Technology Designers as Technology Users: the Intertwining of Infrastructure and Product

Julia Prior **Toni Robertson** **John Leaney**
Department of Software Engineering
Faculty of Information Technology
University of Technology, Sydney
Australia
julia@it.uts.edu.au, toni@it.uts.edu.au, John.Leaney@uts.edu.au

ABSTRACT
This paper is about the developer as technical user interacting with computer technology as part of the infrastructure that makes possible their 'real work' of developing a large and complex software product. A longitudinal ethnographic study of work practice in a software development company that uses an Agile development approach found that the developers spend a large part of their working time designing, creating, modifying and interacting with infrastructure to enable and support their software development work. This empirical work-in-progress shows that an understanding of situated technology design may have implications for the future development of HCI methods, tools and approaches.

Author Keywords
Ethnography, infrastructure, software development, technology design, technology use.

ACM Classification Keywords
H5.m. Information interfaces and presentation (e.g. HCI): Miscellaneous.

INTRODUCTION
Software developers are specialists in designing, creating and improving software systems. End-users' expertise is almost always in a different domain (i.e. not computer software) and they generally make use of computer systems designed by someone else to support their daily work. Developers, however, are technically savvy users of technology who have an understanding of how that technology works and how to design and produce the technology, as well as knowing how to use it.

In practice, software developers make use of an infrastructure that supports and enables the design and production of software products. The infrastructure available, created and maintained in one software development company was examined as part of a longitudinal ethnographic study of professional software developers' work practice.

Understanding the situation of technology use is fundamental to HCI. This paper is about technology designers as technology users, interacting with the software tools and procedures which comprise the local development infrastructure, in order to perform their primary work of making other software products.

The paper is structured as follows: the background section gives an overview of the fieldwork and a site description; a discussion of infrastructure and its role is followed by a section describing the local use of infrastructure and the practice of technology designers (software developers) as technology users; and finally the conclusion reflects on the implications that an understanding of situated software development work practices may have on the design and usability of software products developed for end-users.

OZCHI 2006, November 20-24, 2006, Sydney, Australia.

OZCHI 2006 Proceedings ISBN: 1-59593-545-2

BACKGROUND
Over a period of two years, the first author has been doing ethnographic research in an Australian software development company. 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.

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.*

There are several developer teams. Each focuses on one module of *Connect* e.g. Freight or Customs, and consists of a mix of senior and junior developers, including a team leader.

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).

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 'Big Upfront Design'. 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 software design artefact and the focus of the development effort is producing working program code. The high level design is managed by the *Core Team*, developers responsible for the overall architecture of *Connect.* Thus, in this paper, when the word development is used, it encompasses software design as well as code production.

INFRASTRUCTURE

An environment is "a complex of surrounding circumstances, conditions, or influences in which a thing is situated or is developed, modifying and determining its life or character", according to the online wikipedia [http://en.wikipedia.org/wiki/Environment]. Part of the environment is contained in an infrastructure, which, "most generally, is a set of interconnected structural elements that provide the framework supporting an entire structure" [http://en.wikipedia.org/wiki/Infrastructure].

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.

A paper by Star advocates the examination of infrastructure as an essential part of the study of work practice (Star 2002). Infrastructure 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."

The common understanding of infrastructure, in which it is viewed as a substrate, a separate entity on which some other thing 'runs' or 'operates', is an inadequate, incomplete representation. 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).

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)

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.

TECHNOLOGY USE IN TECHNOLOGY DEVELOPMENT

An examination of the local software development environment illustrates that code and infrastructure are inseparable. Sometimes, infrastructure is realised as code. Most importantly, for the concerns of this paper, 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. The developers 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.

The infrastructure is set up and maintained to support designing, programming and testing in an Agile environment. On the whole, the same tools, processes and system architecture are used by the developers to develop and maintain infrastructure for product developers that are used for the development of *Connect.*

The *Core Team* develops infrastructural elements such as the automated testing harness, 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.

So, as well as developing a non-trivial software product for other users as their primary daily work, the developers have, as part of their infrastructure, their own computerised information system, most of which they design and develop for themselves: 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.

The development environment is largely realised in the infrastructural elements that the developers interact with in their everyday work. Some of these are technology (software) tools, others are processes or policies

When developers and their users are working within the same environment and reliant on the same infrastructure, then the accountability for how their software is used is part of the infrastructure, too. This means, in some sense, this is probably the most ideal environment for producing usable and useful software. This of course, is one of the fundamentals of Agile development i.e. that the user is working full-time in the same room as the developers. For example, Rittenbruch et al (Rittenbruch et al, 2002) demonstrated the very strong complementarities between Agile approaches and Participatory Design

In the remainder of this paper, a couple of the infrastructural elements are discussed as examples. For a fuller description of the local infrastructure and its implications for the work practices of professional software developers, refer to Prior et al (Prior et al, 2006).

Technology for Product Testing

Testing is an integral part of their design and development approach (Beck 2003), 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. Essentially, *TestFirst* is a design approach in which a test for the change or addition to the software product is designed and implemented in a unit test, and the unit test code is executed and tested before the new functional code is written.

The software that the developers employ to execute their unit tests and provide feedback on the results is part of the automated integration and regression testing system, one of the most important tools for development used by the developers. This software is based on the .NET framework classes for unit and system testing, but is primarily developed and maintained in-house by the *Core Team*. The *AutoTesting Monitor* 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* systems.

The testing process has been made possible because of software that they have written to enable, realise and support it, and which they know that they can trust. And they can trust it because it was built out of identified user need in the first place, and during development, bugs were found by users and fixed, bad performance of the system was constantly improved, there was a change in the information given by the *Monitor* to make it more pertinent and useful, in response to the demands of the users. It was not coincidence that this technology works so well, nor was it a trivial accomplishment. The important point is that by intention and design, constant iteration and user-driven improvement are part of the design process.

Technology for Product Building

During the fieldwork, 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 (submitting new or fixed code to be added to the product code base) 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, both developers and clients. For other developers, they cannot check-in their own code because the current system-level test 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.

One of these 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 coding 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.

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 the product code has a different shape because of the infrastructural processes used in its development.

The above is an example of the adoption of a new process, and the development of the technology to support it, as a result of user response to breakdowns in previous procedures. Again, as with the *AutoTesting Monitor* example, the technology to support a new process was built out of identified user need and iteratively improved.

CONCLUSIONS

The paper described what is done in professional software development practice, focusing on the use of computer technology by technical users (i.e. software developers) to enable their 'real work' of developing a large, complex software product. A significant amount of this infrastructural technology is designed and developed by the developers themselves.

The developers in this study are enmeshed in their infrastructure: they have a deep understanding of the technology that they in turn use to design and develop technology for others: they build it and change it and use it to effectively enable their 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. These technically savvy users' heightened level of agency results in an expectation that the technology developed for them by their colleagues should adhere to high usability standards.

Those developers responsible for the infrastructure have to be and are mindful of their users (i.e. the other developers) who are also their colleagues, and are directly accountable to them. True to the Agile development approach, the users are working in the same room as the *Core Team* developers producing their infrastructural technology. If this technology does not provide the required functionality or is not usable in some way, the users communicate this directly and immediately to members of the *Core Team*, who may even be working at next-door desks. Real user acceptance testing (in contrast to simulated use situations) and accountability to their users are embedded in the development of the infrastructure technology.

There are two strong implications of this work for HCI design practice. Firstly, that the way that software products are designed and developed in situated practice, as examined in this paper, does not appear to match textbook assumptions about how code is or should be developed. Design and development approaches that enhance and increase the usability and usefulness of software need to be researched in terms of how software is actually made in practice. An understanding of situated technology use in software product development may be a worthwhile starting point.

Secondly, it is clear that the infrastructural technology discussed in this paper, designed and used by the developers themselves, is usable and useful. Characteristics of software usability are embedded in their infrastructure code, by design, and as part of the ongoing development and maintenance of the company's infrastructure. Given that the developers use the same processes, tools and approaches for developing a software product such as *Connect* as they use for developing their own infrastructure, the issue then arises of how these usability characteristics extend into other environments where the software product is used; in other words, in the workplaces of the 'end-users', enmeshed in their own infrastructures.

This study is a work-in-progress, but hopefully will open up discussion about the relationship between existing and emerging issues of importance to HCI and our understandings of the environments in which software is created and used. That is to say, it could open up further investigation of the effects of situated software development work practices on the design and usability of software products developed for end-users.

ACKNOWLEDGMENTS

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.

Prior, J., Robertson, T., Leaney, J. (2006). "Programming Infrastructure and Code Production: An Ethnographic Study." Ethnographies of Code workshop, proc. in TeamEtho-online, issue 2, June 2006, p112-120.

Rittenbruch,M., McEwan, G., Ward, N., Mansfield, T., Bartenstein, D.(2002) Extreme Participation - Moving Extreme Programming Towards Participatory Design. In: Binder, T., Gregory, J. and Wagner, I. (eds.) PDC '02, Proceedings of the Participatory Design Conference, Malmoe Sweden, June, 23-25, 2002, pp.29-41. CPSR. ISBN 0-9667818-2-1

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

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.