

**4.8.5 Documenting the Defect in the Tracking Report**

The level of documentation required is dependent on when the contextual information is available to the others. A distributed environment can be detrimental to bug reporting, since there situations can arise where the best thing would be if the person discovering the defect could immediately show the defect occurring to the developer who would have to fix it. Doing this would allow them to tease out all the possibilities right there and then. On the other hand, this could mean the work involved was never captured and neither the tester nor the developer would have any documentation to assert that they had spent time on this issue.
The story of the field study

Figure 9 shows an example of the steps necessary to reproduce a defect.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Expected Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Logon to Learning Portal as Extension</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>Click Launch &gt; Collaborative Learning &gt; My Learning</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>Select Enrolled Learning from the Leftlist in My Learning Area</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>Select Certificates Tab</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>Enrolled Certificate page is open</td>
<td>Progress Details page for individual course will display</td>
</tr>
<tr>
<td>6.</td>
<td>Click on Status link for a course (note: if there are no courses under the Certificate, go back and enroll in a certificate that has courses under it)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 9: Screenshots of the Steps necessary to reproduce the defect

In the figure 10 we can see the confirmation that this same defect was resolved with the expected text being displayed in the appropriate place. These screen grab images are often what constitutes the bulk of the additional file content attached in the Defect Tracking Report.
The story of the field study

4.9.   Blockers

*Blocking defects*, as we have mentioned previously, are those defects in functionality that prevent the Testing team from examining functional areas of the software product to completion. The blocking defect, or ‘blocker’, the term used by the PLE team, refers to any defect that stands in the way of the progress of the testing effort. The software problem or defect that is causing the blockage prevents the testers from running their tests to completion in accordance with the test plan. Thereby leaving means some areas of the product untested. This results in delays to the completion of the testing and has a knock on effect on the overall schedule.

During the course of our observation, we witnessed some debate over the definition of what constituted a blocking defect. The PLE project manager at one meeting defined a
The story of the field study

blocker as any defect ‘that prevents you from actually doing testing in an area’. This was in response to a query from the performance team of this definition as a consequence of the test situations that they had encountered, e.g. where code performance was so inefficient that the time elapsed while loading test cases prevented Testing progress to completion in the allotted time but had they sufficient time then it would have completed. Hence from the perspective of the performance team, this was a blocker, yet to the rest of the team and the project manager, it was not considered a blocking defect, as they were actually making progress, albeit very slowly. These issues were resolved as development work progressed and the performance of builds improved.

In such a complex endeavour as a software project, not all defects can readily belong in a definitive manner to one area and not another. Issues could be raised as being defects in one area but which were in fact caused by a problem in another. This happened in one instance with the Data Loader. After investigation by the developer they were shown to be problems with the displaying of the information rather than the generation of it meaning it was a problem for the UI people. In another instance - DTR (#SPII746DXQ) an issue from the dataloader manifested in the performance testing, a member of the performance team wrote the text below in the DTR

"This issue revealed itself while rostering but I believe it is a command queue performance issue that needs to be addressed before we do the 100K enrolments. We also get an error message for which Davie entered an DTR DSCY745N3Z: Rostering 10000 users in dataloader causes error message.

When loading 10K records in one batch it process 34 records a minute. It would take almost 5 hours to complete.
When loading 1K records in one batch it process 190 is h records a minute. It takes less than 1 hour to do 10K.

A snapshot of the database shows that the query highlighted below is taking almost a second to process with a 10K batch while it takes on avg 0.05 seconds with a 1K batch.

We have 100K enrolments to do and it is not practical to break it down in 100 x separate 1K batches. Would it be possible to get a patch?"
The story of the field study

And the response was "Yup - this is an obvious missing index (although I'm gobsmacked that it would appear in a table with less than 10K entries). Fix should be easy enough. Thanks for doing the snapshot work this quickly.

This is actually an important (and first!) performance issue as even without dataloader it could have affected content deployment and several other activities, especially on a heavily loaded server.

I'll supply an index for this tomorrow."

Though the defect can come to light during the performance testing, in this case the problem was actually one that would affect the entire product which used this feature once testing had begun in other areas. This is an example of the different nature of the issues that the performance team faced which shaped what their different view of what was a blocker. The problem was not that the individual functionality was blocked but that the overall length of time involved was so long that the set of tasks could not complete before they would have to be started again with another newer build.

Irrespective of where the cause lies, the important focus is on the issue being resolved. Fixes to defects could also occur indirectly, in one instance where extensive code changes were made to address issues of handling dates properly. These changes fixed a problem that had not yet been investigated. It was not sure why but that is not investigated. It was not possible to wait for one area to be 100% completed before proceeding to work on another area.

4.10. Blockers’ document

The blockers’ document shown in unexpanded form in Figure 12 existed for a number of reasons. The functionality of the defect tracking database included the ability to view all blocking defects. However, the actual generation time for this database view was quite slow. In part to overcome this limitation during meetings but more importantly, maintain his control of what he adjudged to be "actual" blocking defects the project manager, chose to maintain a separate blockers’ document.

This was a practice that other project teams at the site used from time to time when they deemed it necessary and which the PLE team had used on a previous release. The content of the blockers’ document was a subset of the defects maintained in the defect tracking database,
The story of the field study

and it was maintained as a stand-alone entity within Lotus Notes and controlled by the project manager. The blockers’ document was populated with defects sent by emails directly to the project manager. Whilst these defects remained contained within the defect tracking system, the project manager maintained control on whether he considered the defect to meet his criteria to be a blocker. The blockers’ document consisted of a table with basic information about the defect as show earlier in figure 11 and a link (similar to a hypertext link) to the actual defect report in the defect tracking database.

The important reason is one of control in that it allows the project manager to decide what is and is not included as a blocker and which is then opened up to the possibly of being patched. While not all blockers are patched, it is the case that non-blockers are not patched.

This document is in daily use and tracks which defects that are blocking the progress of testing. It is reviewed at the daily meeting to enhance public awareness and control. There is a master version of this document that contains all the blockers encountered so far in a database. A subset of the content of this document containing those currently active or unresolved defects is what forms the subject of the daily meeting. The meeting is timed to last 30 minutes in total.
The story of the field study

**Figure 11: The collapsed blockers’ document**

The small green triangles indicate where the text can be expanded to show hidden sections, in those collapsed sections are the tables we have shown earlier in the vignette and which we can see more examples of in Figures 12 to 15. A traffic light colour code was used for the background of each row in the document to readily highlight awareness of the current status of the blocking defects. The colour was the background to the text in the blockers’ document table.

- Red: current blocking issues
- Yellow: defect no longer blocking the progress of testing due to patch. The defect has to remain on the list until tested as an ordinary code change in a build.
- Green: defect resolved in a build without a patch
The story of the field study

We see each type in the following examples from the blockers’ document. These are the individual entries in the blockers’ document as they appear when the document sections have been expanded by click on the small green triangles. Figure 12 below shows an open DTR

<table>
<thead>
<tr>
<th>Build date</th>
<th>DTR#</th>
<th>State</th>
<th>Sv</th>
<th>CS</th>
<th>QA</th>
<th>Dev</th>
<th>Proj</th>
<th>Description</th>
<th>Current Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>19/4</td>
<td>AMGE6Y8H</td>
<td>Open</td>
<td>1</td>
<td>V6.0. 1</td>
<td>Raquel Sohani</td>
<td>Seol Young Enright</td>
<td>Portal Learning Extensio n</td>
<td>Cannot see course details</td>
<td>This was not a problem on the original 1004 deployment but since it was redeployed this has appeared.</td>
</tr>
<tr>
<td>20/4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>19/4 - Paul has just restored Snoozy again so Seol Young will investigate now.</td>
</tr>
<tr>
<td>23/4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20/4 - still WIP.</td>
</tr>
<tr>
<td>24/04</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>23/4 - Fix will be delivered on during the week. Not a blocker atm</td>
</tr>
<tr>
<td>26/4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>24/04 - Patch may be required if the issue reproduced.</td>
</tr>
<tr>
<td>27/04</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>26/4 - Seol Young working on issue - not a blocker</td>
</tr>
<tr>
<td>30/4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>27/04 - getting exception for courses in curr/cert, but it’s a different exception</td>
</tr>
<tr>
<td>15/5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>30/4 - Fix will be checked in this week (today/tomorrow)</td>
</tr>
<tr>
<td>17/5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15/5 - Can now reproduce. Workaround, go into course details first and course details will be available.</td>
</tr>
<tr>
<td>24/05</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>17/5 - waiting for Seol Young to return</td>
</tr>
<tr>
<td>28/5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>24/05 - WIP</td>
</tr>
<tr>
<td>30/5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>28/5 - Seol Young plans to deliver tomorrow</td>
</tr>
<tr>
<td>5/6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>30/5 - Target deliveries for Thurs 0531</td>
</tr>
<tr>
<td>6/6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5/6 - delivery completed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6/6 - delivery has caused many problems - Sean, Sinead working on fixes</td>
</tr>
</tbody>
</table>

**Figure 12: Red entry from blockers’ document = as yet unresolved**
<table>
<thead>
<tr>
<th>Ticket</th>
<th>Status</th>
<th>Priority</th>
<th>Version</th>
<th>Assignee</th>
<th>Category</th>
<th>Issue Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOCR72KJC6</td>
<td>Pending</td>
<td>2</td>
<td>V6.0.1</td>
<td>Annette McGuire</td>
<td>Portal Learning Extension</td>
<td>Edit to Rating scale in skills area not saving</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Senan Connor</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Portal Learning Extension</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Edit</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>03/05</td>
<td></td>
<td>- Verified patch on bert(0205 build).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Change order from Low to high or vice versa</td>
<td></td>
<td>- is not working, while changes to Prof. level and delete working as expected.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4/05</td>
<td></td>
<td>- one outstanding issue (not a blocker) from original issue</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Change order from Low to high or vice versa</td>
<td></td>
<td>- is not working, while changes to Prof. level and delete working as expected.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8/5</td>
<td></td>
<td>- Patch should have fixed issue and delivered last week. To be confirmed with</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Raquel</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9/5</td>
<td></td>
<td>- Current issue</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Change order from Low to high or vice versa</td>
<td></td>
<td>- is not working</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5/6</td>
<td></td>
<td>- Fixed checked in, FVT can regress in today's build</td>
</tr>
</tbody>
</table>

**Figure 13:** Yellow entry in blockers’ document – patch verified as fixed, yet to be verified in build

<table>
<thead>
<tr>
<th>Ticket</th>
<th>Status</th>
<th>Priority</th>
<th>Version</th>
<th>Assignee</th>
<th>Category</th>
<th>Issue Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCRX73HFB6</td>
<td>Open</td>
<td>1</td>
<td>V6.0.1</td>
<td>PARAG ANDERSON</td>
<td>Fintan Clancy</td>
<td>Portal Learning Extension</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>mailed to Seol Young E.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>29/5</td>
<td></td>
<td>- Possibly not a bug</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>30/5</td>
<td></td>
<td>- Pended as NAB</td>
</tr>
</tbody>
</table>

**Figure 14:** Green entry in blockers’ document – verified as ‘Not a Bug’ no Development action needed

<table>
<thead>
<tr>
<th>Ticket</th>
<th>Status</th>
<th>Priority</th>
<th>Version</th>
<th>Assignee</th>
<th>Category</th>
<th>Issue Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KBMY73FGL8</td>
<td>Open</td>
<td>1</td>
<td>V6.0.1</td>
<td>Denis DeYoung</td>
<td>Philip J. Fry</td>
<td>Portal Learning Extension</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>23/5 - Planning to deliver fix for 0525 - blocks virtual courses</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>28/5 - FVT regress in today’s build</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>29/5 - Fix expected to be delivered today</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>30/5 - FVT to regress in today’s build</td>
</tr>
</tbody>
</table>

**Figure 15:** Green entry in blockers’ document – patch verified as fixed in build after code submission

Once a blocker is resolved the background for the associated entry in the blockers’ document becomes white. After that it is removed from the daily version of the blockers’ document to be used at the meeting but it remains in the master version on contained in the database. Hence white status blockers were no longer discussed at the blockers meetings nor are they included in the sub-table which is mailed out as part of the daily blockers meeting. However,
they continue to be retained in the master document which is contained in the project database.

4.11. The daily meeting

The practice of holding the daily meeting is part of the larger ecology of practices of the PLE team. It is supported by and dependent on the other local practices, such as the pre-build, the use of DTR system to distribute patches and others. The format of the daily meeting cannot simply be pulled out and used elsewhere without modification. There are contrasting views held by each of the participants as to where defect reports fit in their work. In particular this is highlighted through our examination of how the blocking defects are discussed in the blockers meeting. The manager looking at the overview facility in the DTR system can see the outstanding DTRs and to use this information to assess if the schedule is on track or not. This is done by comparison with metrics from previous similar sized projects.

The daily blockers meeting was normally chaired by the project manager either via phone or physically present on-site and included representatives of the Development, test and Deployment groups along with others such as the performance team. In his absence another experienced member of the team typically from Testing would chair the meeting in his place. As part of their preparation, prior to the meeting the project manager would copy the previous day’s blocker’s document into a draft email to the PLE team; this email was then used as the basis for the agenda and discussion at the meeting, and was circulated to the team upon conclusion of the meeting. In the daily blockers meeting, all the defects were gone through sequentially as they appeared in the blockers’ document. As the project manager proceeded through the list, he would read the last 4 digits of the defect id and then wait for the other participants to intervene and share relevant information they had on the status of that specific defect. The order is known and long established, though if the circumstances required the order could and did change with the agreement of the participants, i.e. discussing the Beta Blockers earlier as the Beta release date grew closer.

During the course of the meeting the current status of each blocking defect was touched on if even for only a moment, efforts for the rest for the coming day were agreed and coordinated and resources were allocated or re-allocated as necessary. To maintain overall awareness within the group it was vital that each defect on the list was touched upon. New defects were added to the bottom on the section of the table they would last to be read but as defects were resolved and removed and newer defects added that they moved further up the table. The updated blockers’ document that resulted from the meeting was circulated to all the
Development and test team members by the project manager immediately following the meeting.

4.11.1 To patch or not to patch

What is a key part of the decision, of whether to patch or not, was the tension between how much additional work efforts the patch would allow Testing to do versus what is the risk to the rest of the build of the code change that the patch involves. The more code that is changed; the greater the apparent risk there is of having a negative effect on other areas of the built product. This would in turn delay testing work in these other areas. This potential for an impact on other areas was one of the reasons behind having representatives from each subgroup at the daily meeting. These representatives would then be expected to communicate back to other team members about what changes were underway. If a change, resulting from the deployment of a patch, negatively impacted on an area of the product and the representatives of that product area were not in attendance or did not read and respond to the meeting minutes as circulated immediately after the meeting then they were not in a position to complain. The daily blockers’ meeting was accepted as the venue to raise any objections or concerns about the decision to proceed with a patch.

Serving as a counter point to the uncertain consequences that patching could have on other areas of the product was the reality that the verification of a patch code actually increased the certainty around the associated code fix when it entered the code base. Patches did not form part of the code base until they had been verified by Testing thus limiting any negative impact on the production of the build itself. Were the code changes submitted directly into the source control, any problems that arose from the change would not be as easy to reverse.

Information on the current state of progress made to date is gathered at the daily meeting. This information is used by the workers to make timely and locally applicable decisions which enable an optimisation of local resources and to build up and maintain a picture of the wider work and resource environment and make more globally relevant decisions. By building incrementally by having the meetings at short intervals they lessen the burden on the workers involved in maintaining a picture of the current state of work outside their own. Indeed, the picture of the state of the work from even of a few weeks ago can seem very distant with so much happening in between.
4.12. **The Patching process**

The types of blocking problems that the patching is addressing are varied. They can include fixing functionality problems, replacing graphical elements in the User Interface, correcting data loss or errors in the display of data and improvements to the general usability or performance. Problems in each of these areas can lead to a loss of testing time. In many ways the particulars of the problem itself are a lesser consideration than the consequences of the problem as it affects the overall project schedule.

The patch itself is a relatively small piece of software code in one or perhaps several code files designed to update or fix problems within a computer program or its supporting data. In this case the local ‘patching’ work practice emerged to address the iteration gap that arose in the availability of software builds, and the subsequent negative effect on the resolution of blocking defects. Rather than checking in code to the code repository and waiting for a compile, build and install cycle to complete in order for Testing to be able to validate the fix, the developers chose to patch currently installed versions of a build in situ to expedite the work. Thus the standard practice of checking code fixes into the code repository for inclusion in the next build was temporarily bypassed. This served a dual purpose, if the patch was successful it allowed Testing to continue and it also meant the effectiveness of the code changes was verified before they were submitted to the source control.

4.12.1 Origins of the patch

The patch comes into being via a process which is separate to regular software development work. It involves a practice that source control systems were not designed to support. While the system was not designed to support it a means of working as been arrived at that does allow for the practice to be supported. In this way the supporting software tools have not broken down as such but rather have been augmented as Gasser (1986) suggested addressing a need which has arisen from working with large complex software. This is most appropriate for a software product that requires considerable time to install and set up proportionate to that required for patched changes to the code e.g. it could take a day or more to install a new build from scratch but only 5/10 minutes to deploy a patch. Those at the site highlighted to us that such a practice would not be possible or desirable if the Testing nor Development work, in addition to the build process, had been distributed over a temporal distance. Such distribution would mean the deployment work affecting the build was occurring while Development or Testing was offline.
The story of the field study

The patch in the defect report appears as in Figure 17, in the section allocated to bitmaps and file attachments. The actual patch is the jar and jsp files circled in red. There are two patches here as the first changes were found to be insufficient to resolve the defect.

Workaround:

Other Info:

Bitmaps & Attachments:

**PATCH 1:** 17/05/2007 - 17:15

In addition to file from patch below...

Copy this file:

![Image](file.png)

`profile/App_profile/installedApps/MyLearningServer.ear`

and restart server...

**PATCH 2:** 18/05/2007

Copy this as `progress.jpg` into `PortalServer/installedApps/MyLearningServer/learningPortlet/htmlViewingCourses`.

Figure 16: New Patches as they appear when embedded in the DTR

The other attempted patches can be seen in a section which we have expanded in Figure 18, three other patches had been made in the weeks before this. Those earlier attempts to patch can be seen, indicated by the arrows, in the Figure 17 along with instructions on how to apply the patch.
The story of the field study

4.12.2 Retaining Control of Patching

The client/server architecture of this particular software product assists in the distribution of code patches. The code patches are deployed (typically by copying them into the appropriate file directory over writing the existing version of the code file) to a central
The story of the field study

location on the server and then each tester need only undertake minimal tasks to refresh their local setup, such as clearing their cache and re-launching their browser. It also allows the Deployment team to ensure that all testers are using a consistent version by pointing them in the direction of a specific server. However, even this benign situation can become rapidly confused when there are multiple server machines with differing builds and patch scenarios available.

For that subset of blocker’s defects that were patched the DTR was being used as a type of precursor code repository. The code files that it was thought might solve the defect was placed in the defect report. By doing so the project team was temporarily deviating from the normal process of source control with the knowledge of the manager and with the cooperation of the Development, Testing and Deployment teams’ members.

The means of installing the patch files was solely controlled by the Deployment engineer, who received the files and installation directions from the relevant developer via the DTR. By posting the patched files in the DTR this meant the information was also visible to the tester that a patch had been provided. The fact that a patch could be attached to the corresponding defect report meant that the solution to the defect was clearly visible to all members of the team. It provided a visible history of the work completed to date in attempting to resolve a particular blocking defect. However, on some occasions the developer would email the patches to the Deployment engineer directly in addition to attaching the files to the DTR.

The uncertainty that each new build of the software products presents needs to be dealt with in a manner sufficiently agreeable to all concerned rather than being avoided. As a course of action the practices of “Patching” did not always make sense and were not used in all cases where there was a defect ‘blocker’. In cases where too many code changes were involved, or too many interdependent components would be affected by the proposed change the standard development process was followed. The modified code was committed to the code repository, and the result was delayed until the next available build. Ultimately there were occasions when there was nothing to do but wait until the code fix would arrive in the normal manner as part of a build. This would negatively impact on the testing schedule and delay developers from moving their attention onto other features as they could not stop working on the area until testing had been performed and any defects present uncovered.

In the representation in Figure 18, the ClearCase source control provides the files that are use to create the build, the build is under the control of the Deployment team and used by QE/SVT who uncover defects in it and write them up in the Defect reports. These defect
The story of the field study

reports either lead to a fix or patches being founded to resolve the defect and both are submitted or applied as required to the source control or the build. Once the patch has been successful it becomes a fix.

Figure 18: the Patch’s place in the development Process

4.13. Chapter Summary

In this chapter we have represented the background to the project and the consequences of the iteration gap, and through a vignette of a daily blockers’ meeting shown some of the defects they were confronted with on a regular basis. We then proceeded to examine in more detail the artefacts they use as part of their practice of addressing this iteration gap. This includes the defect tracking report, the patches as they appear in the defect report, the blockers’ document and the individual entries for patched defects in the blocker’s document. We have detailed how the patching practice came about and how it was used from day to day and some of the consequences of its use.
Chapter 5 – Discussion

5.1. Introduction

In this chapter we discuss the scale and the frenetic nature of the actuality of the work undertaken which we have presented in chapter four. Three strands stand out for us from our work in the field site. They are the iteration gap that results from the distribution and how they deal with this, the maintenance of the practices involved in dealing with the iteration gap by the visibility afforded by the practices themselves, and sheer scale and complexity of the work being undertaken at our field site.

Their work practice shifted to accommodate the contingencies they were confronted with. Work practices that had the elasticity to be adapted while making use of pre-existing artefacts, such as defect tracking reports and installed software builds, in combination with new artefacts that they created such as the blocker’s document. Key to all this is how the visibility afforded it by one practice, the daily meeting, contributes to the maintenance of the overall collection of associated practices of ‘patching’. All of this takes place against the backdrop of the urgency of the ‘code race’, that ever-present pressure that exists to get the work done by the finish line, of the release date.

5.2. Dealing with the distributed reality of software development work

As we have seen this is a highly pressured work environment, dealing with a large number of changes and a lack of day to day certainty as to the precise extent of those changes and their effects. Therefore, the PLE team existed in a state that required continuous contingency planning and reprioritization of their work tasks. Dealing effectively with the changes and maintaining the daily work cycle requires continuous minor adjustments. This needs to happen in order for everyone to remain synchronised and on course to releasing a quality product by the deadline.

To accommodate the unpredictability of their work the PLE team had creatively found a way of adapting their pre-existing practice that kept their work flowing. In arriving at this adaption, they were presented with many options and considerable freedom existed for the
team members in what choices they could make. They were able to deal with the uncertainty they faced by means of the heterogeneity surrounding them so that progress was made because of this variety and complexity and not in spite of it.

The software team is part of an organisation that allows the team members considerable flexibility in choosing to organise their work. Yet they had found themselves faced with a method of orchestrating their work that led to the exaggeration of the effects of a gap in iteration on the resolution of blocking defects. The practices of patching are used to support a work around that acted to reduce the effects of this iteration gap.

In addition, they were using these short, focused, daily meetings as coordination mechanisms, or ‘mechanisms of interaction’ (Schmidt and Bannon 1992) to reduce the overhead in ensuring continual readjustment of priorities whilst maintaining an overall awareness amongst the team of the current status of the work.

To an external observer, the situation that faced the PLE team might have seemed completely chaotic and uncertain; however, these people were calm and undisturbed in what looks like on the surface a permanent state of crisis.

The temporal rhythm inherent in the synchronisation of centralised builds and daily meetings is assisted by the frequent reference to and engagement with *interlocked objects*. These interlocked objects such as the software product builds, the blockers’ documents and the defect tracking reports link one defect to the next, one parcel of work to another, thus bridging the boundaries of one day to the next.

**5.2.1 A breakdown of DTR work**

This harsh reality of GSD work can be hard to grasp without the benefit of hard data. In order to give some idea of the scale of the problems they are faced with, we detail the number of the blockers in Table 3. The total number of DTRs increases and decreases over the period covered in the table presents the sheer numbers of blockers from week to week. The precise totals on individual days between each of these sample points might be slightly different, but these are a broadly representative sample of the size of the situation they face.
Table 3 – Breakdown of the categories of the DTRs over a four month period

<table>
<thead>
<tr>
<th>Date</th>
<th>BV T</th>
<th>Smokey Test</th>
<th>FV T</th>
<th>Performanc</th>
<th>Beta Blocker</th>
<th>Instal l</th>
<th>TV T</th>
<th>GVT / Accessibilit y</th>
<th>SV T</th>
</tr>
</thead>
<tbody>
<tr>
<td>May 16th</td>
<td>3</td>
<td>6</td>
<td>22</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>May 23rd</td>
<td>4</td>
<td>4</td>
<td>17</td>
<td>1</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>May 30th</td>
<td>1</td>
<td>3</td>
<td>28</td>
<td>2</td>
<td>-</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>June 6th</td>
<td>1</td>
<td>3</td>
<td>26</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>June 26th</td>
<td>1</td>
<td>4</td>
<td>20</td>
<td>5</td>
<td>9^7</td>
<td>-</td>
<td>-</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>July 6th</td>
<td>1</td>
<td>9</td>
<td>15</td>
<td>5</td>
<td>19</td>
<td>-</td>
<td>1</td>
<td>9/14</td>
<td>3</td>
</tr>
<tr>
<td>July 13</td>
<td>-</td>
<td>6</td>
<td>20</td>
<td>3</td>
<td>5</td>
<td>-</td>
<td>-</td>
<td>11/17</td>
<td>6</td>
</tr>
<tr>
<td>July 20th</td>
<td>1</td>
<td>4</td>
<td>19</td>
<td>4</td>
<td>5</td>
<td>-</td>
<td>-</td>
<td>8/14</td>
<td>4</td>
</tr>
<tr>
<td>July 26th</td>
<td>-</td>
<td>5</td>
<td>23</td>
<td>4</td>
<td>5</td>
<td>-</td>
<td>-</td>
<td>6/6</td>
<td>4</td>
</tr>
<tr>
<td>August 7th</td>
<td>-</td>
<td>3</td>
<td>24</td>
<td>4</td>
<td>5</td>
<td>-</td>
<td>2</td>
<td>10/6</td>
<td>17</td>
</tr>
<tr>
<td>August 13th</td>
<td>-</td>
<td>2</td>
<td>29</td>
<td>5</td>
<td>5</td>
<td>-</td>
<td>5</td>
<td>11/6</td>
<td>19</td>
</tr>
<tr>
<td>August 20th</td>
<td>-</td>
<td>3</td>
<td>14</td>
<td>5</td>
<td>5</td>
<td>-</td>
<td>3</td>
<td>4/-</td>
<td>15</td>
</tr>
<tr>
<td>August 27th</td>
<td>2</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>5</td>
<td>-</td>
<td>3</td>
<td>4/-</td>
<td>12</td>
</tr>
<tr>
<td>Sept 6th</td>
<td>-</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>5</td>
<td>-</td>
<td>3</td>
<td>4/-</td>
<td>6</td>
</tr>
<tr>
<td>Sept 13th</td>
<td>-</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>-</td>
<td>3</td>
<td>4/-</td>
<td>6</td>
</tr>
<tr>
<td>Sept 20th</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>1</td>
<td>5</td>
<td>-</td>
<td>3</td>
<td>4/-</td>
<td>7</td>
</tr>
<tr>
<td>Sept 27th</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>1</td>
<td>5</td>
<td>-</td>
<td>3</td>
<td>4/-</td>
<td>7</td>
</tr>
</tbody>
</table>

^7 The Smoke Test consists of a number of tests performed to see if the build has minimal functionality necessary to proceed with a full FVT test cycle. The name comes from the practice of filling a vessel that should be water tight with smoke in order to detect leaks.

^8 The Beta Build was made available for selected customers to download on July 13th from then on the remaining Beta blocker defects continued to be included in the blockers table but were from that point on no longer actually preventing the release of the Beta Build. The Gold Candidate build was produced on September 12th. The release of the product was made via the net by means of what was termed an ‘i-release’
From the blockers’ document that was last updated on June 26th there had been 216 blockers recorded and dealt with. Of these the breakdown was as follows in table 4

<table>
<thead>
<tr>
<th>BVT</th>
<th>Smoketest</th>
<th>FVT</th>
<th>SVT</th>
<th>Performance</th>
<th>Beta</th>
<th>GVT / Accessibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>22+101</td>
<td>6+13</td>
<td>10</td>
<td></td>
<td>15</td>
</tr>
</tbody>
</table>

**Table 5 – Breakdown of the functional categories of all DTRs**

<table>
<thead>
<tr>
<th>Functional Area /Sub-Area</th>
<th>Common</th>
<th>Delivery</th>
<th>LCMS</th>
<th>LMS</th>
<th>Skills</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data loader</td>
<td>103</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Back End Server</td>
<td>N/A</td>
<td>361</td>
<td>24</td>
<td>85</td>
<td>552</td>
</tr>
<tr>
<td>UI</td>
<td>N/A</td>
<td>390</td>
<td>42</td>
<td>142</td>
<td>774</td>
</tr>
</tbody>
</table>

There were a total of 2107 defect reports entered into the defect tracking system. The breakdown by functional component area is shown in Table 5.

The defect reports were created by 148 different people, in most cases by the person who uncovered the defect, but not necessarily always. There were 51 developers who had defects assigned to them to resolve. However, 8 of them had more than 100 defects assigned to them and thus this smaller group were doing the bulk of the defect resolution work.

Blockers of one form or another form roughly 10% of the total defects found and dealt with during the course of the project. This is not the largest grouping of defects but they are sufficient in number to justify the effort expended in dealing with them.

All of the above numbers shows that a large software project presents problems of its own from the sheer size. Gaps in iteration in a smaller project are more manageable on an informal ad-hoc basis but larger projects require more.
5.2.2 Restricting the effects of patching to the local site

What was being done at this local site with the application of patches was confined to the local site. Locally applied patches did not have a knock on effect on what was happening at the other sites. Once the code changes were confirmed as capable of resolving the defect when in patch form, the code changes were submitted to the source control in the usual way so that the code could be later verified in a build. These changes were no longer patches but part and parcel of the build. However, if they had not been using the patching process, there would be a global impact as the turn-around time for the changes they made would have been significantly increased.

In Figure 19 is a representation of how the regular flow of software development work might be viewed. The work that the testing/Quality Engineering (QE) (SVT) team is performing on the Builds uncovers defects which are then documented in the Defect Reports. The Development team works to produce resolutions to these defects in the form of Fixes which are submitted to the Clear Case Source Control system. The next step involves the production of a new Build from this altered Source Control and the cycle begins once again with more testing work on the new Build, once it has been installed on a server. This work includes confirming if the fix has resolved the Defect and the uncovering of other new Defects.

![Diagram of interaction between the build, fix and defect report](image_url)

**Figure 19: Interaction between the build, fix and defect report**
This could be read as the source control begets the build which begets defects leading to code changes which begets the next build which feeds back into eliminating those defects while perhaps introducing or allowing more to be uncovered.

Both of the interlocked objects, software product builds and defect tracking reports are altered in tandem during the course of the group’s efforts to resolve the blocking defects. The builds are altered by means of the additional deployment of patched code files, after the builds have been installed and configured. The DTRS are altered by the parallel addition of the same patched files to the defect reports. Doing this makes visible to the test team the changes made to the software on the servers. Adding the patched files to the DTR serves to make visible to the testers the work the developer has done but that they, as members of the test team, are not necessarily able readily to see for themselves. In Figure 20, we show the place that the patch takes in development cycle. The patch exists between the defect report and the fix that enters the source control, and is applied to the build post install. It is under the control of the Deployment team. The QE team works on the installed build and writes defect reports, they have control over the Patches, or access to the source control or the Fixes submitted to it.

![Diagram](image)

**Figure 20: Interaction between the build and patch, fix and defect report**
5.2.3 Boundaries made visible

The boundary that exists between the work of Development team, who are primarily focused on producing code, and the Testing team, who are focused on examining the builds produced from that code, can be quite opaque. That such boundaries exist is made clearer by the necessary acknowledgment of the combined effort to make plain and visible what goes on ‘under the hood’. Specifically in this case the work on the test servers which are not in sight of the test team and is under the control of the Deployment team. Artefacts or objects such as software builds and defect reports are markers at the extremities or territorial boundaries of the work of Development, Deployment and Testing but are also points of intersection where exchange of work parcels takes place between them.

Whilst Star and Griesemer (Star and Griesemer 1989) were studying paper based artefacts, their work also applies here where multiple groups and individuals use the same artefacts for different purposes. The work of each individual alters the object in a manner that allows the work of the other to progress without being impeded. Development and Testing have quite divergent interest in the objects, builds and defect tracking reports. Developers see the builds as an end work product; an output of their code work. Testers see the builds as an input to or the raw material for their work. Meanwhile the defect reports are seen by the developers as an input into their code work while the testers see them as an outcome of their work.

These objects are not without their limitations, yet Development and Testing are facilitated, by the design of the objects and how they choose to use them, in making use of the same object for their differing short term intentions. The changes made are agreed in a transparent and visible fashion aided by the daily blockers’ meetings and with the visible attachment of the patched files in the DTRs. These objects, the builds and the defect tracking reports, are not static in form or content or isolated from their environment. Indeed, it is the constant human intercession with them that allows them to continue to be of use in the face of the impediments of the iteration gap.

Yet, without the practice of the daily meeting to make these intercessions visible, they would fail, as silent or hidden changes would undermine their validity. It is only with sufficient flexibility in both artefacts and practices that such changes can be made to allow for the defects to be unblocked. Both the internal content of the actual objects and the practice of their use are extended to facilitate a successful working around of the standard process to
resolve the defect blocking the testing work. Yet, the work around is temporary and limited in scope. First the code in the build on the server being used by Testing is patched, then it is fixed in the source control, finally it is resolved in the defect report once it has been verified. The work around of the practices of patching is the means to map out the path to a resolution. As we show later in more detail when talking about the iteration gap itself, it is primarily the consequence of failed attempts that are prevented from affecting the schedule by use of patching. There is a comparatively short term benefit if the first attempt to resolve the blocker works as planned.

5.2.4 Allowing practices to emerge

It is necessary that a workplace allow for the emergence of local practices, and that practice be open to adaptation to local needs and the appropriation of technologies even when that comes at the risk of misuses or less than optimal uses. Too closed a work environment, where strict rules dictate behaviour, will result in workers who are unable to overcome unforeseen events. The fear of misuse of technology and deviation from practice is not entirely unwarranted, but one that should be lived with in order to make the most of the opportunity.

However, the developers do not have direct access to the installed builds Testing is working on. Only the Deployment team has this access, and this is crucial to the Testing group’s acceptance of the deviation involved in the practices of ‘patching’. The Deployment team arrange with the Testing team a suitable time to take down the server, and also let them know once the new patch has been deployed so that the particular defect it relates to can be verified.

As a means to expedite the alerting of the Deployment engineer of the existence of the patch it can be sent to him by email by the developer, in parallel to its attachment in the defect report, in order for it to be applied on the selected current test builds installed at the local site. Not all builds on all servers are patched in the same way. This is to ensure that, if there are any negative effects they are confined only to the server and the testers who were verifying this particular defect.

Which specific server should to be patched for which defect was based on an agreement between the Testing team and the Deployment engineer. The Deployment engineer tracked which patches were deployed on which servers by means of his own tracking document that contained all the current patches for that build. When a new build was deployed, a new
document was created, by the Deployment engineer, which inherited from previous builds the patches that remained under ongoing assessment by Testing.

Once the patch was verified and work was again progressing in the blocked area, the developer checked the corresponding changes made in the patched code files into the ClearCase code repository, in effect retro-fitting to conform to the standard process. This was because they were not seeking to undermine the process of using central source control, but to enhance it, as if they were temporarily and locally creating branches in the code tree. This re-integration ensured the code would from then on form part of the standard code base and the fix would be available in subsequent builds, thus maintaining overall code quality and control. Once a patch had completed its progress through the local practice and was verified by the test team as resolving a specific blocking defect and testing was again progressing in that area, the developer re-integrated with the standard process by turning the ‘patch’ into a ‘fix’ in code repository. In the next build that Testing had of which this fix was part, the defect as detailed in the DTR, would be tested yet again. If it is confirmed as being resolved then only at that point would the defect entry in the blockers’ document be tagged ‘white’ in and removed from daily version of the table.

Thus the standard practice of checking code fixes into the code repository for inclusion in the next build was temporarily bypassed before being resumed once the fix had been verified as working without affecting negatively other parts of the products. A fix to one area that caused significant problems in other areas would result in having to back those code changes out of the build and the ‘loss’ of the build which contained the change and they must wait for the next build to be produced before resuming work in the affected areas. If the patch does not work, it can be removed from the installed build in a matter of minutes and work can resume as before.

This highlights the importance of allowing for work practices and associated technological supports that are not entirely prescriptive in nature. Doing so allows flexibility in adoption and use, as well as flexible management procedures and protocols that allow the local team to determine that a local solution can be introduced that, while from a global view of the project might be seen as introducing unacceptable local revisions. This is done without disturbing the overall project management process.
5.3. **Maintaining practice through visibility**

Our data presented in the previous chapter illustrates in detail how the practice of daily meetings meant a local work-around, such as the practices of ‘patching’, can be found and allowed to proceed within the existing situation. In time that such a work-around then becomes part of the wider family or ecology of established practices of the workplace. Via the visibility afforded by the practice of a daily meeting, team members construct and maintain a locally applicable picture of the overall state of the work for use as the basis for ad-hoc yet consistent decision making. In this way contingences are dealt with in a more predictable and reliable way allowing for work to make ongoing progress even when faced with considerable unexpected impediments.

Work Practices that are unhelpful can undermine efforts towards the success delivery of the software product while a practice that is readily understood, easily transferred to new entrants and can be maintained with minimal effort can be beneficial influence a successful outcome.

However work practices can, as Grinter (1995) noted when referring to mechanisms of interaction, constitute a model of “how software development work should be done.” This model allows developers to make assumptions about their work environment and naturally act to constrain their actions because “the procedures support certain ways of working” implicitly while not supporting certain other ways of working.

In order that a practice can be successfully maintained through the course of a project it must be acceptable by those involved and seen as beneficial to them. As (Lethbridge, Singer et al. 2003) suggest “rather than forcing SEs to perform cost ineffective work, we should strive for simple yet powerful documentation formats and tools”. As we have seen practices in real work are intricately meshed with one another. No single practice exists in isolation; it is wrapped around and within other practices. Change one and the others will shift too. What we have seen is heterogeneity, a mixing and matching of a family of practices, both socially based and technological. Remove or weaken one strand from the weave and the entire tapestry of work practices at this location may unravel. As (Lethbridge, Singer et al. 2003)

Similarly light weight supporting artefacts such as the simple blockers’ document can be used to make relevant, timely and consistent decisions in dealing with the delay due to the
distributed build process. In this way benefits in the operation of and emergence of local practices in response to distribution were be realised by the team.

5.3.1 Propagating practice

During the course of our observation we witnessed an illustrative exchange amongst the project team at a blockers’ meeting which demonstrated the requirement in the course of what (Star and Griesemer 1989) termed “disciplining information” to have a consistent agreed view on the definition of what constituted a blocking defect. Lethbridge, Singer et al (2003) realised that “Propagating methods is not an easy task.” By making the methods more visible through the patching rather than behind closed doors it is more acceptable to those affected.

At one of the daily blockers meeting, the PLE project manager was asked by a member of the performance team “Just what is a blocker?” and the PLE project manager stated that a blocker was any defect ‘that prevents you from actually doing testing in an area’. This query had resulted from a problem whereby progress was occurring but much too slowly for the performance team to complete their work. From the performance team’s perspective, though they were able to work, it was at a pace too slow to complete their task that in their view was a blocker. While it did not get designated as a blocker, in this instance it was dealt with expeditiously. The developer whose areas it was agreed to look closely at the problem as a matter of urgency even though it did not get the designation of being a blocker.

The reasoning being that the performance team had readily to hand log files that could assist in pinpointing the problem and that to ignore it could lose the performance team’s agreement to the practices of patching, if their needs were not met. The practice of patching and daily meetings did not adhere strictly to the norms of software development. However, it was existing tools, the DTRs, and practices, such as daily meeting, which are already common to their workplace that were appropriated in such a way as to maintain the essence of the practices while in a very practical sense ensuring the ongoing flow of work in trying circumstances. The defect tracking reports easily lent themselves to this back and forth of designation and re-designation.
From this illustration of the pathway from a defect to a fix in Figure 21 the designation of a defect as a blocker opened up courses of action that were potentially beneficial to all concerned but also carried with them risks in undermining the wider practice if used to excess. Alternately, by denying a defect the status of being a blocker it served to close off actions too. If it was not a blocker then it would not be patched. And for those who changed the builds they were using much less frequently that was a major problem as their iteration gap was considerably greater.

The intended functioning of the computing resources is that one should always input software code to source control, compile a build and then release it to Testing. Yet this does not fit with the real world expectations when the turnaround time required to do all that and to install the new software build is too great.

Gasser (Gasser 1986) notes that the fixing, augmentation and fitting come into play when seeking to work around problems thrown up when the computer resources do not fit with the work practice.

5.3.2 The conservation of defects

Defects are being uncovered rather than created; the test team are bringing into the light something that already exists in the code. The defect reports in turn are accounts of the uncovering of the defects and the history of how they come to be resolved. The people who require the information contained in these accounts need it so that they can reproduce or make the defect visible once again. The audience for this includes the developer or developers who will work on fixing the defect, but also members of the test team who will need to confirm that the fix proposed by the developer and the tester themselves will need to recreate the
circumstances again at a later point. The defect reports can pass through many hands before being resolved. In Figure 22, over the course of 5 months 9 people made changes to the state information for this defect. Due to confidential reasons we have obfuscated their names.

Figure 22: The Comment section of a DTR showing the time line for state changes

This collection of defect tracking reports remains up to date in contrast to other internal documentation and maintains a higher degree of accuracy and relevance to the day to day work. This supports the findings of (Lethbridge, Singer et al. 2003) who noted “SEs (Software Engineers) were likely to update documentation continually only when it was attached to the process for completing a change request. In other contexts, SEs sometimes would and sometimes wouldn’t update documentation. The documentation’s accuracy seemed to depend largely on its recentness and the amount of change that had occurred in the relevant code sections (with greater source code change corresponding to greater discrepancy between the code and the documentation).”
– Discussion

As a form of organisational memory the fact that the content of the memory is created as part and parcel of the work as it is being performed rather than involving some extra steps or being required after the fact makes it more faithful to what has actually occurred rather than what is later remembered as having occurred. The notes and comments that developers create in some DTR for their own use are also a part of this pattern of writing as they go. The prompts for these details and comments are made at the opportune times, documents.” (Lethbridge, Singer et al. 2003) “adding a simple comment as you fix a bug requires little effort, and maintenance is semiautomatic” when the detail is fresh and more easily updated “The relationship between accuracy and consultation frequency is the highest for testing and quality documents and the second highest for low-level design documents.” (Lethbridge, Singer et al. 2003)

5.3.4 Making invisible work visible.

In order that the practice of patching installed builds might be maintained, it was necessary that the actions associated with it take place in as visible a way as possible. The potential ability of a developer or the Deployment engineer to be replacing files unknown to the Testing team would entirely undermine the testing process as the Testing team would never be sure what it was they were in fact testing. In such a scenario they could not be sure what version of software they had tested, what remained to be tested and what tests had been run on which versions. So this capability to make alterations to what Testing are working on had to be managed in as transparent a manner as possible while not creating an excessive additional communication burden.

The work practices were given increased visibility in part by the separate need to appraise both non-core team members and new members of the practice as it exists: showing that ‘this is the way things are done here’. The invisible or less visible actual work of the patching being done by Testing and Development is made more visible and traceable for those immediately outside those doing it i.e. Deployment and the developer who creates the patch, by means of the daily meeting, the circulation of the meeting minutes and the presence of the patches in the DTR. The daily meeting, where the decision to proceed or not with a patch is made, and the DTR, which serves as the place for the patch to be stored when being conveyed to the Deployment engineer, means the work is made visible is part and parcel of the practice. It does not require an additional overhead. While perhaps not a complete means for creating shared understanding between distributed team members, the daily blockers’ meeting and DTR do afford an opportunity to begin to create a shared understanding. As explored by
(Grinter 1995), the work practice of documenting defects is most commonly in use by software professionals and reflects with great fidelity the way in that their work is performed.

At the daily meeting, the link between the forms of increasing the visibility is in part due to the updating of the blockers’ document by the chair as the meeting progresses. There exists a fear expressed in the corporate environment that as communications are made easier the likelihood increases that it will be used for frivolous non-work purposes. While Instant Messaging technology, or ‘chat’ as it is commonly termed, has been feared by corporate organisations as a potential facility for time wasting it has been shown in at least one study that the overwhelming majority of chat content was work related (Handel and Herbsleb 2002). A caveat in that study is that the workers themselves conducted the assessments of their own content as to which they deemed to be work related ‘chat’ and which they considered was personal.

In the course of the daily blockers meeting we observed Instant Messaging being used very specifically to allow for side communication with people who were not present but who possessed information necessary to the progress of the meeting. This was done in a manner which was non-disruptive to the rest of meeting and expedited the resolution within the meeting timeframe. On one occasion we observed it was used to get information from someone who was not attending the meeting as it hadn’t been their turn to attend but they were the one who knew what the current status was. This allowed the meeting to continue with the information being communicated once it was received.

5.3.5 The importance of order

The order of the DTRs in the blockers’ document table provides a context and ready history in the course of the meeting with new DTRs being added to the end of the table under the relevant section. So the first defects discussed at each meeting are also the oldest. On occasion there may be requests or attempts by attendees to guide or force the meeting so as to jump ahead in the table to the defect they were most concerned with but they are rebuffed by the chair. The meeting chair knows that sticking to the table and preserving the order helps those not playing 100% attention to the defects outside their area to know when their particular line is up for discussion. That is not to say that they were completely ignoring all discussions other than for their own defects. Indeed, it brings the attention of the workers away from their own individual reports for a while and keeps them aware what the overall state of the project work is compared to previous meetings.
Further more localised visibility at the meetings for individual defects was necessary as the team members tended to only have eyes for those parts of the blockers’ document or DTR that affected them and their work. The key mechanism for talking about a DTR, its entire ID and title, are not conducive to identifying it during a meeting, the title tend to be too similar and generic and the IDs are too long and complex. In large part people talk about ‘that one’, or ‘the one John was working on’, to provide a less onerous context in the flow of their discussions. The project manager when chairing the meeting would use only the last 4 characters of the ID in identifying it.

5.4. Circumventing the iteration gap and ‘winning’ the code race

This suite of local practices as we presented in the preceding chapter consisting of: a “patching” process coupled with a daily meeting had emerged to address the iteration gap arising from the centralised build process. The daily meeting was similar to that a daily scrum found in the agile development model. The alternative to daily centralised builds is local ad-hoc builds created as required but such Ad-hoc builds lead to problems with maintaining consistency. This temporary work around or side shunt allows progress to be made more quickly and with no impact on the final quality of the product as the patched code is later reintegrated into the source control for final verification.

Let us compare the Iteration Gap with the consequences and opportunities that result from Patching; we start at a point where the developer has some code that she believes will fix the defect. In the Iteration Gap timeline this pans out as shown in the left-hand column of table 6 and with the patching process the time line for the iteration gap pans out on right hand column.
<table>
<thead>
<tr>
<th><strong>Iteration Gap timeline</strong></th>
<th><strong>Patching process</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Day One</strong></td>
<td><strong>Day One</strong></td>
</tr>
<tr>
<td>□ Code is available but needs to be locally checked in before 3pm (local time) in order to make that day’s pre-build</td>
<td>□ Code is available to be deployed locally, contact is made with Deployment as to what the changed files are</td>
</tr>
<tr>
<td>□ A pre-build is performed locally and completed by 5pm*. This is to allow problems that arise in the compilation and build to be corrected or backed out before the next step of submitting all these changes to the US Source control</td>
<td>□ The server is stopped and the files copied over to it by the Deployment engineer. The server is restarted</td>
</tr>
<tr>
<td>□ The code passes the pre-build process</td>
<td>□ The defect report is updated with the altered files notifying testing that the defect is ready for verification</td>
</tr>
<tr>
<td>□ The code is checked into the US source control (Clearcase)</td>
<td>□ Testing run their test and verify if the patch works or not.</td>
</tr>
<tr>
<td>□ Central Build commences on the US East Coast at midnight (US time) - 5am (local time)</td>
<td>□ The DTR is updated as to whether or not it has worked.</td>
</tr>
<tr>
<td><strong>Day Two</strong></td>
<td><strong>Day Two</strong></td>
</tr>
<tr>
<td>□ Code is available but needs to be locally checked in before 3pm (local time) in order to make that day’s pre-build</td>
<td>□ If the patch has worked then the Iteration Gap as on the left commences from this point on. However in the mean time Testing can continue on the now unblocked area.</td>
</tr>
<tr>
<td>□ A pre-build is performed locally and completed by 5pm*. This is to allow problems that arise in the compilation and build to be corrected or backed out before the next step of submitting all these changes to the US Source control</td>
<td>........</td>
</tr>
<tr>
<td>□ The code passes the pre-build process</td>
<td>□ The code is checked into the US source control (Clearcase)</td>
</tr>
<tr>
<td>□ The code is checked into the US source control (Clearcase)</td>
<td>□ Central Build commences on the US East Coast at midnight (US time) - 5am (local time)</td>
</tr>
</tbody>
</table>

*‘local time’ refers to the time at the field site in Ireland.
- Discussion

<table>
<thead>
<tr>
<th>Day Three</th>
<th>Day Three</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build is posted on a US based server at 4am (US time) - 9am (local time)</td>
<td>Build is posted on a US based server at 4am (US time) - 9am (local time)</td>
</tr>
<tr>
<td>Build is pulled down to Dublin 5am (US time) - 10am (local time)</td>
<td>Build is pulled down to Dublin 5am (US time) - 10am (local time)</td>
</tr>
<tr>
<td>Build deployment is completed by 4pm (US time) - 9pm (local time)</td>
<td>Build deployment is completed by 4pm (US time) - 9pm (local time)</td>
</tr>
<tr>
<td>Build is subject to BVT and smoketests which take roughly half a day</td>
<td>Build is subject to BVT and smoketests which take roughly half a day</td>
</tr>
<tr>
<td>End point for build being ready for use by Testing lunch time. This is at lunch time almost 48 hours after the code change was made and checked into source control by the developer.</td>
<td>End point for build being ready for use by Testing lunch time. This is almost 48 hours after the code change was made and checked into source control by the developer.</td>
</tr>
<tr>
<td>Defect can start to be regressed.</td>
<td>Defect can start to be regressed by the testing team.</td>
</tr>
<tr>
<td>Fix is found not to work. Developer starts working on a new fix.</td>
<td>Fix is found not to work. Developer starts working on a new fix.</td>
</tr>
<tr>
<td>Developer finds the problem and makes new code change. This is likely to be after the 3pm deadline</td>
<td></td>
</tr>
<tr>
<td>Fix is found not to work. Developer starts working on a new fix.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Day Four</th>
<th>Day Four</th>
</tr>
</thead>
<tbody>
<tr>
<td>(restart of activities as from Day one)</td>
<td></td>
</tr>
<tr>
<td>Code is available but needs to be locally checked in before 3pm (local time(^{10})) in order to make that day’s pre-build</td>
<td></td>
</tr>
<tr>
<td>A pre-build is performed locally and completed by 5pm*. This is to allow problems that arise in the compilation and build to be corrected or backed out before the next step of submitting all these changes to the US Source control</td>
<td></td>
</tr>
<tr>
<td>The code passes the pre-build process</td>
<td></td>
</tr>
<tr>
<td>The code is checked into the US source control (Clearcase)</td>
<td></td>
</tr>
</tbody>
</table>

\(^{10}\) ‘local time’ refers to the time at the field site in Ireland.
Central Build commences on the US East Coast at midnight (US time) - 5am (local time)

So this gap of 3 days or so is repeated at the iteration; so that say 3 attempts will cost a minimum of 9 working days or thereabouts even if each of the 3 changes is found by the developer in a matter of minutes. Meanwhile, 3 attempts in the patching process will cost 3 days plus the time for the developer to find the 3 changes. In other words,

\[
\text{Iteration Gap Time Lag} = \text{Time to develop code changes} + (\text{No. Of attempts} \times 3 \text{ days})
\]

\[
\text{Patching Process Time Lag} = \text{Time to develop code changes} + 3 \text{ days} + \text{time to verify the patch}
\]

In a considerable portion of cases the first attempt at providing a patch fails to resolve the blocking defect. This failure of the patch to resolve the defect was not a remarkable event. Everyone in the team was aware and experienced in how unpredictable defect resolution efforts can be. The defect history in Figure 3 illustrates various attempts that had been made at solving one particular blocker ‘SJTY’. In the right hand column the notes on each attempt are recorded, with cycles of investigation, patching, checking and investigation again. This covers considerable work that goes into investigating the defect.

The example in Figure 3, in chapter 4, illustrates the importance of being able to locally bypass and work around the standard practice as the length of the ‘iteration gap’, of three to four work days per each attempted fix, would have presented an insurmountable obstacle for this project. As it was even with the local patching process testing was blocked on this defect for over a month before testing could be resumed and work completed on resolving the defect.

In this instance the introduction of the patching practice means that the iteration stage is reduced considerably where multiple attempts to correct the problem are made. While a standing charge of three days exists in both cases, in the patching case Testing can proceed immediately from the point the fix is found and has been applied. The gap is further compounded with iteration in the builds.
5.4.1 Cycles and rhythms

Each person must work on what they had worked on a few days ago, what they are working on today and also plan and prepare for what they will work on tomorrow. Often this means taking simultaneously steps forward and backwards and it is not possible to wait for certainty.

They use the rhythms and routine cycles of the work day, work week and the various cycles of the types of builds to find relative stability amongst the shifting ground that they work on. Each team member seeks to find their sea legs by means of the routine. The use of the defect reports in the blockers meeting serves as a contingent means to resynchronise the daily work cycle. In the meetings the build number serves as another means to subtly focus attention, the name of the build being derived from the date on which the build was made available.

They work with the cycles of the build process rather than pushing against it, making use of it to afford them the time need to reflect and plan iterative development. Each daily build contains code up to the check in deadline of 3pm the previous day and is rarely available to testing work on that same day as it is downloaded. The local Build Engineer plays a key role in facilitating the patching practice though the timing of the local pre-build. The local deadline of 3pm GMT when the source control gets closed to ensure what is checked in does not break the main build. After this 3pm deadline a local pre-build is done.

If there are problems flagged at this point then the expectation is that the person whose area the problem is found in will be available to investigate. They can then either correct the problem by making changes to the code files or 'back out' (remove from submission to the source control) their changes. This is all done prior to the code being committed to the central source control which is used for the nightly main build which is happening at overnight US Eastern Time. This mechanism is designed to minimise the impact on the main US build of a problem occurring when local contributors to it are offline and unable to respond due to being asleep. They are using the predictability of the build cycle to manage the uncertainty that comes as a natural result of making changes to the code.
5.4.2 Taking uncertain steps into certainty

The practices of ‘patching’ was a creative workaround which allowed the team to overcome the iteration gap in building the software product and its effect of creating a delay in fixes being applied to an ‘official’ build being available to the Testing team. A degree of certainty in the face of this unpredictability is attained by repeated synchronising and common reference to the information contained in the Defect Tracking Reports and the Blockers’ Document in the familiar and settled surroundings of the daily blockers meeting.

Patching is a more controlled and limited response to the risk generally attached to making significant changes to the code. Moreover it is a more reassuring course of action when one is very conscious of the fallout from breaking the build. This is moreover the case as the build is being produced for consumption by more than your own group and where you are not confident of the wider impact of the code change being made. It is far easier to back out a patch that has been added to a specific test server than to do the same with a code fix from a build that is common to all. A code fix that results in problems will cause previously working areas of a build to stop working or to even break the build meaning the build process does not run to completion and thus no build at all is produced for that day.

A new build brings with it potential for new problems in addition to these existing problems which had been identified to date. Rather than delaying the test process and introducing a newer build which contained further unknowns the decision was to patch the existing one about which much more is known. As they work with each build they learn about its limitations and that which is functional takes on the form of being the familiar steady state of affairs.

The uncertainty of the consequences of acting is an ever-present concern. Throughout the course of our study many members of the team spoke of the contingent nature of their work that making progress on defect X was dependent on a successful outcome for defect Y. Yet even a successful outcome to defect X was not certain to deliver a resolution for defect Y. They were merely the basic conditions necessary for progress to be made and were not sufficient in themselves to guarantee success.

The progression path from patch to fix is well established within the group but the actual progress path that each defect will take is not always certain. The steps are logical and well understood within the project team. However, it did not appear as a matter of astonishment to
them if a problem reasserts itself after the code changes that had resolved the issue when used as a patch have been checked into the code stream. They are naturally disappointed and curious as to why this happened but not surprised. There is recognition that other elements have changed in the new build and that what worked with the previous build that had been patched may not always work in the newer build.

As they try to find resolutions to defects there is a tension between staying with what they know is stable in some areas but blocked in others, and moving to a new build which might be unblocked but in which new blockages might exist. Towards the end of the project as they concentrate more on fixing DTRs and less on create new code to implement new features, they tend to move to new builds with greater urgency and frequency. In part this can be the result of the build fixing a problem. When a DTR is raised the developer tries to reproduce it on their local build which is often a newer archive build and if the defect cannot be reproduced then it may have been fixed inadvertently.

Indeed it may have been caused inadvertently too in the specific build that the problem was uncovered in. For developers the process in reverting back to previous builds that they have moved on from is not a trivial task. It involves removing all installed traces of the previous build, and of accessing a code base that may be a number of weeks old. Being concurrent in how things were implemented at that point and implementing changes that might be dependent on components that have subsequently been altered. In truth they would be creating obsolete or dead-end patches that may not be necessary and may not work in the current build. Hence stepping backwards is resisted.

The patching practice also places centre stage work that would otherwise take place off stage. Yet while it aids in forming a more complete picture the content remains only a tantalisingly partial story. Whatever source is used to assist in completing the picture it is important to remember that the story is always going to be incomplete. When it comes to a decision between completeness or fidelity our findings support the contention of (Lethbridge, Singer et al. 2003) that the defect tracking system has fidelity to the current real picture even though it may be incomplete. All such observations are incomplete.
5.1. **Summary**

We have discussed phenomena that show the complexity, scale and reality of software development in a global environment. We have shown how they deal with a gap in iteration and how the maintenance of work practices is aided by making them as visible as possible. The work rhythm of each day, the process of gearing into it, of getting updated on what has gone before and which takes up some hours.

We have utilized the extensive dataset provided by the content of the defect tracking systems of this software project team as a means to illuminate much of the activity that takes place off stage. The defect tracking system has been co-opted to function as a means to distribute and archive the work of patching.
Chapter 6 - Conclusions

6.1. Introduction

This research work has involved bringing forth a story of how software professionals succeed in working around blockages in their work that resulted from a core part of their work process being distributed overseas. This distribution which affected their test and defect resolution cycle was supported by work practices. The field study we conducted illustrated how the work practices were mutually maintained and the part played by the heterogeneity of the practices and technology at the field site. The lightweight modifications in the use of their existing defect tracking system were accompanied by a series of daily meetings supporting frequent contact between the divisions of work. However, it also demonstrates the highly problematic limitations that could prevent the transplantation or re-application of portions of these individual practices in another setting.

The first section in this final chapter brings together the core contributions of this research reiterating what we have outlined already through the course of this thesis and showing how these contributions may be built upon while acknowledging some of the limitations. In the second section, prospective avenues for future work based on this research are suggested.

6.2. Core Contributions

This research throughout the progress of this thesis has uncovered a number of the practices of software professionals who are working with the daily consequences of distributed environments. This has been done through means of an account of their day to day routine coupled with a number of examples of the documentary artefacts. This is a work routine rooted in the mundane and everyday which they see as unremarkable. They find themselves managing complex and shifting problems which have arisen as a consequence of the distribution of portions of their work and the highly contingent and interdependent nature of their practices.
By peeled back these practices to their bare bones, we expose how the work practice and electronic artefacts have been extended to allow work to continue to make progress. We also show how these daily activities are maintained through the course of the development of software by increasing the visibility of one practice that of patching of builds by means of another practice, having daily meetings and by attaching the patched files themselves in the defect reports. However, it also identified how the practices can be limited in their separate application in circumstances different to those that pertained at our field site. They cannot readily be plugged out or simply copied wholesale to be reinserted or reapplied elsewhere.

6.3. Emergence and Maintenance of practices and technology

In this thesis we have shown that local practices emerge, quite readily, in response to the impact of global distribution of even very discrete functions. This is not to state that all local practices emerge this way rather that they can do. What we have shown is primarily the question of “how” this happens in the course of this work and in doing so to acknowledges that the broader questions of “why” this is necessary merits more immersive and in depth investigation. The “how” centres on the manner in which these practices are then maintained, shaped, reshaped in response to changes in the environment that they originated from. It also includes “how” those practices are supported and or alternately hindered by the technology and the readiness of those working with it to adapt in order to overcome the consequences of the distribution.

6.3.1 Application of this practice to other settings

We have seen that no single practice exists in isolation rather that they are inextricability linked with and influenced by other practices. There is a broad heterogeneity of practices and technology in place and this is reflective of an increasing trend in software development. As we have seen practices in real work are intricately meshed with one another. Remove or weaken one strand from the weave and the entire tapestry of work practices may unravel.

It is entirely possible, though we believe unlikely, that only in this particular set of circumstances could these practices succeed, for succeed they do. Ideally in the view of many, a practice, in order to be useful to others, should be open to translation or readily transferable to other environments. The potential isolation of particular individual strands of the practice
from its environment and its reapplication elsewhere is perhaps limited. This can affect the transferability of individual practices. The visibility that allows the practice to emerge and be accepted is reproducible.

Yet once the motivation for the practice is understood, the process by which they came to the practice and what was necessary for it to be maintained, this can have as much value as the specifics of the particular practice itself. Knowing how they arrived at their practice and what work is needed to maintain it is transferable knowledge. However our contention is that the mere fact of gaining an understanding into how a work practice works, whether in this setting or another, it is possible even if the exact set of circumstances do not exist elsewhere to learn from “how” that practice was developed and maintained and applying the lessons of that “how”.

6.3.2 Ensuring acceptance of emergent work practices

Even quite constraining and complex practices work well when they are seen as necessary to both short term goals of the individual team members as well as the overall team goals rather than simply being something that is nice to have or which might lead to some benefit down the road and when seen and understood to be necessary by all those involved. There is a clear and well versed vested interest for the Testing team in having builds that they can test properly and to as close to exhaustion as possible, there is a vested interest for the developers in knowing that what work they have done to date is correct and there is a vested interest in the management in ensuring that the schedule is adhere to and that things occur in the order in which they are supposed to. Ensuring continued agreement or ‘buy-in’ to a continuing practice and maintaining it by making repeatedly plain the need for that buy-in through the visibility afford by the more visible practice.

6.3.3 Visibility from meetings

The frequency of the meetings though they were short in duration stands out as a key factor in the maintenance of the visibility of the practices of patching where meetings are concerned. Frequent and short works better than infrequent but long when it comes to such meetings which are to monitor the progress of work of this nature and in making people aware of any adjustments. Longer meetings increase the overhead in managing what is rapidly shifting circumstances. The Blocker’s Document which forms the core of the meeting shows that simple actions can readily assist in tracking status changes. It is not a black and white world, and colour can and does play a part in readily highlighting status changes.
6.4. **Limitations of the study**

It should be noted that these conclusions are based on one team working at one site who were involved in one software project. We were not aware whether other teams at the site were using the patching work practices at the time, but those teams would have been aware of them as project managers met regularly to exchange experiences. There are teams working on other projects at the site that were faced with different problems. It is possibly that this practice would not find a suitably receptive audience for many reasons in other teams.

While the rigidity and comparability of numbers will long continue to be a key feature of software and process improvement generally, it is also the case that no two projects are ever alike and not two sets of numbers are directly comparable. Ignoring the role played at the code face of the mundane details on how practices are made malleable, digestible and reworked so that progress continues to be made from day to day, from build to build from integration point to integration point, is to ignore the most practical data source of all. If patching were used on its own or for smaller scale projects it could prove to be more of a hindrance than assistance. When the iteration gap is too small, and the size of the patch too large compared with the overall product, the changes would require too much retesting and the time saved would be insufficient to justify that amount of retesting.

Such an environment rich in hard data would benefit from the application of a more quantitative methodology. Quantitative work could also serve to explore of the evolving state of a project from start to finish. Alternatively an ideal would be for further research work to involve making a comparison with another team with another organisation or the same one, faced with a similar iteration and blockers problem but which had not adopted practices of patching in order inferences drawn from how their work progresses. Or a multisite study which looked at the impact of a rigid centralised build process on the local client sites who were unable to patch.

6.5. **Avenues of future research**

As with any work of this nature, many more questions are generated from the answers we have unearthed of how this work is progressed and further research is warranted in a number of areas. We suggest some possible avenues for future work to address some of those questions and build upon the work done.
This study reiterates and adds weight to a fresh impetus to the need for research into software development to move beyond the comfort zone for software engineering of primarily quantitative measureable data points.

As stated above and detailed in chapters four and five, our research found that the existence of the practices associated with patching act as a form of source control precursor. The code changes are made temporarily outside of the source control but are eventually added to the source control. This approach allows for the additional inclusion of the work of failed attempts by developers to be archived and tracked in the defect reports. Even if this capture and archiving was somewhat incomplete and imperfect, it aids the developers in not repeating steps they or others working on the same defect have already taken when they can potentially have lost familiarity with the particular details of the defect.

In terms of applications that could arise from the findings of this work a proposal might be made for an operating system that properly supported a source control method of replacing, archiving and commenting of the modifying of installed files might be a more secure means to give effect to this type of piece meal iteration and revisionist installation.

6.5.1 Industry interaction

There is a palpable interest from the software industry community in research of this type but the opportunities must be created to reflect this work back to them. During the course of the conduct of the field study it was remarked upon by the team members whether they would get to see the outcome. The active research in the academic community must seek out what opportunities exist for those in industry to access such work and to reflect, respond and where appropriate implement any findings that they feel are of use of them.

6.6. Summation

Finally, what truly represents the best outcome is that the software product realises the ambitions of those who design it, create it and use it. Mere adherence to a set of protocols does not make for a successful product. Strict deference to rules for their own sake makes by those who should be in control of their own work makes them being people ruled by rules instead of ruling through rules. Our admiration for the inventiveness of those is renewed; the code race is won by arriving all at the finish line in unison.
There was a young lady named Bright.
Whose speed was much faster, much faster than light.
   She departed one day
   In a relative way
And returned on the previous...
Bibliography


Gerace, T. and H. Cavusoglu (2005). The critical elements of patch management. 33rd annual ACM SIGUCCS conference on User services, Monterey, CA, USA, ACM New York, NY, USA


Halverson, A. C. A., A. J. B. Ellis, et al. (2006). Designing task visualizations to support the coordination of work in software development. 20th anniversary conference on Computer supported cooperative work Banff, Alberta, Canada


Herbsleb, J. D., A. Mockus, et al. (2000). Distance, Dependencies and Delay. 2000 ACM conference on Computer supported cooperative work


Lutters, W. G. and M. S. Ackerman (2002). *Achieving safety: a field study of boundary objects in aircraft technical support*. Computer supported cooperative work, New Orleans, Louisiana, USA.


