Situated Software Development: Work Practice and Infrastructure are Mutually Constitutive

Julia Prior
Faculty of IT, University of Technology, Sydney
julia@it.uts.edu.au

Toni Robertson
Faculty of IT, University of Technology, Sydney
toni@it.uts.edu.au

John Leaney
Faculty of IT, University of Technology, Sydney
johnl@it.uts.edu.au

Abstract

*Software developers' work is much more interesting and multifarious in practice than formal definitions of software development processes imply. Rational models of work are often representations of processes defined as they **should be** performed, rather than portrayals of what people **actually do** in practice. These models offer a simplified picture of the phenomena involved, and are frequently confused with how the work is carried out in reality, or they are advocated as the ideal way to accomplish the work. A longitudinal ethnographic study (45 days of fieldwork over 20 months) of a group of professional software developers revealed the importance of including their observed practice, and the "infrastructure" that supports and shapes this practice, in an authentic account of their work. Moreover, this research revealed that software development work practice and the infrastructure used to produce software are inextricably entwined and mutually constitutive over time.*

1. Introduction

> "Formal descriptions of work (e.g. "office procedures") and of learning (e.g. "subject matter") are abstracted from actual practice…We, by contrast, suggest that practice is central to understanding work. Abstractions detached from practice distort or obscure intricacies of that practice. Without a clear understanding of those intricacies, and the role they play, the practice itself cannot be well understood, engendered (through training) or enhanced (through innovation)." [15, p40]

An assumption that software development work can be adequately represented by formally defined development methodologies and their concomitant processes masks the intricacies and complexity of the work, and renders it rather dull. Developers' work as it is done in professional practice is far more elaborate than formal definitions of software development methodologies and processes suggest. A procedure to perform some particular task is defined and articulated linearly as if it were an isolated phenomenon. And because it can be described this way in the abstract, the definition is often held up either as a sufficient depiction of the practice performed to accomplish the work, or as the ideal way to do the work. These 'patterns for behaviour' (PFBs) only express expected behaviour or norms, and are inadequate as a representation of 'patterns of behaviour' (POBs) which is what people are observed to actually do. Ethnography was the approach used in this research as it is particularly suited to gaining an in-depth understanding of how work is accomplished in practice by a group of people, foregrounding apparently mundane factors that make an important contribution to the successful accomplishment of daily work.

Software developers are technically savvy users of technology who have an understanding of how that technology works and how to design and produce it, as well as knowing how to use it. In practice, software developers make use of an "infrastructure" that supports and enables their primary work of developing software products. "Infrastructure" does not refer to lower-level software, such as an operating system or middleware, on top of which application software can be executed. And the common notion of infrastructure, in which it is viewed simply as a substrate, a separate entity on which some other thing 'runs' or 'operates', is rather narrow. The term has a particular meaning in this study, which will be further elaborated in section four. Briefly, the infrastructure in this context includes specific procedures which the developers are expected to follow to perform various aspects of the development work, i.e. PFBs, tools such as code editors, compiling and testing applications, and standards (see Figure 1). The participant developers had an excellent understanding of their infrastructure

and its role in their daily work, and in fact, constructed much of it themselves over time.

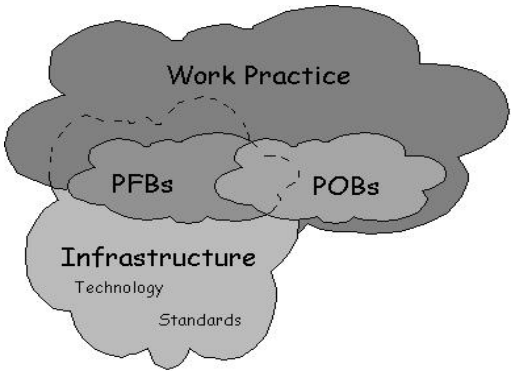

Figure 1. The Relationship between Infrastructure and Work Practice

Figure 1 is a representation of how infrastructure and work practice relate to one another. Work practice, i.e. what people do to accomplish their work, is comprised of both 'patterns of behaviour' (POBs) and 'patterns for behaviour' (PFBs). PFBs are also an important part of the infrastructure which enables and supports work practice. Work practice and infrastructure need to be examined together as inextricably entwined phenomena. This diagram is intended merely to aid comprehension of these issues and how they are related to each other as discussed in the rest of the paper; it should not be deemed a prescriptive theoretical model.

Although work practice and infrastructure are initially discussed as discrete concerns in this paper, this is not how they should be interpreted as occurring *in situ*. Software development work as experienced daily by the developers is an ongoing, organic interplay, a *bricolage* of infrastructure (technology and procedures) and work practice. The crux of this research is that these phenomena should not be regarded as distinct issues: the infrastructure created, used and maintained in the participant company is inseparable from its developers' work practices. The local infrastructure and work practice are mutually constitutive: they become more and more entwined over time, shaping and impacting each other in a symbiotic relationship. To attempt to understand them in isolation from one another results in a limited, if not inaccurate, perspective of how software is produced in professional practice.

The next section of the paper explains the how, why and where of the research undertaken: a description of the research method, the research rationale and the field site, and two vignettes from the fieldwork data are recounted. The concepts of work practice and infrastructure are explored in sections three and four respectively. These set the stage for section five which focuses on the local realisation of both phenomena in the participant company, with particular attention to the 'TestFirst' approach, advocated as the developers' core practice. Section six is a discussion of the main theme of the paper i.e. infrastructure and work practice are intertwined and mutually constitutive over time. Concluding remarks in section seven summarise the main points of the paper and suggest some implications of its findings.

2. Background

Schön's [14] view is that the knowledge, skills and practices of practitioners should be explained by their demonstrated abilities and competencies, not limited to academic theories and models. This is echoed by Seely Brown and Duguid [15] in their concern that 'canonical practice' (work as described in organisational manuals, training courses and job descriptions) is not an authentic portrayal of how work is accomplished in practice.

In the Human-Computer Interaction and Computer-Supported Co-operative Work research areas, there is a substantial literature on work practice research, including ethnographies, which is used as the basis for designing software applications that faithfully support the users' work practices, for instance Jordan [9], Suchman [22] [23] [24], and that summarised by Harper [8]. Ethnography is a research approach based on fieldwork, i.e. spending a considerable time observing and participating in the daily lives of a particular group of people, which endeavours to understand local beliefs, attitudes, values and behaviour.

Ethnography is also not foreign to software engineering and system development, particularly in requirements engineering and systems design. More than a decade ago, Bentley et al [3] did an ethnographic study of air traffic controllers, which became a seminal work in the software systems design research literature. Sommerville (Ibid) has published extensively since on the use of ethnographic research of user work practices to inform the requirements engineering and systems design process, for example [17] [18] [25].

The aim of research into software systems design and development is more often than not the definition of theoretical models, and empirical research done on

an experimental basis to prove that these theories are effectual and useful.

> "There is a severe decoupling between research in the computing field and the state of the practice of the field. That is particularly problematic in the SE [software engineering] field." [6, p505]

This was one of the conclusions that Glass, Vessey et al made in their comprehensive review of software engineering research literature in leading research journals. Little has been done on studying the work practices of professional software developers in a manner similar to that in which users and their work practices are examined as part of the software design and development process. Notable exceptions are Sharp and Robinson's ethnographic work on XP developers [13] and [16], and Sommerville et al's ethnographic study of software testing [19]. But the dearth of research into software development practice remains.

2.1. Research Method

> "There is a recognised need for empirical studies of software engineering, including ethnographic studies. One of the strengths of ethnography is the ability to take a broad focus on work rather than on a particular method or technology." (Ibid, p603)

The interpretivist paradigm provides the underlying theoretical perspective for this research. Ethnographic research attempts to understand the situation from the perspective of the group members. Comprehension of individuals' behaviour is not the point of ethnographic search; the focus is a set of describable patterns which occur over time, that are repeated by most of the group members, most of the time. Together, these patterns constitute the culture of the group and, according to LeCompte and Schensul [10], it is the emphasis on culture that sets ethnography apart as a research method.

An ethnographic study is done *in situ* and is not intended to be replicable or generalisable. The results are not based on large, representative samples, but on an in-depth understanding of a specific situation drawn from 'up-close and personal', prolonged observations. The data collected is rich, but messy, resisting formalism. It is not objective research. The aim is sense-making, rather than seeking generalised features to provide control and predictability. Significantly, no hypothesis is offered at the beginning of the research that needs to be proven. No theory or model is implemented experimentally with the intention of collecting evidence to demonstrate its veracity. In fact, the research is open-ended and the results are emergent and unanticipated.

Ethnography recognises that we must first discover what people do, and the reasons they give for doing it, before we can assign to their actions interpretations drawn from our own personal experience or our academic disciplines. The spotlight is on practices (what people are observed to actually do) versus norms (what people are expected to do). Ethnography highlights the everyday, apparently mundane, and taken-for-granted aspects of practice, as well as being able to account for those things that change or shift over time, and those that stay the same.

The aim of this study was to gain an understanding of what professional software developers do as 'situated action', i.e. what they are observed to do in their everyday work environments. Over a period of 20 months, the first author did ethnographic research in an Australian software development company. The fieldwork consisted of 45 site visits, each lasting between three and eight hours: the developers' everyday work practices in their normal work environment were observed and comprehensive fieldnotes recorded, company documents, policies and resources such as email were investigated, meetings attended and conversations held with the developers.

As mentioned, with ethnography there is not a hypothesis that motivates the research. Instead, an open-ended, guiding question initiates the fieldwork, and the research questions gradually become more refined as the fieldwork proceeds. Data gathered is progressively analysed and reflected on throughout the fieldwork period, and the interim results inform further fieldwork. Patterns start to come out of this iterative data gathering, data analysis and reflective process. Thus, the themes in this research are strongly grounded in the rich fieldwork data, and were only established after intense, reflective analysis and consideration of this data and the patterns that emerged from it. For example, the terms 'work practice', 'infrastructure' and 'mutually constitutive' were not employed until the themes had become clearly visible and identifiable, and needed to be named in order to usefully discuss them. They were not concepts imposed on the data in order to analyse or comprehend it.

2.2. Field Site Description

The participant company, Raptor Systems, develops software products for use in the freight forwarding industry. The company does not develop customised software for individual clients, but rather develops

software products that support the rules and regulations of the freight forwarding industry, and clients in this industry purchase these products to support their own operations. In addition to being software developers, the CEO (who is also a hands-on developer) and senior developers and project managers have significant experience in this industry i.e. have experience as users of the type of software that they produce. This is invaluable in terms of understanding the requirements of their users and producing software that supports the extremely complex laws, rules and regulations in the industry, which change on a regular basis. At the time of the fieldwork, the flagship product was a large, complex software suite called 'Connect'.

There were several developer teams. Each focused on one module of 'Connect' e.g. Freight or Accounts, and consisted of a mix of senior and junior developers, including a team leader. There were between 50 and 70 developers who all worked in one big open-plan office, at similar workstations grouped by team. This arrangement reflected the relatively flat organisational hierarchy; one would not have known who were the senior developers or management, or even the CEO, by looking at the physical environment.

The development approach used was strongly Agile [1]. In essence, this means that the following are particularly valued: people and their interactions and collaborations, working software released frequently, and responding actively to change. These principles are the dominant forces for development, rather than processes and tools, comprehensive documentation and plans, and contract negotiation [4]. One of the characteristics of Agile development is that design is considered to be an integral part of the development process, not a discrete phase early on; although requirements are progressively documented, there is very little in the way of formal design diagrams or separate documentation of development decisions and process, as in traditional software development. Agile developers talk about the design being 'in the code' and the code (and consequently the software product) is designed and built incrementally. Thus, program code is the major artefact and the focus of the development effort is producing working program code. The high level design at Raptor was managed by the 'A-Team', developers responsible for the overall architecture of 'Connect'. The software development infrastructure and program code at Raptor is described more completely in an earlier paper [12] which focused on infrastructure and program (software product) code. In this paper, this infrastructure is discussed in terms of its relationship with the software developers' local work practice.

2.3. Two Development Vignettes

Two vignettes (brief accounts) of the participant developers' work derived from the field study are provided here. Vignettes are used to make the insights of an ethnographic study available to readers by describing activities within their contexts in a succinct way. Wenger [26] makes use of this approach in his ethnography on insurance claims processors, in order to ground the explanation of the 'communities of practice' framework that resulted from his study. The vignettes are situated examples of core practices in the participant company. The first of the vignettes highlights the 'TestFirst' design and development approach, which the developers are expected to use when writing product code, and its crucial role in the developers' practice. The second vignette focuses on 'Code Reviews', a quality assurance process that was introduced in the participant company about a year after the fieldwork started.

It was the team's daily stand-up meeting. As each developer gave a brief report on their current work task and any problems they were experiencing with it, one developer mentioned that they had not used the TestFirst approach "for a change." One of the senior developers reacted immediately, stating emphatically, "There is no excuse for ever **not** doing TestFirst; the design [contract] should always be defined before coding the solution".

A brief discussion of the importance of TestFirst concluded with a novice developer reporting that another developer had recently spent some time showing her how to use TestFirst and assuring the team that she would be using it from then on.

The formal code review process had been part of the development approach for several months. One of the code reviewers was looking through an 'A-team' developer's programs, the last in a set of four he had recently completed. As they discussed the code, the reviewer asked the developer to draw a diagram to clarify what he meant. The reviewer then added to the diagram whilst explaining her solution. Now that code reviews had been part of their daily practice for a while, the reviewer commented that she spent less time on each review with some developers as their code quality progressively improved, and she knew that "their tests would be fine if their others have been [recently]". So, she did not feel it was necessary to do detailed reviews and "check every line of code." With other, less experienced developers, she did more detailed reviews as she was "also doing training, and design" during these ones. She gave some suggestions for code refactoring to the developer which she said

were a "better design, less repetition of code, for [unit] tests").

These vignettes will be referred to again in section five in the description of local work practice and infrastructure.

3. Work Practice

As can be seen in Figure 1, work practice is characterised in this study in terms of Jacob's 'patterns of behaviour' and 'patterns for behaviour', which are cited by LeCompte and Schensul [10] in their discussion of the emphasis of ethnography on the concept of culture. Culture can be considered behaviourally in terms of what people are observed to do (actual behaviour) as opposed to what they are expected to do, or say that they do ('norms' or reported behaviour). Actual behaviour is portrayed by patterns of behaviour, expected behaviour by patterns for behaviour:

> Patterns of behaviour represent behavioral variations or choices in the group; patterns for behavior represent cultural expectations for behavior. (Ibid, p22-23)

3.1. Patterns For Behaviour

Patterns for behaviour (PFBs) convey what people are expected to do. In general, there are PFBs in prescribed approaches to getting work done in any organisation, and they are typified by formally defined company policies, procedures and guidelines. PFBs are often defined and referred to in official company documents, and have labels or names that have become part of the company's vernacular. If asked what they do and how they do it, people will often describe their work in terms of PFBs, reporting these as an accurate representation of their daily practice.

The traditional model of work implies that if one simply follows the processes i.e. the PFBs, the work gets done. Or, conversely, if the work is accomplished, the assumption is that it was because the process was followed to the letter. This research found that this representation of work is foreign to what actually happens in practice.

3.2. Patterns Of Behaviour

Patterns of behaviour (POBs) are observed work practices: the steps that people actually take and the activities that they perform in order to get their work done. They are often *ad hoc* activities performed in order to complete a formal task. Their implementation may even be facilitated, or implicitly supported, by the tools and processes officially provided. PFBs do not usually include a representation the work that is required to get the 'real' or 'primary' work done. This is called articulation work and can only be identified and described as a result of examining POBs. Suchman [24] contends that the recognition of articulation work is crucial to an understanding of everyday work practices. In the context of software development work, Grinter [7] defines articulation work as "all the coordinating and negotiating necessary to get the work at hand done". According to Gasser [5], articulation work is necessary to "keep some primary work going smoothly." Seely Brown and Duguid [15] refer to PFBs as 'canonical practice'. Using Orr's ethnographic studies of service technicians [11] to support their position, they stated that "reliance on canonical practice can blind an organization's core to the actual and usually valuable practices of its members (including non-canonical practice, such as 'workarounds')". This is problematic as it "is the actual practices, however, that determine the success or failure of organisations" (Ibid).

POBs may or may not be concomitant with the PFBs. Note, however, that there is no value judgement involved in defining behaviour as a PFB or a POB. In themselves, PFBs are not 'good' and POBs 'bad', or vice versa. What is important is the recognition that PFBs by themselves do not adequately describe work that is done. And, in fact, not even PFBs and POBs together are sufficient representations of work. The remainder of the paper emphasises that an accurate representation of the developers' work practice needs to include their POBs and their infrastructure, and the relationship between them.

4. Infrastructure

The meaning of "infrastructure" in this research is defined most aptly by Star and Ruhleder [21]:

> "Infrastructure is a fundamentally relational concept, becoming real infrastructure in relation to organized practices…Analytically, infrastructure appears only as a relational property, not as a thing stripped of its use" (p380)

The prefix *infra-* means 'below', and thus infrastructure could be interpreted as the structure underneath and the main system. As mentioned in section one, this definition is deficient. In the world of work, infrastructure refers to the tools, processes, rules, policies and guidelines that exist together in an organisation to underpin all the 'real' work performed by a group. However, infrastructure is created in its

use. It exists in, and is characterised by, its embodiment in work practice, it is not simply a prop. Infrastructure cannot be understood as a discrete, static phenomenon. The shape of the infrastructure, and the role that it plays, is a consequence of its context of use. A unique infrastructure is constructed within each working environment because of the work practices used there.

A paper by Star [20] advocates the examination of infrastructure as an essential part of the study of work practice. Infrastructure is generally regarded as background to more compelling and appealing research interests. Infrastructure may be considered mundane from a research point of view, but it is actually a very important aspect of what developers do in their daily work practice. As previously indicated in section 2.1., one of the characteristics of ethnography is that it examines and analyses the mundane and the taken-for-granted. Ethnography always probes formal and informal work practices, "not taking either for granted as 'the natural way' of doing things" (Ibid). Star sees "infrastructure as part of human organisation, and as problematic as any other part…foregrounding the truly backstage elements of work practice, the boring things" (Ibid).

This aspect of work (i.e. infrastructure) may not seem as exciting or inspiring to study as other features of software development work, but this does not mean that it is straightforward or easy to make sense of it. In fact, infrastructure is very difficult to 'pin down':

> "Infrastructure is usually singularly unexciting as a research object for ethnographers. The human, symbolic, interactive aspects of infrastructure are terribly difficult for ethnographers to "open up" in the way that we easily may open up conversations, rituals or gestures. Infrastructure often appears simply as a list of numbers of technical specifications, or black boxes, wires and plugs, in the scientific/disciplinary workplace. (Where is the human behaviour side of that?)…infrastructure can be messy and distasteful…" (Ibid: p109)

There are two main reasons for this elusiveness. The first one is that infrastructure is usually transparent, and taken for granted, by its users. Whilst they are familiar with it, understand very well how to use it, and do so regularly, they may not be able to recognise its role or its significance. The second reason has to do with the strangeness of an infrastructure to an observer, who initially has no comprehension of what comprises the infrastructure, or its unique role in the users' work practices. Software developers' electronic infrastructure is used by technically adept people, perhaps rendering it even more incomprehensible than other infrastructure. Even if the researcher is aware of the infrastructure, initially it may seem inscrutable. Fieldwork and data analysis from this perspective are very challenging: infrastructure is difficult to unravel, define and understand. Extensive fieldwork is crucial to gaining insights into this aspect of getting work done.

5. Local Software Development: Work Practice and Infrastructure

The development environment was largely realised in the infrastructural elements that the developers interacted with in their everyday work. Some of these were processes, policies, standards and other intangible elements; others were technology (software tools). Thus, studying the developers' work practices involved observing and investigating their use of development methods, project management policies, software applications and other infrastructure as tools of their trade.

PFBs formed a considerable part of the developers' infrastructure (refer to Figure 1), which included explicitly defined company processes, policies, rules and guidelines for development work, as well as the tools provided directly by the company to enable the developers to perform their work and various standards. Locally, PFBs were touted as company best practice which the developers were required to adhere to in their work practice, and for which they were held accountable if they did not.

In the main, infrastructure is comprised of processes and technology. Some of the local technology was proprietary software purchased by the company, for example, Microsoft's .NET framework; other tools were developed in-house, such as the automated testing system (ATS). Examples of explicitly advocated company policies and processes are the 'TestFirst' design methodology, and the 'Code Review' process. These policies and processes were usually complemented by appropriate software tools.

A significant amount of the infrastructural technology was designed and developed by the developers themselves. So, as well as developing a non-trivial software product for other users as their primary daily work, the developers had their own computerised information system as part of their infrastructure, most of which they designed and developed for themselves. It included applications and automated tools for downloading existing code onto their local work stations (checking out code), changing or adding program code, compiling and building code on local machines, designing GUIs/forms, code testing

(unit testing), submitting new or changed code (checking in), executing integration and system tests, and creating 'GoodBuild' versions of the software product. The infrastructure was set up and maintained to support designing, programming and testing software in an Agile environment. Furthermore, the same tools, processes and system architecture were used by some of the developers to develop and maintain infrastructure for product developers that were used for the development of 'Connect'.

The ATS was based on .NET classes, and carried out unit, integration and regression testing. It was executed automatically every 90 minutes or so, executing the entire set of tests for the 'Connect' product on several dedicated machines. The Automated Testing Monitor was in-house software which provided real-time reporting to the developers on the status of the automated integration tests and system builds. Other software tools supported the development processes; for example, the ATS included a program which verified that checked-in code adhered to the company's coding standards and sent email notifications to developers about code that did not.

The core practice of 'TestFirst', part of the Test-Driven Development methodology used by the company [2] and illustrated in the first vignette in section 2.3, was probably the strongest PFB specified, fundamental to the way the company designed and developed its software product. The CEO described TestFirst as "not optional". The developers were expected to create 'Unit Tests' for all modifications that they made to the program code, and these tests had to be written before any new functional code was written. Part of the test harness used in code development, a Unit Test is code written specifically to test one small piece of function code. If carried out as required, this practice ensured, firstly, complete test coverage of all functional code, and secondly, that those tests were executed as part of the ATS. If any Unit Tests failed during a test cycle, the developers were alerted to them, they could be identified and the error corrected before buggy product code was checked in (i.e. added to the main code base).

Experienced developers with significant service with the company were particularly passionate about TestFirst and adamant that no other approach was acceptable to produce software, as TestFirst produced robust code that met the requirements. For instance, after working unsuccessfully until midnight on a problem the night before, one senior developer said when he began to tackle it again,

"Now at the stage I'm writing the code before the unit test".
{"Do you do that often?"}
"If I'm on a tight schedule, more often. But it's not very productive. You think it's faster, but it's like speeding on your way home, quicker till you get caught, or worst case, wrap yourself around a tree!"

A relatively new developer with Raptor, who had used a more conventional design and development approach at the previous company they worked at, declared that "writing [unit] tests first clarifies that I understand what the problem is…" (before designing and coding the solution).

The POBs, i.e. observed practice, revealed that not all developers consistently used TestFirst. Several POBs occurred with regard to writing software:

- using TestFirst as prescribed i.e. designing and coding a Unit Test before coding the functionality;
- coding the functional code and then the Unit Test for that functionality;
- coding functional code without any Unit Test at all; and
- coding part of the functional code, then part of the Unit Test, then another part of the functional code, then another part of the Unit Test, and following this pattern iteratively

Increasingly, during the fieldwork period, unreliable, buggy code was being checked in, causing failing tests in the ATS cycle, and increasing the time interval between 'GoodBuilds', i.e. system builds that had no failing tests during the ATS cycle, and thus could be released to the clients as updated product code. Over a period of 18 months or so, the number of 'GoodBuilds' fell from three or four per day, to only a couple per week. More seriously, the TestRun sometimes resulted in 'GoodBuilds' that were actually 'BadBuilds', i.e. contained buggy code, as they included functional code that did not have Unit Tests written for it, so it did not cause test failures during the ATS cycle. The updated product code was then released without being tested at all, and any bugs and errors were discovered (too late) by clients using the software in their daily operations.

Another POB was related to the process of checking-in, i.e. uploading new, enhanced or fixed code to be added to the main product code base. Previously, it was up to the developers themselves to decide when it was appropriate to check-in their code, and they took responsibility to fix it if it was problematic. However, the aforementioned patterns of

behaviour related to TestFirst meant that some developers were checking-in buggy code which caused problems for others, developers and clients. Other developers could not check-in their own code because the TestRun was failing, and extra effort was required to keep track of results of the test process until they were able to check-in. Some clients, who had been given GoodBuilds which were actually BadBuilds, were running unreliable software.

Consequently, PFBs for formal code inspections and subsequent authorisation for checking-in code were introduced. Developers were required to have their code reviewed by a 'Code Reviewer' (senior developer whose daily work included these inspections) and have it authorised as being 'ready for check-in' before they attempted to check-in the code. The second vignette in section 2.3. is a description of one Code Review observed during the fieldwork.

The developers actually used their own local implementation of Connect, called 'Connexion', which they used for workload and task management, and to support development processes and project management. During the early part of the fieldwork, the use of Connexion by the developers was very much *ad hoc* and although some developers made good use of it, others did not appear to use it at all. Later, once the Code Reviews and check-in scheduling processes were introduced as PFBs, a more formal and intense use of the Connexion system to record, audit and monitor developers' work practices in this regard was adopted. Connexion was part of the developers' infrastructure and regularly modified to more effectively support their work practice.

6. Discussion: Work Practice and Infrastructure are Mutually Constitutive

This discussion of situated developers' work deliberately tries to avoid dichotomies that separate concrete from abstract, situated practice from formally defined procedures and the expected use of the technology. A fundamental problem in our understanding of software development work practice is that we confuse the PFBs (patterns for behaviour), and the technology that is designed to support them, for actual work practice, i.e. POBs (patterns of behaviour).

Whilst a procedural approach may be suitable for defining computer system behaviour and we can reasonably expect software to conform to these systematic steps, it is not an adequate way of describing what people actually do. Suchman [23] demonstrates that ignoring the situatedness of human action provides an inadequate picture of human behaviour and we are in danger of confusing theoretical knowledge with actual performance. Studying software developers' 'situated action' (Ibid) means that we are less likely to confuse expectations with actual behaviour, or theory with practice.

The field study for this research shows infrastructure facilitating and shaping certain POBs. POBs in turn shape infrastructure. And if the infrastructure is not used in the way intended or required, i.e., POBs are not consistent with the PFBs over an extended period, more infrastructure, in the form of PFBs and supporting tools, is added in an attempt to encourage compliance, e.g. Code Reviews were introduced to check on test coverage and the use of Unit Tests.

POBs, including those that are concomitant with the PFBs, are often more significant and have more value than straightforward compliance or developers 'doing things properly'. With the Code Reviews, for example, although the PFB for this was primarily designed for quality assurance, its use in practice was not limited to direct assessment of the standard or acceptability of the code. .Both reviewers and developers viewed a Code Review as a learning opportunity, with reviewers guiding the developers in the use of and importance of Unit Tests as a design approach in TestFirst, test coverage, coding standards and optimal design. This extra value is not at all apparent when one just looks at the formal documentation, i.e. the PFB, for Code Reviews, which consists of the steps that should be covered when performing one.

Without the infrastructure discussed, and the ability and freedom to use it and shape it dynamically to support their work practice, the developers' would be unable to do their primary work: designing and implementing a large and complex interactive system. One of the developers stated:
"A lot of the infrastructure development seems to be driven by a desire not to do the same [work] over and over. If a process is repetitive enough to be automated, then someone will write a tool to automate it. Then, as the tools are built, they expose new bottlenecks in the development process that can be improved by a new set of tools. Then as the new tools exist they open up new possibilities to improve the process."

The developers in this study are enmeshed in their infrastructure. They have a deep understanding of the technology that they use in turn to design and develop

technology for others: they build it and change it and use it to effectively enable their own daily work. They continually consider how it is designed, how it can best be exploited, and how it can be adapted or extended to improve the support it provides to their primary work.

7. Conclusion

This paper is based on fieldwork done during a longitudinal ethnographic study of a group of professional software developers. The aim of the study was to gain an understanding of what these developers were observed to do in their everyday work. This sense-making research was open-ended, and no theory or particular framework was imposed on the data during its analysis. The resulting issues were unanticipated and emergent, strongly grounded in the comprehensive fieldwork data, and revealed that an authentic account of the developers' work needs to include:

- their work practice, comprised of:
    - 'patterns of behaviour' - what they were observed to actually do,
    - 'patterns for behaviour' that the company expected them to follow;
- the infrastructure underpinning their primary work; and
- most important, the mutually constitutive relationship between their work practice and their infrastructure.

Infrastructure in this research has a particular, more complex meaning than it is commonly given. Here, it includes the technology, formal processes and standards that enable the primary work of software production, but it is regarded as relational rather than as a static supporting structure.

The developers' work practice and infrastructure become progressively more entwined over time because they are mutually constitutive. This finding is noteworthy principally because it makes visible the developers' extraordinary skills in accomplishing their daily work. Software development practice is far more intricate and demanding than merely using particular tools or complying with formal processes. The developers continually mesh their work practice and their infrastructure for their particular work context.

The recognition of these sophisticated skills has implications for software engineering research, theory and praxis. Software development education and training currently focuses on separate infrastructural elements such as formal processes (patterns **for** behaviour) and tools. Innovative and effective ways of teaching skills based on professionals' meshing abilities, and other patterns **of** behaviour, to students and novice developers need to be explored. Similarly, software project management usually acknowledges the patterns **for** behaviour, but approaches that make visible and account for patterns **of** behaviour (including the activities performed in meshing infrastructure and work practice) should be investigated.

8. Acknowledgements

The first author was tremendously privileged to share in the daily working lives of the CEO and developers at Raptor Systems, who willingly and enthusiastically allowed her to scrutinise their practice. Great appreciation and thanks to them all. Grateful thanks also to Tim Mansfield, who provided invaluable advice on both the writing of this paper and the first author's PhD work.

9. References

[1] Agile Alliance, "The Agile Manifesto", 2001, http://www.agilealliance.org/intro [accessed 10th Feb 2006].

[2] Beck, K., *Test-Driven Development by Example,* Pearson Education, Inc., Boston, 2003.

[3] Bentley, R., Hughes, J.A., Randall, D., Rodden, T., Sawyer, P., Shapiro, D. & Sommerville, I., "Ethnographically-informed systems design for air traffic control", in *Proceedings of the ACM Conference on Computer Supported Cooperative Work,* ACM, Toronto, Canada, 1992, pp123-129.

[4] Cockburn, A., *Agile Software Development,* Addison-Wesley, Boston, 2002.

[5] Gasser, L., "The Integration of Computing and Routine Work", *ACM Transactions on Office Information Systems 4,* 3 (July 1986), pp205-225.

[6] Glass, R.L., Vessey, I. & Ramesh, V., "Research in software engineering: an analysis of the literature", *Information and Software Technology* 44 (2002), pp491-506.

[7] Grinter, R.E., "Supporting articulation work using software configuration management systems", *Computer Supported Cooperative Work (CSCW) 5*, 4 (Dec 1996), p447-465.

[8] Harper, R., "The Organisation in Ethnography - A Discussion of Ethnographic Fieldwork Programs in CSCW", *Computer Supported Cooperative Work (CSCW)* 9, 2 (May 2000), pp239 - 264.

[9] Jordan, B., *Ethnographic Workplace Studies and Computer Supported Cooperative Work,* Institute for Research on Learning, Palo Alto, 1994.

[10] LeCompte, M.D. & Schensul, J.J., *Designing and Conducting Ethnographic Research (The Ethnographer's Toolkit, Book 1)*, AltaMira Press, Walnut Creek, 1999.

[11] Orr, J. *Talking about machines: an ethnography of a modern job*, Cornell University Press, 1996.

[12] Prior, J., Robertson, T. & Leaney, J., "Programming Infrastructure and Code Production: An Ethnographic Study", *Ethnographies of Code workshop: Computer Programs as the Lived Work of Computer Programming*, Lancaster, U.K., proc. in TeamEtho-online, issue 2 (June 2006), p112-120.
[13] Robinson, H. & Sharp, H., "XP Culture: Why the twelve practices both are and are not the most significant thing", *Agile Development Conference*, Salt Lake City, 2003.
[14] Schön, D.A., *The Reflective Practitioner: How Professionals Think in Action,* Arena, Ashgate Publishing Ltd, 1983.
[15] Seely Brown, J. & Duguid, P., "Organizational Learning and Communities-of-Practice: toward a unified view of working, learning, and innovation", *Organizational Science 2*, 1 (1991), pp40-57.
[16] Sharp, H. & Robinson, H., "An Ethnographic Study of XP Practice", *Empirical Software Engineering 9* (2004), pp353-375.
[17] Sommerville, I., Bentley, R., Rodden, T., Twidale, M. & Sawyer, P., 'Integrating Ethnography into the Requirements Engineering Process', *Proc. 1st IEEE Int. Symposium on Requirements Engineering*, San Diego, IEEE Computer Society Press, January 1993.
[18] Sommerville, I., "Making ethnography accessible", *Proc. Workshop on Software Engineering/HCI*, Institute of Electrical Engineers, Edinburgh, May 2004, pp36-40.
[19] Sommerville, I., Martin, D., Rouncefield, M., & Rooksby, J., "'Good' Organisational Reasons for 'Bad' Software Testing: An Ethnographic Study of Testing in a Small Software Company", *Proc. 29th Int. Conference on Software Engineering (ICSE'07)*, IEEE Computer Society, May 2007 pp602-611.
[20] Star, S.L. "Infrastructure and ethnographic practice", *Scandinavian Journal of Information Systems 14,* 2 (2002), pp107-122.
[21] Star, S.L. & Ruhleder, K., "Steps Toward an Ecology of Infrastructure: Design and Access for Large Information Spaces", *Information Systems Research 7*, 1(1996), pp111-134.
[22] Suchman, L. "Office Procedure as Practical Action: Models of Work and System Design", *ACM Transactions on Office Information Systems 1*, 4 (1983), pp320-328.
[23] Suchman, L., *Plans and situated actions: the problem of human-machine interaction*, New York, U.S.A, Cambridge University Press, 1987.
[24] Suchman, L., "Supporting Articulation Work", in *Computerization and Controversy*, ed. R. Kling, Academic Press, Inc., Santiago, California, 1996, pp407-423.
[25] Viller, S. & Sommerville, I., "Ethnographically informed analysis for software engineers", *International Journal of Human-Computer Studies 53 (*2000), pp169-196.
[26] Wenger, E. *Communities of Practice: Learning, Meaning, and Identity*, Cambridge University Press, Cambridge, U.K., 1998.