TeamEthno-online, issue 2, June 2006, 112-120

Programming Infrastructure and Code Production: An Ethnographic Study

Julia Prior
Department of Software Engineering
Faculty of Information Technology
University of Technology, Sydney
Australia
email: julia@it.uts.edu.au

Toni Robertson
Department of Computer Systems
Faculty of Information Technology
University of Technology, Sydney
Australia
email: toni@it.uts.edu.au

John Leaney
Department of Software Engineering
Faculty of Information Technology
University of Technology, Sydney
Australia
email: John.Leaney@uts.edu.au

Abstract

Infrastructure is an essential part of all human endeavours. Instead of viewing the infrastructure necessary for the production of software systems and the program code as separate entities and independent of one another, this paper offers the view that neither the infrastructure nor the code are discrete things that can be examined in isolation. They are phenomena that are inseparable; they intertwine and interact, shaping each other over time. The research is based on a longitudinal ethnographic study of work practice in a software development company using an Agile development approach.

"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."
(Glass, Vessey et al. 2002)

1. Introduction

Some form of infrastructure is needed for all human activities. The Concise Oxford English Dictionary defines infrastructure as 'the subordinate parts of an undertaking', or the basic physical and organisational structures needed for the operation of a society or enterprise. The prefix *infra-* means 'below', and thus infrastructure could be interpreted as that which is underneath or supports the main system. Most of us are familiar with the infrastructure provided by electricity and telecommunication systems, for example. These constructions sustain and shape the way that we get things done in our everyday lives. 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. In software development, infrastructure to support the production of code is comprised of, for instance, programming languages, code editors, compilers, testing environments, form design tools, database management systems, version control software, development methodologies, processes and techniques, and programming style standards. Not to mention hardware, utilities, people and anything else that maintains the physical, social and cultural environment in which the developers work.

Some research seems to address software development (i.e. the production of program code) and the infrastructure that is necessary to support this endeavour as two discrete entities. Each is treated as though it exists, and can be defined and realised, as an independent, isolated thing. From a phenomenological standpoint, it is not possible to adequately understand phenomena in this way. The central theme of this paper is that any software development infrastructure is unique, dynamically created, maintained and adapted over time, and cannot be defined or understood outside of the setting which shapes it.

The infrastructure available, created and maintained in one software development company is examined as part of a longitudinal ethnographic study of software developers. It is apparent from this work that infrastructure is relational and dynamic. Code production and infrastructure are not discrete phenomena – in situated practice, they are inseparable, intertwined and, over time, shape each other.

The first section of the paper gives some background to the work: there is a description of the research site and the fieldwork done, and the concept of infrastructure and the role it plays in an organisation is discussed. The main section deals firstly with the code and infrastructure of the participant company – the tools, processes and policies used by the software developers for code production - as well as the creation, maintenance and use of local software development infrastructure. Secondly, two issues emerging from the ethnographic work are considered: one person's infrastructure is another's core work, and the interlacing of infrastructure and program code development.

2. Background

Over the past 20 months, the first author has been doing fieldwork in a software development company that specialises in developing software products for the freight forwarding, logistics and customs brokerage industry. The fieldwork to date consists of 45 site visits, each lasting between three and eight hours: the developers' everyday work practices in their normal work environment were observed, company documents, policies and resources such as email were investigated, meetings attended and conversations held with the developers. The data collected is mostly in the form of fieldnotes. There are also email communications between developers, company documents, including statistics, and some audio recordings and photographs.

The first author has considerable software development and programming experience, and this knowledge has enabled her to observe and understand the work practices of the participating developers at a much deeper technical level than would have been possible by a 'lay' fieldworker. Whilst the presence of the fieldworker would have affected the developers' behaviour, we believe that this effect was minimised by numerous visits over a long time and the developers' familiarity with and acceptance of her presence. As a software developer, her observations have probably also been less intrusive and disruptive to the developers' work and productivity.

The participant company has been extraordinarily open to this research, and essentially gave the first author *carte blanche* to investigate and observe wherever she deemed necessary. Over time, her presence has become unremarkable to the developers. She has built good relationships with them, and they trust her to maintain confidentiality, their privacy and anonymity.

A key stance taken in this paper is the dynamic, shifting nature of both infrastructure and code. Themes discussed here have emerged as a result of observations made over a lengthy period, and would not have been manifest in a short-term study.

2.1. Fieldwork Site Description

The company does not develop customised software for individual clients, but rather develops software products that support the rules and regulations of the freight, logistics and customs industry, and clients in this industry purchase these products to support their own operations. The flagship product is a large, complex software suite called Connect. The number of software developers during the fieldwork period has grown from 50 to 60, the majority of whom work solely on the development of Connect. The term 'developers' in this research refers to people whose major objective is to produce working program code

as part of a complex software system product. Although a very significant part of their time is spent programming, their jobs require more than simply writing code, and the term 'developer' describes their role more completely.

The organisational hierarchy is almost flat. The developers work in an environment that is open-plan in one large room where everyone – developers, managers and support – has their workstations. No one has a separate office, including managers and more senior developers. They all work side-by-side, with the same kinds of desks, chairs and workstations. There are several developer teams, each of which focuses on one module of Connect e.g. Freight, consists of a mix of senior and junior developers and has a team leader. Other company departments, such as marketing, training and client support, are organised in a similar way.

2.2. Infrastructure

A paper by Star advocates the examination of infrastructure as an essential part of the study of work practice (Star 2002). A significant part of the research discussed in this paper is based on her work and applied in the context of my fieldwork. Infrastructure is assumed to exist and to be functional, but it is generally regarded as background to more compelling and appealing research interests. Infrastructure may be considered to be mundane from a research point of view, but it is actually a very important part of what developers do in their daily work practice. 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 (Ibid) sees "infrastructure as part of human organisation, and as problematic as any other part…foregrounding the truly back stage elements of work practice, the boring things."

Although it is used in the first paragraph of the introductory section of this paper, the common understanding of infrastructure, in which infrastructure is viewed as a substrate, a thing on which some other thing 'runs' or 'operates', is an inadequate, incomplete representation. In an earlier paper, Star emphasises that "infrastructure is fundamentally and always regarded as a *relation*, never a thing" (Star and Ruhleder 1994).

In a similar vein, Bucciarelli talks about a web of infrastructural elements – strands and lines with interconnections at various levels. These interconnections are dynamic, not static: existing ones are continually expanding and contracting, and new connections are being made. He characterises infrastructure as "a dense, interwoven fabric that is, at the same time dynamic, thoroughly ecological, even fragile." (Bucciarelli 1994)

Star suggests that users are probably not explicitly aware of the different, specific components and connections of their infrastructure, unless something goes wrong with it (Star 2002). This cannot really be said of the developers in the participant company, as they have a very good understanding of their infrastructure and its role in their work, and in fact have constructed much of it themselves, and continue to do so. The Core Team and other more experienced developers in particular, work with infrastructure as part of their daily work, and understand its significance to the software development work. However, in some ways, the developers do appear to consider it unremarkable. For instance, one of the ways to identify an infrastructure component is that the developers do not explicitly talk about using it. So, when explaining what work they are currently busy with, a developer will refer to the design or development activity itself, not the tool(s) they are using to perform the task, for example: 'I am changing the layout of this form', or 'I am changing the functionality here…'. They do not say something like, 'I am using SourceSafe to check-out the class where the functionality needs to be changed…I am using Visual Studio's Window Forms Designer to define the layout of the form…'. If they are following a process, and

maybe using a software tool to do so, they will remark, for instance, 'I need to get this [completed job] scheduled for check-in', and then complete the task record in the task management software system as 'pending Check-In' to actually accomplish this task. It is often in what they do not specifically mention, but are actually using to perform a task, that one can identify some kind of infrastructural element or tool, one that the developer seems to be 'taking for granted' and does not explicitly acknowledge.

3. Programming Code and Infrastructure

An examination of the local software development environment illustrates that code and infrastructure are inseparable. Sometimes, infrastructure is realised as code. Code used as infrastructure by some developers is the focus of other developers' daily work. Infrastructure changes shape as the code development effort requires different tool and process support.

3.1. Local Software Development Environment

The development approach used in the participant company is strongly Agile (Agile Alliance 2001). 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 (Cockburn 2002).

Thus, program code is the major software artefact. Although requirements are progressively documented, there is very little in the way of formal design diagrams or separate documentation of development decisions and process. The focus of the development effort is producing working program code. The development infrastructure is set up and maintained to support programming and testing in an Agile environment.

As well as developing a non-trivial software product as their primary daily work, the developers have their own computerised information system: applications and automated tools, mostly proprietary, for downloading existing code onto their local work stations (*checking-out*), 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*), automated system/integration tests and building '*Release*' versions of the software product. Some of the automated tools and processes the developers use are customised or developed in-house by the developers themselves. There are also company policies that the developers are expected to follow as they develop programs. In other words, there is an infrastructure, comprised of tools, processes, rules and guidelines, which underpins and interacts with the developers' core work of making and enhancing a software product to support a particular industry.

Table 1 describes some parts of the local development infrastructure. Some of these entries are coding tools, others are processes or policies. The table only lists a few of the infrastructural elements that the developers deal with in their everyday work. It should be noted that a table has been used for brevity, and its use should not be taken as an over-generalisation and simplification of the infrastructure employed by the developers. As discussed earlier, infrastructure is not regarded as a thing. Its elements are of interest and are significant only in their situated use and interaction with each other, and with other aspects of code production and infrastructure.

Infrastructural Element	Source	Purpose
Integrated Development Environment (IDE): Visual Studio for the .NET framework	External, proprietary software: Microsoft	Software development – code editor, compiling, runtime environment, .NET base class libraries, etc
C# programming language	External, proprietary software: Microsoft	Development language used for writing object-oriented programs to run on the .NET Framework
Visual SourceSafe	External, proprietary software: Microsoft	Keeps track of what code is *checked out* to which developers and handles version control of all program code files
Unit Tests	In-house code modules	Part of the test harness used in code development, a *unit test* is code written specifically to test one small piece of function code; specific, in-house software is used to execute and verify these tests.
TestFirst	Design approach used in Agile software development.	Core practice: the developers are expected to create *unit tests* for all modifications that they make to the program code and these tests must be written before any new functional code is written.
Automated Testing System	In-house software, based on .NET classes	Integration and regression testing; executed automatically every 90 minutes or so, executing the entire set of Unit Tests for the Connect product and other system tests on several dedicated machines.
Automated Testing Monitor	In-house software	Real-time reporting on the status of the automated integration tests and system builds in Monitor's web pages.
CArch	In-house software	Architectural layer of base classes lying between .NET framework and program code that comprises the Connect product.
Code Inspections	Company policy	Before any code may be *checked-in*, it must be reviewed by another, usually senior, developer. An in-house tool for task management is used to record status of development tasks.
Email	External, proprietary software: Microsoft's Outlook.	Many different functions, one of which is to notify developers of recent testing events, and also used extensively by developers to discuss design, coding and technical issues.
Connexion	In-house software	Used to manage the development process: work to be done, task assignment, status of tasks, check in scheduling etc.

Table 1. Local software development environment

The approach to code design and production is Test-Driven Development (Beck 2003). Testing is an integral part of their design and development approach, and the most significant element of the development methodology used by the company is the principle of *TestFirst*, implemented in the coding of *unit tests* before coding any functional code.

One of the most important tools for development is the automated integration and regression testing system. This software is based on the .NET framework classes for unit and system testing, but is primarily developed and maintained in-house, in particular the AutoTesting Monitor, which plays a crucial role in the test and build cycle. The results of the latest automated test are displayed in the Monitor's web page, and each developer has a Monitor icon on the status bar of their desktops so that they can access it quickly. If all the tests pass, the solution code can be released to current clients to upgrade their implemented Connect system.

3.2. Infrastructure for Some Developers is Core Work for Others

In some cases, one developer's infrastructure is another's core work. If a developer uses some element, application or object as a tool, then it is probably forms part of their infrastructure, whereas if they produce it, or make it, it is actually not part of their infrastructure, but their 'core business'. For example, CArch provides a framework of the base classes for most major objects in Connect. It manages object persistence, provides GUI (graphical user interface) controls, defines what validation must be implemented and how, etc. The product developers use CArch extensively to develop the code for Connect. CArch extends the .NET framework, and is developed and maintained by part of the Core development team. Thus, CArch is not part of this team's infrastructure, as developing it is part of the purpose of their work i.e. to provide good architectural infrastructure to the product developers. However, for an application/product developer who uses CArch objects while working on developing software that forms part of the Connect product, CArch is infrastructure because they are using it to develop application code.

The testing harness (which includes regression tests, unit tests, integration tests, reflection tests and code standards tests) is based on classes in .NET, but is written and maintained in-house by members of the Core Team. This team also develops infrastructural elements such as: the automated testing Monitor, software to check that developers are using the required programming style, software for managing error reports from clients' systems, software for managing bug fixes, and software for managing programming jobs, often created from error and bug lists. On the whole, the same tools, processes and system architecture are used to develop and maintain infrastructure for product developers that are used for the development of Connect.

The same applies to database development: for most of the developers the database and the database management system software are infrastructure components, for the database developers (who are members of the Core team), the database is not infrastructure, it is core work.

Some senior developers move between the roles of product developer and member of the architecture (Core) team, so they are not just building a software product, but are simultaneously building infrastructure to enable this product development. Their infrastructure will have a different shape depending on the role they are fulfilling at a particular time.

3.3. Programming Code and Infrastructure

Infrastructure is created in its use. It exists in its ability to be embedded in work practice, as an actor, not simply as a prop. 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 as a result of the work practices used there. Star and Ruhleder (Star and Ruhleder 1996) expressed this as follows:

> "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).

For instance, about a year into the fieldwork, two new processes were introduced to improve the quality of the code released and to shorten the interval between getting 'Good Builds' i.e. system builds that could be released to the clients. The process of *checking-in* new or fixed code used to be informal and *ad hoc*: 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, *checking-in* buggy code causes problems for others, developers and clients. For other developers, they cannot *check-in* their own code because the TestRun is failing, and extra effort is required to keep track of results of the test process until they are able to *check-in*. For clients, who may be given 'Bad Builds' in the next code release, they would be running unreliable software.

The first of these processes is that all code now has to be inspected in a formal code inspection by another developer and has to be deemed acceptable before it can be *checked-in* to become part of the next new code release. This process formalises the idea of someone other than the author developer verifying the code, and supporting their indirect claim (by wanting to *check* it *in*) that the code is ready for release to clients. Any development work required to be done is recorded as a job in a database, and only once that job has been given a formal status of 'reviewed and approved' may the developer request the *check-in* of the work.

The second of the new processes is Check-In Scheduling. Instead of the developers taking the decision to *check-in* their code themselves, nowadays they have to mark the job as 'ready for check-in' after a successful code inspection has been done. The Check-In Scheduler, a senior developer in charge of this process, goes through the job database several times a day, and prioritises and schedules the jobs to be checked-in. Once a developer's work is scheduled for check-in, they are notified of this, and can then *check-in* their work in the usual manner.

The effect of these processes on the production code is two-fold: firstly, the senior developers who have done the code inspections since the process was introduced (for the first few months only 2 developers performed code inspections; more developers now perform this function) have stated that they have noticed a marked improvement in the quality of the code, measured mostly in terms of adherence to coding standards and test coverage (100% coverage in a program class would mean that every line of code has been tested in some way in a unit test). Also, in the first few months after the inspections were introduced, the number of release builds increased from one every three or four days, to one every few hours, indicating that the code being produced was considerably less 'buggy' and problematic. So, the program code itself is probably different to how it would have been coded prior to the introduction of the code inspection process. Secondly, as a result of the *check-in* scheduling process, what actually constitutes the final product code at any one time is dependent on the decisions made by the Check-In Scheduler, rather than being the result of every developer checking in all their work whenever they

consider it to be ready, so again the code has a different shape because of the infrastructural processes used in its development.

Code production and its development infrastructure are closely intertwined with one another, difficult to prise apart and examine separately. And, probably, they should not be treated in this way. It is not possible to program code without some sort of infrastructure, however inadequate, and the infrastructure has no meaning, or even existence, without the development effort that it supports. And more than being necessary for each other's existence, code and infrastructure are actually mutually constitutive: as code is developed, infrastructure is created, improved and maintained, and as the infrastructure becomes more sophisticated and effective, complementary code is produced, and the programming effort is better supported and facilitated.

4. Conclusion

Bucciarelli in his work on Engineering Design (Bucciarelli 1994) declared that

> "if we allow the [design] object to fix our view of designing, we see only hard edges, formal drawings, irreversibly machined and mated pieces…This view is static, ahistorical, and rigid – a lifeless landscape in which everything is rationally determined. In fact, designing is otherwise." (p149)

There are a number of other issues related to infrastructure and programming code that have emerged from the ethnographic study done in this research which will be addressed in future work. They are, for example, the relationship between product size and infrastructure, formal and informal infrastructure, the transparency and opacity of infrastructure, and the enabling/disabling paradox of infrastructure.

The most significant points made in this paper are:

- infrastructure is temporal – it takes shape over time;
- infrastructure is continually and dynamically shaped by the code production effort that it supports;
- code production is affected by the infrastructure sustaining it; and
- consequently, code production and its supporting infrastructure cannot be considered apart from one another as they are interwoven and interdependent.

Although they may not acknowledge this directly, infrastructure is part and parcel of the work environment and the work done by the developers in the participant company. They make use of infrastructure to develop software; they create, maintain and enhance infrastructure to support software development; their approach to software development is adjusted periodically to fit with changing infrastructure; some of them construct infrastructure for other developers to use; and in their development of software product, they are providing the means to construct infrastructure for third parties (i.e. their clients). Code production and infrastructure are not discrete phenomena – in situated practice, they are inseparable, intertwined and, over time, shape each other.

Acknowledgements

Great appreciation and thanks to the CEO and developers at Raptor Systems, who have willingly opened their work practices up to the scrutiny of the first author, who is tremendously privileged to have the opportunity to share in their daily working lives.

References

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

Beck, K. (2003). Test-Driven Development by Example. Boston, Pearson Education, Inc.

Bucciarelli, L. L. (1994). Designing Engineers. Cambridge, Massachusetts, USA, MIT Press.

Cockburn, A. (2002). Agile Software Development. Boston, Addison-Wesley.

Glass, R. L., I. Vessey, et al. (2002). "Research in software engineering: an analysis of the literature." Information and Software Technology **44**: 491-506.

Star, S. L. (2002). "Infrastructure and ethnographic practice." Scandinavian Journal of Information Systems **14**(2): 107-122.

Star, S. L. and K. Ruhleder (1994). Steps towards an ecology of infrastructure: complex problems in design and access for large-scale collaborative systems. ACM conference on Computer supported cooperative work, Chapel Hill, North Carolina, United States, ACM Press.

Star, S. L. and K. Ruhleder (1996). "Steps Toward an Ecology of Infrastructure: Design and Access for Large Information Spaces." Information Systems Research **7**(1): 111-134.