Information and Software Technology 70 (2016) 220–231

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

What does it mean to use a method? Towards a practice theory for software engineering

Yvonne Dittrich*

IT University of Copenhagen, Rued Langgaards Vej 7, DK-2300 Copenhagen S, Denmark

ARTICLE INFO

Article history:
Received 15 December 2014
Revised 1 July 2015
Accepted 3 July 2015
Available online 13 July 2015

Keywords:
Cooperative and human aspects of software engineering
Software engineering methods
Practice theory

ABSTRACT

Context: Methods and processes, along with the tools to support them, are at the heart of software engineering as a discipline. However, as we all know, that often the use of the same method neither impacts software projects in a comparable manner nor the software they result in. What is lacking is an understanding of how methods affect software development.
Objective: The article develops a set of concepts based on the practice-concept in philosophy of sociology as a base to describe software development as social practice, and develop an understanding of methods and their application that explains the heterogeneity in the outcome. Practice here is not understood as opposed to theory, but as a commonly agreed upon way of acting that is acknowledged by the team.
Method: The article applies concepts from philosophy of sociology and social theory to describe software development and develops the concepts of method and method usage. The results and steps in the philosophical argumentation are exemplified using published empirical research.
Results: The article develops a conceptual base for understanding software development as social and epistemic practices, and defines methods as practice patterns that need to be related to, and integrated in, an existing development practice. The application of a method is conceptualized as a development of practice. This practice is in certain aspects aligned with the description of the method, but a method always under-defines practice. The implication for research, industrial software development and teaching are indicated.
Conclusion: The theoretical/philosophical concepts allow the explaining of heterogeneity in application of software engineering methods in line with empirical research results.

1. Introduction

Methods and processes that rationalize and support the development of quality software are at the heart of software engineering as a discipline. And the process models, modelling approaches, architecture patterns and programming techniques, together with the tools to support them are widely used to improve the practice of software engineering. It is, however, difficult to pinpoint and quantify effects in practice: While effectiveness of specific techniques for isolated tasks (such as the effectiveness of reading vs. testing for finding bugs, respectively, different reading strategies) can be measured and compared [1], the comparison of project-level measurements, based even on a population of similar projects, is still not possible [2].

Qualitative empirical research indicates that software teams balance what is recommended by the method with the specific technical and organizational circumstances of the project. Button and Sharrock, for example, report a specific interpretation of Yourdon development methodology, CASE and C in reaction to the specific contingencies [5]. They argue that 'methods are *worked at* phenomena, that they are made to work in the circumstances of their deployment and that the details of that work are part and parcel of the development process' [5, p. 237, highlighting as in the original]. Early on, software engineers recognized that tools supporting software processes need to support exceptions and adaptations to allow developers to react to situated contingencies [4]. Martin et al. report a software team balancing optimal test design, computer resources and people, and organizational circumstances when testing software [6]. In some cases, the applicability of a method depends on how the business area is organized [7]. The 'work arounds' when applying SCRUM [51] have in the agile community gotten an own the name even: 'scrumbut' [8]. Based on surveys and interviews, Fitzgerald concludes that only 6% of all practitioners apply a formally defined method [55].

* Tel.: +45 7218 5177.
E-mail address: ydi@itu.dk

http://dx.doi.org/10.1016/j.infsof.2015.07.001
0950-5849/

So what is responsible for the difference between the method as described and the method in use? Researchers take varying positions. Some argue that software practitioners are not educated and trained well enough, as e.g., Parnas takes this view in his debate article on empirical research, which acknowledges the situated rationale of observed non-compliance with methods and disciplinary norms [9]. In the article 'The rational Design Process – Why and how to Fake it' [53], Parnas and Clements acknowledge the impossibility of following a 'rational design process.' The authors, however, recommend following it as closely as possible and complementing the documentation with additional information regarding the design decision taken.

A second position is to argue that the methods do not fit the situated contingencies, as especially authors inspired by Computer Supported Cooperative Work (CSCW) propose; thus, practitioners need to work around what the method prescribes in order to get their tasks done [6,7,10]. Fitzgerald provides a more comprehensive and differentiated appraisal of both positions in [54].

A third position proposes that we might need to revise our understanding of methods and practice. Button and Sharrock, e.g., conclude their discussions with: "If we think that methods are procedural recipes to follow we might think that all we have to do is to develop or alight upon the best method for our purposes and our problem will be solved by cranking the methodological handle. … If instead of thinking of methods as procedural recipes to be used in the course of development, we think of them as tools in the organization of development, then the artful and contingent *use* of those tools is as important as the character of the tools themselves" [5, p. 237]. Arguing along a similar line, Fitzgerald et al. distinguishes between formalized methods and methods-in-action, and propose several levels of tailoring and appropriation of methods [56].

This article aims at taking the discussion a step further: It argues that we need to develop a theoretical base for understanding software development as a social practice in order to understand how methods and tools are appropriated in everyday software development. In other words, what is needed is a practice theory of software engineering. The purpose of such a theory should be to help explain the phenomena that we observe in empirical research in software engineering. The goal is to be able to address the question indicated in the title of this article "What does it mean to use a method?" not only empirically but also based on a set of concepts that allows the explaining of the observed phenomena.

In their article 'Theorizing about Software Development Practice,' Pävärinta and Smolander [11] propose developing a theory of software development practice based on empirical research. This current article proposes a specific way to conceptualize practices, their rationales and their relation to contextual contingencies. The aim is to encourage the discussion of software engineering method and theory (SEMAT) [52] that includes a theoretical underpinning of the social side of software engineering.

To develop a practice concept, the current article appropriates concepts from social theory. I use Schatzki's concept of integrated practice to describe software development as shared social practices based on common understandings, rules and teleoaffective structures [12]. Based on Knorr Cetina's concept, I argue that software development is an epistemic practice, one that unfolds its object as the team proceeds in the development [14]. Based on this foundation and referring to software engineering discussions of the character of methods, I define methods as practice patterns, explicitly formulated sets of (tool supported) understandings, rules and teleoaffective structures that need to be integrated in existing practices.

The genre of this article is thus philosophical argumentation: it develops concepts based on literature and shows that these concepts can be used to better explain empirical results. Examples of such argumentations are the articles by Knorr Cetina [13] and Schmidt [15] that are further discussed below. The quality criteria for philosophical argumentation are subject to philosophical subdisciplines and in part also depending on the philosophical school the argument is contributing to. In the context of Software engineering, I propose to apply (a) rigour of argumentation and (b) relevance of results; for example, the theory should render results of empirical research as examples of the theoretical concepts and relations rather than idiosyncratic behavior. The article requires its reader to adjust to an unusual style of argumentation. Philosophical texts tend to use longer citations in order to show how the original author defines a concept before it is adapted and applied in the new context. These citations are used as inline citations rather than formatted as an own paragraph.

The article is structured as follows: Section 2 introduces the Software Engineering discussion on methods and their usage. The discussion leads to an understanding of software development as a social practice. Section 3 provides a contextualizes the philosophical approaches used with respect to philosophy of sociology and CSCW. Section 4 'A practice concept for software engineering' develops the conceptual base; the concepts of social practice and epistemic practice are presented and software engineering is described as epistemic practice. Based on these concepts, Section 5 'Methods and method usage' then addresses the research question regarding the usage of methods requiring integration in, and adaptation to, an existing practice with its specific setting and purpose, and further addresses the necessary substantial explicit adaptation of practice, which extends Schatzki's or Knorr Cetina's work. Section 6 'Practices are constantly maintained and developed' further explores the continuous adaptation and maintenance of practice referring to the concept of articulation work by Strauss. The results are summed up and discussed in Section 7, and implications are proposed for research, industrial practice and teaching. Section 8 summarises the conclusions.

2. Methods and practice in software engineering

The development and dissemination of methods in order to inform and improve practice are at the heart of software engineering. Nevertheless, surprisingly few researchers have discussed the character of methods and how they inform software development. This section begins with a discussion of methods in software engineering and then argues for a concept of software development as social practice in order to inform the development and usage of methods.

2.1. Methods in software engineering

In ordinary language, the term 'method' describes a systematic way of addressing an endeavor. For example, Webster's dictionary defines method as the following

> **method**
> 1: a procedure or process for attaining an object: as
> *a (1)*: a systematic procedure, technique, or mode of inquiry employed by or proper to a particular discipline or art *(2)*: a systematic plan followed in presenting material for instruction
> *b (1)*: a way, technique, or process of or for doing something *(2)*: a body of skills or techniques
> 2: a discipline that deals with the principles and techniques of scientific inquiry
> 3: *a*: orderly arrangement, development, or classification: plan *b*: the habitual practice of orderliness and regularity
> …
>
> Webster's Dictionary [30]