Contents lists available at ScienceDirect

Information and Software Technology

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

# Software engineering process theory: A multi-method comparison of Sensemaking–Coevolution–Implementation Theory and Function–Behavior–Structure Theory

Paul Ralph

*Department of Computer Science, University of Auckland, Auckland 1142, New Zealand*

ARTICLE INFO

*Article history:*
Received 27 November 2014
Revised 29 June 2015
Accepted 29 June 2015
Available online 15 July 2015

*Keywords:*
Process theory
Software process
Case study
Questionnaire

ABSTRACT

*Context:* Software engineering has experienced increased calls for attention to theory, including process theory and general theory. However, few process theories or potential general theories have been proposed and little empirical evaluation has been attempted.

*Objective:* The purpose of this paper is to empirically evaluate two previously untested software development process theories – Sensemaking–Coevolution–Implementation Theory (SCI) and the Function–Behavior–Structure Framework (FBS).

*Method:* A survey of more than 1300 software developers is combined with four longitudinal, positivist case studies to achieve a simultaneously broad and deep empirical evaluation. Instrument development, statistical analysis of questionnaire data, case data analysis using a closed-ended, a priori coding scheme and data triangulation are described.

*Results:* Case data analysis strongly supports SCI, as does analysis of questionnaire response distributions ($p < 0.001$; chi-square goodness of fit test). Furthermore, case-questionnaire triangulation found no evidence that support for SCI varied by participants' gender, education, experience, nationality or the size or nature of their projects.

*Conclusions:* SCI is supported. No evidence of an FBS subculture was found. This suggests that instead of iterating between weakly-coupled phases (analysis, design, coding, testing), it is more accurate and useful to conceptualize development as ad hoc oscillation between making sense of the project context (Sensemaking), simultaneously improving mental representations of the context and design space (Coevolution) and constructing, debugging and deploying software artifacts (Implementation).

## 1. Introduction

The Software Engineering (SE) field has witnessed increasing calls to theorize about its core concepts and processes (e.g. [1–7]). However, SE remains preoccupied with normative research on software development methods, methodologies and process models [8] and characterized by "a lack of interest in theories aimed at *understanding* and *explaining* the how and why of the observed design activities" in favor of "a rush from observation and description to prescriptive modeling and the construction of design tools" [9].

Building and empirically evaluating SE theories has many benefits. Theories synthesize, preserve and communicate empirical knowledge, thereby implicitly coordinating future inquiry. Unlike method and tool knowledge, theories withstand fashions and fads. Adopting a theoretical mindset furthermore implicitly refocuses researchers on fundamental rather than superficial features of SE.

A theory is simply a collection of interconnected concepts. Theories have differing purposes including *to describe*, *to explain*, *to analyze* and *to predict* [10] and units of analysis including *individual*, *group*, *process*, *organization* and *industry* [11]. Middle range theories apply to specific empirical phenomena while general theories apply to a broad class of phenomena. Variance theories focus on *why* events occur while process theories focus on *how* events occur [12]. Variance theories employ different approaches to causation including *regularity* (Y always follows X), *counterfactual* (Y cannot occur without X), *probabilistic* (Y is more likely given X), and *teleological* (X, an agent with free will, chooses to do Y) [13]. Meanwhile, process theories may approximate one of several "ideal types" – *lifecycle* (a sequence of phases), *evolution* (a population of competing, reproducing elements), *dialectic* (struggle between several entities with varying power) and *teleological* (goal-oriented,

*E-mail address:* paul@paulralph.name

http://dx.doi.org/10.1016/j.infsof.2015.06.010
0950-5849/© 2015 Elsevier B.V. All rights reserved.

self-directed actions of an autonomous agent) [14]. Given the diversity of possible theoretical approaches, deeply understanding sociotechnical phenomena including software development necessitates numerous theoretical perspectives [15,16].

Following Brooks' [17] insightful elucidation of the fundamental confusion surrounding the software development process, this paper focuses on software development process theory. Specifically, it aims to empirically evaluate Sensemaking–Coevolution–Implementation Theory (SCI), which diverges from traditional engineering thinking to explain more accurately how software is developed in practice [18]. SCI is evaluated against a rival theory, The Function–Behavior–Structure Framework (FBS), which expresses a more traditional view of the development process [19,20]. The paper presents an extensive, multi-method, empirical initiative to evaluate these two theories, driven by the following research question.

***Research Question:*** *Does Sensemaking-Coevolution-Implementation Theory better explain how teams develop complex software systems in practice than The Function-Behavior-Structure Framework?*

Here a *complex* system is a collection of interconnected elements that exhibits behaviors not predictable from those elements [21]. Focusing on complex systems excludes routine re-implementation of well-understood artifacts. Meanwhile, software development here "encompasses all the activities involved in conceptualizing, framing, implementing, commissioning, and ultimately modifying complex systems" [22]. This paper furthermore focuses on development by individuals or coordinated teams predominately working together, rather than projects involving mass-collaboration, hostile teams working at cross purposes or multiple autonomous teams. Additionally, it primarily concerns direct actions of development teams, rather than indirect actions and related concepts including project management, politics, power and time.

Section 2 discusses process theory in SE, including detailed presentations of FBS and SCI. Section 3 presents the multi-methodological research design. Section 4 summarizes the results and Section 5 discusses the study's limitations and implications. Related research is discussed throughout.

## 2. Software engineering process theories

While a comprehensive review of theories used in SE is beyond the scope of this paper, Hannay et al. [4] identified 40 theories that were experimentally evaluated in studies published between 1993 and 2002. However, only two of these were used in more than one article: (1) the Theory of Cognitive Fit, which posits that the alignment between a task and the presentation of information needed for the task affects task performance [23,24], and (2) the theory that reading techniques affect software inspection effectiveness [25–27].

In the following decade, empirical research continued gaining prominence in SE, with, for example, the ISESE and METRICS symposia, followed by the Empirical Software Engineering and Measurement conference. However, most empirical work in SE continues either to evaluate specific tools and techniques (e.g. bug prediction approaches [28]) or to investigate specific SE phenomena (e.g. source code clone maintenance [29]). Similarly, most SE theories concern specific SE activities (e.g. search-based testing [30]; visual notation [31]). Meanwhile, little theoretical and empirical work investigates the software development process comprehensively. Software process research is predominately prescriptive and method-focused [8]. This has produced more than one thousand software development methods [32], some of which (e.g. the Waterfall Model [33], Spiral Model [34], Axiomatic Design [35]) are sporadically treated as theories. For example, statements like "in conventional software development, the development lifecycle in its most generic form comprises four broad phases: planning, analysis, design, and implementation" [36], treat Waterfall as a theory.

However, methods are not appropriate foundations for process theories [9]. Methods prescribe ostensibly good approaches to an activity. Process theories, in contrast, encompass both good and bad approaches by explaining the fundamental properties of an activity. Therefore, this section focuses on process theories, not methods or prescriptions.

A recent review [37] found no software development process theories other than SCI. However, it found four process theories that *could* apply to software – The Basic Design Cycle [38], The Problem-Design Exploration Model [39], The Self-Conscious Process [40] and The Function–Behavior–Structure Framework (FBS) [19]. Adopting one of these as a rival theory facilitates a more rigorous evaluation (see below). The Problem-Design Exploration Model is a poor choice as it is intended to explain design using genetic algorithms while SCI is intended to explain how human teams develop software. As SCI is partially based on The Selfconscious Process, the latter would provide too weak a rival. The Basic Design Cycle, meanwhile, is a special case of FBS, so FBS is a stronger rival due to its greater explanatory power. Moreover, FBS has been conceptually – although not empirically – applied to SE [41,42]. Consequently, FBS is the best choice for rival theory. This section therefore reviews and conceptually evaluates SCI and FBS.

### *2.1. Sensemaking–Coevolution–Implementation Theory*

SCI (Fig. 1; Table 1) posits that complex software systems are produced by an agent (individual or team) that alternates between *sensemaking*, *coevolution* and *implementation* in a self-determined sequence [18].

Sensemaking refers to perceiving, assigning meaning to and organizing beliefs about a phenomenon or experience [43–45]. In SE, then, it may include interviewing stakeholders, writing notes, organizing notes, reading about the domain, reading about technologies that could be used in the project and sharing insights among team members. Sensemaking also includes problem framing [46] and the kinds of testing that are closely related to understanding the context (e.g. acceptance testing, usability studies).

Coevolution refers to a situation where two or more interconnected objects develop and change over time such that changes in one trigger changes in the other and vice versa. Meanwhile, a *schema* is "a mental representation of some aspect of experience, based on prior experience and memory, structured in such a way as to facilitate (and sometimes to distort) perception, cognition, the drawing of inferences, or the interpretation of new information in terms of existing knowledge" [47]. In SCI, and the interdisciplinary design literature more generally, Coevolution refers to mutually exploring and refining the design agent's schemas of the project context and design space [48–50]. Coevolution may occur in planning meetings and design meetings, following breakdowns or during an individual's internal reflection. It is especially evident when a team stands around a whiteboard drawing informal models and discussing how to proceed.

For example, suppose a professor gives regular quizzes to evaluate students (design schema), but students complain that feedback on quizzes is taking too long (context schema). The designer imagines giving online quizzes with automated marking and instant feedback (design schema). However, the professor complains that online quizzes are too difficult to police for formal evaluation. They quickly realize that online quizzes are better for encouraging preparation (context schema). This suggests numerous changes to quiz structure, such as giving students multiple