What are Software Practice Studies Good For?

Olle Lindeberg
Dept. of Software Engineering and Computer Science, University of Karlskrona/Ronneby
SE-372 25, Ronneby Sweden, +46 457 38 58 30
Olle.Lindeberg@ipd.hk-r.se

Abstract

Software engineering and ethnography has fundamental different perspectives on practice, and different aims. Is it possible to use results from ethnographic studies when doing software-engineering research? In this article some of these problems are taken up and exemplified by trying to use some ethnographic inspired studies to shed some light on a comparison of two different philosophical understandings of what the software development process is about.

.

Keywords:

Software development, ethnography, methods.

1 Introduction

The perspective on work practice in ethnography and in software engineering research is quit different. In description of ethnographical methods the respect for the observed is emphasized. The observed persons are seen as knowledgeable and informed experts on their own practice. The ethnographer on the other hand should have a humble attitude, being a novice in the field and trying to learn something about it. In software engineering research the view taken of practitioners is another, the practices used in the industry is seen as immature and uncontrolled. It is possible to talk about ‘bad practice’. The goal of the research is often to find new methods, rules and perspectives to take on software engineering with the assumption that it is possible to change the practice to the better. This change perspective is a common theme in engineering, the belief that change is possible and that it is always possible to make things better and produce them in a more effective way. The software engineering focus on change and the possibility to talk about ‘bad practice’ is not easy to fit together with the ethnographic aim at description and emphasis on the knowledge of the observed persons.

A problem with the different views is that the software engineering community may dismiss ethnographic findings as simply being examples of ‘bad practice’. This is of course an oversimplification, in software engineering there has for some time been a discussion about the need to support any claims made by empirical studies. The normal way to do this is by quantitative approaches but using ethnography can be seen as a complementing approach.

I will exemplify some of these problems by comparing two different philosophical understandings of software development. Some ethnographic inspired studies will be used to compare the work practice of software developers with the different understandings. In the conclusion the methodological problems with the different perspectives is taken up again.

2 Two philosophical understandings of software development process

The software engineering community has for a long time been concerned about the differences between practice and theory. Most researchers acknowledge that the practices in the industry are not in agreement with the methods taught. This has been known for a long time. In the beginning the researchers believed that it depended on the field’s immaturity, but the discrepancy continues. This is a reason to start questioning the basic assumptions of the research. One example of this is 'Software Development as Reality Construction' [4], where Christiane Floyd suggest that we should look for another philosophical grounding of software engineering rather than the now dominating rationalistic tradition. She describes the basic assumption of the dominating view as:

- 'viewing software development as the *production* of software systems on fixed requirements'
- 'the separation of production from use and maintenance'
- 'the division of production into linear phases'
- 'the almost exclusive use of intermediate results in the form of documents'
- 'the view of methods as rules laying down standardized working procedures to be followed without reference to the situation in hand or the specific groups of people involved'
- 'the one-sided emphasis on formalization at the expansion of communication, learning and evolution'

The Cleanroom method [5], Hausler et al, is an example of the dominating view, here we also find one more basic assumption:

- Programs can be developed top-down as a hierarchy where each part can be written without concerns about the rest of the program.

Christiane Floyd continues by suggesting radical constructivism as a basic for an alternative view. This result in a different set of basic assumption:

- 'We do not analyze requirements; we construct them from our own perspective.'
- 'We do not apply predefined methods, but construct them to suit the situation at hand.'
- 'We don not refer to fixed means of implementation'

In another article [3] Christiane Floyd distinguish the following domains of discourse in software development:

- '*Requirements* are anchored in the world of user work processes.'
- '*System functions and architecture* arise as a result of design and draw on modeling concepts from the formal world of methods.'
- '*Programs* serve to control the computer and thus form the connection to the technical world of realization means.'

Contraries to the dominant position in software engineering she do not se the domains as having a temporal order but having a logical relationship. Software development is described as a basically cyclical activity.

3 Some ethnomethodologically inspired studies of software development.

I have not found many ethnographic inspired studies of actual software development, there are many studies in connection with requirements engineering and design but few of the actual software development. The studies used where originally done with other aims than what I will use them for here. To be able to use the studies to shed some light on the different philosophical understandings on software development I will reinterpret the studies for something else than what they where written for. This is my reinterpretation of the studies; the authors may not share my way of interpreting their results.

Button and Sharrock study the code writing practices in 'The Mundane Work of Writing and Reading Computer Programs'. [1] The main conclusion is that in software engineering it is accepted that 'the vulgar competencies of common-sense knowledge and practical reasoning' is essential for software development, at least in the sense that using tools like structured editors and pretty printing is essential. They contrast this with natural and social sciences where this goes unrecognized. In the same article they also make the observation that the software engineering methods partly have the goal to make the work visible by the demands on documentation and comments in the code. This is contrasted to mathematics where the end product is a proof with no trace of how the mathematician succeeded in finding the proof. The focus of the study is in the detailed practices when writing code and not in the overall development process. The practices they found has nevertheless implications on the validity of the basic understanding of software development. An example is the programmers' practice to sometimes use temporary names for variables (as 'temp-1') instead of immediately finding an intelligible name. The reason for this is that they are uncertain about how to name the variable until they have completed a larger part of the program and the role of the variable is clear. They will then change to temporary name into something intelligible. Button and Sharrock describe the activity of selecting informative names as that the programmer is engaged in constructing 'taxonomy of a semantic domain, for their practical and situated purposes'. My conclusion is that this concern contradicts the basic assumption for top-down programming, that each part can be written without concerns about the rest of the program.

In the article 'Here, there, and nowhere at all' [6], Susan Newman describes a study of the development of a middleware in a fortune 500 company. The actors in the design of this software is a divers and shifting group, both inside the company and in other companies (she lists 12 types of actors). These actors form an interconnected web that changes over time. The main problem taken up in the article is how to define 'the place' for the ethnographic study but this is not our concern here. Interesting is that the design is made by shifting groups of different actors, each with his own set of concerns. My interpretation of this is that the design process described is in no way a linear process starting with goals and requirements and ending with a complete design. Instead the design is done concurrently with the specification; an example is how the decision of what standards to be used is delayed.

> 'In the course of exploring this design proposal, Jerry and his colleagues discuss the problem of avoiding unnecessary commitments to some forms of standardization, as they might undermine the project of gaining commitments of third party developers to the particular standards deemed necessary for the middleware to succeed.' [6]

My interpretation is that an important design is delayed and that it is not possible to follow the linear model. It should also be stressed that the standard selected (or created) is a part of the requirements for the middleware under development.

In a study of software engineers maintaining a large telecommunication system ('An Examination of Software Engineering Work Practices' by Singer et. al. [7]) the focus is on what activities is most common. It turns out that searching and looking at source code was much more common than looking at the documentation. This is not at all according to the norms of the software engineering community where it is the documentation that should describe what the code does; looking at the code should only be done when the place to make the change is found. The authors coin the term 'Just in time comprehension of programs' for the work practice used. It is also interesting to note that when asked about what they do, reading documentation was the most frequent answer.

Large software development projects always have the problem of coordinating several groups of developers. In the article 'Talking Design' by Dittrich and Rönkkö [2] the problem of reaching a common understanding has been studied with ethnographic methods. The article takes up the problem that written documents do not necessarily mean the same thing to everybody. Their main example is a design discussion where it takes two hours before it becomes apparent that the designers have differences in how the design sketched on the whiteboard is to be understood. To reach a common understanding they use the whiteboard complemented with gestures to enact the actions of the components to finally reach a common understanding. The protocol made after the meeting includes the same picture as the whiteboard sketch and not much more. At the meeting they misunderstood this picture for one hour, the question is how it can work as documentation. The authors suggest that the necessary understanding had been communicated in the every day face to face talk. To support this suggestion they describe the problems in the same project with a subgroup working in another country. The subgroup is doing a well-defined part of the project but without the possibility of routine face to face meetings. This result in misunderstandings that harm the progress of the project before they are corrected. The experiences described do not fit with the view of written documentation as the exclusive means of transmitting intermediary results.

Suchman and Trigg in the article ‘Artificial intelligence as craftwork’ [8] concentrates on the use of whiteboard sketches. One conclusion is ‘… that the work of AI involves a series of transformations or re-representations…’ starting with observations of the social world and ending in executable code. In the article a design talk between two researchers is described, they use a whiteboard to sketch on and use gestures to show the dynamics. The figures on the whiteboard are seen as both representing a logical formalism and a Lisp construction. They don’t make a clear difference between these levels but make reference to the level most relevant for the moment. In a sense we can say that they follow the traditional waterfall model since they do not start writing the actual code before the formalism is elaborated. In another sense they completely break with the waterfall model since they let reasoning on the code level settle how the formalism should be.

4 Conclusions

The studies cited are few and do not cover the whole field of software development but they all support Floyd’s alternative view rather than the classical. Even if we assume that further studies would give the same result this is not enough to make a decision between the two perspectives. We can still interpret the empirical results as ‘bad practice’ rather than an indication of what software development really is about. There are other reasons to criticize the value of the studies. Some of the studies have been done with an ethnomethodological perspective (e.g. [1] Button and Sharrock) and their theoretical ground is close to Floyd’s. It is possible that this theoretical ground makes every ethnomethodological study biased towards finding support for Floyd’s view. In other studies the results can be seen as result of ‘bad practice’ (e.g. [7] about software maintenance). The work in artificial intelligence [8] is maybe not software development at all but rather scientific research. Even if my own understanding of the software development process is close to Floyd’s I fail to see that my interpretation of the studies really prove anything.

A possible way to get further would be to study a software project where there is a emphasis on following methods, examples of such exemplary projects exists, (e. g. the Cleanroom effort [5]). If the software developers work practices in such projects do not conform to the classical view there must be something wrong.

Several of the studies cited above has been done by social scientists and not by software engineers. It is possible that we can get a deeper understanding if the studies are done by researchers with a software engineering background that can make a deeper analyze of the results, trying to answer questions about why the practices are as they are. Such a deeper analysis is not in line with ethnography’s emphasis on the observed person’s own understanding of what they are doing.

References

1. Button, G., and Sharrock W. The mundane work of writing and reading computer programs. In P. ten Have, G. Psathas, eds., *Situated order: Studies in the social organization of talk and embodied activities*. Washington, D.C., University Press of America, 1995, pp 231-58
2. Dittrich, Y. and Rönkkö, K. Talking Design. In *Proceedings of IRIS 22*, 1999
3. Floyd, C., Reisin, F., and Schmidt, G. STEPS to Software Development with Users. In *Proceedings of the ESEC 1989*, Berlin.
4. Floyd, C. Software Development as Reality Construction, In *Software Development and Reality Construction*. Springer Verlag, 1992, Berlin.
5. Hausler et al, Adopting Cleanroom Software Engineering with a Phased Approach. In *IBM Systems Journal 1994/1.*
6. Newman, S. Here, there, and nowhere at all: Distribution, negotiation, and virtuality in postmodern ethnography and engineering. In *Knowledge and Society*, 11: 235-267.
7. Singer, J., Lethbridge, T. , Vinson, N. And Anquetil, N., 'An Examination of Software Engineering Work Practices', In *CASCON 1997.*
8. Suchman L. and Trigg R. Artificial intelligence as craftwork. In *Seth Chaiklin, Jean Lave, Understanding practice – perspectives on activity and context.* Cambridge University Press, NY, 1993.

Autobiographical note

Olle Lindeberg works as assistant professor in the area of software engineering. From the beginning a physic engineer I have been a software developer, mostly doing real-time embedded systems. Later I have been doing research in formal mathematical models of knowledge databases. At the University I teach in the interdisciplinary study program 'People, Computer, and Work' and in Software Engineering. My interest in using ethnographic methods is new and I am just starting up a research project in the area. Two other members in my research group, Yvonne Dittrich and Kari Rönkkö, have sent in position papers as well.