Sensemaking-Coevolution-Implementation Theory

A Model of the Software Engineering Process in Practice

Paul Ralph
Department of Management Science
Lancaster University
Lancaster, UK
paul@paulralph.name

***Abstract*—Sensemaking-Coevolution-Implementation Theory is a teleological process theory of the practice of designing complex software systems. It posits that an independent agent (design team) creates a software system by alternating between its three titular activities. Its veracity has been demonstrated using questionnaire and case-study methods. It has been used to evaluate software engineering curricula and highlight deficiencies in software engineering methods and practices.**

Keywords-SCI Theory; process theory; design; coevolution

I. SCI THEORY

Theories of the software engineering (SE) process have historically been dominated by stage-gate or lifecycle models, beginning with the Waterfall Model [1]. This was followed by a "methodology era", during which SE was usually conceptualized through a methods lens, and a "post-methodology era" where methods continued to dominate conceptualization of SE despite their decreasing relevance to practice [2]. These lifecycle models and the methods based on them are fundamentally misleading due to their empirically debunked assumptions [3], [4].

Sensemaking-Coevolution-Implementation Theory (SCI) was developed as an alternative to lifecycle models of SE [5]. It is based on Alexander's model of the "selfconscious" design process [6], reflection-in-action [7], and theorizing of coevolution by [8] among others. SCI (Figure 1, Table 1) posits that where a complex software system is developed by an independent, goal-oriented agent, that agent will engage in three basic processes – Sensemaking, Coevolution and Implementation – in a self-directed sequence.

The agent may be an individual or team. The arrows in Figure 1 indicate relationships between concepts and activities, not sequence – the agent may transition between activities in any order. In a typical project, Sensemaking may include interviewing stakeholders, writing notes, organizing notes, reading about the domain, reading about technologies that could be used in project, sharing insights among team members and acceptance testing (getting feedback from stakeholders on prototypes). Implementation may include coding, managing the codebase, writing documentation, automated testing, creating unit tests, running unit tests and debugging.

While Coevolution does not directly map to a variety of well-known software engineering activities, it is observable in real projects. For example, when a team stands around a whiteboard drawing informal models and discussing how to proceed, they often oscillate between ideas about the design object (e.g., 'how should we distribute features between the partner channel screen and the partner program screen?') and the context (e.g., 'you know what, I think channels and programs are just different names for the same thing.'). This mutual exploration of context and design object is Coevolution. Coevolution may occur in planning meetings and design meetings, following breakdowns or during an individual's internal reflection.

Evolution and coevolution are easily confused. In design literature, evolution, specifically evolutionary prototyping, denotes the gradual improvement of a software object. In contrast, coevolution refers to "developing and refining together both the formulation of a problem and ideas for a solution, with constant iteration of analysis, synthesis and evaluation processes between the … problem space and solution space" {Dorst:2001tq, p. 434}. SCI therefore distinguishes between two types of iteration – coevolution denotes simultaneously revising ideas of problem and solution within minutes or hours, while evolution denotes improving software artifacts over weeks and months.

SCI is a teleological process theory, intended to explain how software is developed in practice. Van de Ven [9] distinguishes two types of theories – variance theories explain the causes of consequences of something and often specify the relative contribution of multiple antecedents, while process theories explain *how and why* an entity changes and develops. Process theories come in at least four types [10]: lifecycle theories posit that an entity progresses through a series of stages in a predefined sequence; evolutionary theories posit a population of entities that changes as less fit entities expire and remaining entities change and recombine; dialectic theories posit that changes result from shifts in power among conflicting entities; teleological theories posit an agent who purposefully selects and takes actions to achieve a goal. SCI therefore takes a teleological approach to causality: software artifacts change as human beings (having free will) choose to change them. This differs from the probabilistic approach to causality adopted by many variance theories.

A survey [11] of over 1300 software development professionals found that SCI better described their processes than either Waterfall or an alternative SE process theory, the Function-Behavior-Structure Framework (FBS) [12]. Emerging evidence from an ethnographic study of an English software development team also supports SCI's core claims and the impossibility of understanding conventional SE through Waterfall or FBS. SCI has been used to analyze SE curricula [13]. It can also be used to analyze design

methods and practices, and to teach SE and project management.

REFERENCES

[1] W. Royce, "Managing the development of large software systems," presented at the Proceedings of WESCON, the Western Electronic Show and Convention, Los Angeles, USA, 1970.

[2] D. Avison and G. Fitzgerald, "Where Now for Development Methodologies," *Communications of the ACM*, vol. 46, no. 1, pp. 79–82, 2003.

[3] F. P. Brooks, *The Design of Design: Essays from a Computer Scientist*. Addison-Wesley Professional, 2010.

[4] P. Ralph, "Introducing an Empirical Model of Design," in Proceedings of The 6th Mediterranean Conference on Information Systems, Limassol, Cyprus, 2011.

[5] P. Ralph, "The Sensemaking-Coevolution-Implementation Theory of Software Design," *MIS Quarterly,* under review.

[6] C. W. Alexander, *Notes on the synthesis of form*. Harvard University Press, 1964.

[7] D. A. Schön, *The reflective practitioner: how professionals think in action*. USA: Basic Books, 1983.

[8] N. Cross, "Research in Design Thinking," in *Research in design thinking*, N. Cross, K. Dorst, and N. Roozenburg, Eds. Delft, Netherlands: Delft University Press, 1992.

[9] A. H. Van de Ven, *Engaged scholarship: a guide for organizational and social research*. Oxford, UK: Oxford University Press, 2007.

[10] A. H. Van de Ven and M. S. Poole, "Explaining development and change in organizations," *The Academy of Management Review*, vol. 20, no. 3, pp. 510–540, Jul. 1995.

[11] P. Ralph, "Comparing Two Software Design Process Theories," in Proceedings of the Fifth International Design Science Research in Information Systems and Technology Conference, St. Gallen, Switzerland, 2010, vol. 6105, pp. 139–153.

[12] J. S. Gero and U. Kannengiesser, "An ontological model of emergent design in software engineering," presented at the 16th International Conference on Engineering Design, Paris, France, 2007.

[13] P. Ralph, "Improving coverage of design in information systems education," in Proceedings of the 2012 International Conference on Information Systems, Orlando, FL, USA, 2012.

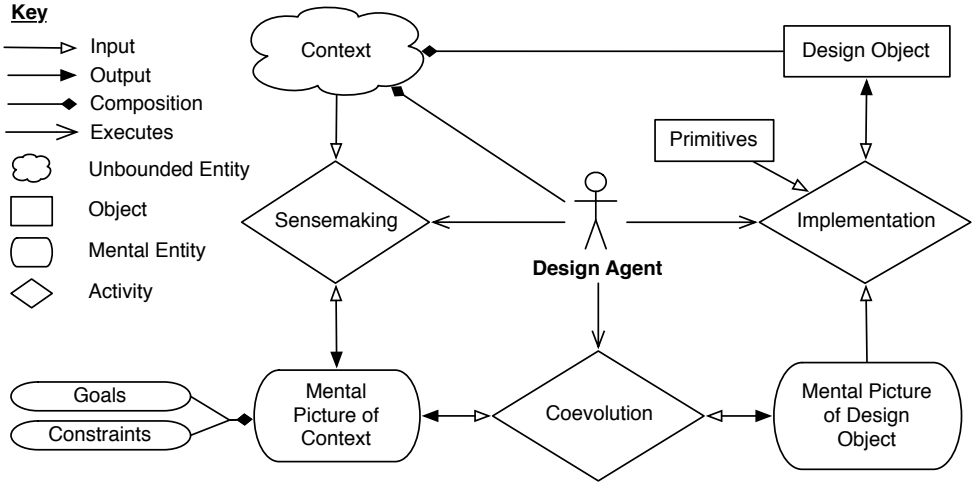

Figure 1. Example of a TWO-COLUMN figure caption: (a) this is the format for referencing parts of a figure.

Concept / Activity	Meaning
Constraints	the set of restrictions on the design object's properties
Design Agent	an entity or group of entities capable of forming intentions and goals and taking actions to achieve those goals and that specifies the structural properties of the design object
Context	the totality of the surroundings of the design object and agent, including the object's intended domain of deployment
Design Object	the thing being designed
Goals	optative statements about the effects the design object should have on its environment
Mental Picture of Context	the collection of all of the design agent's beliefs about its and the design object's environments
Mental Picture of Design Object	the collection of all of the design agent's beliefs about the design object
Primitives	the set of entities from which the design object may be composed
Sensemaking	the process where the design agent organizes and assigns meaning to its perception of the context, creating and refining the mental picture of context
Coevolution	the process where the design agent simultaneously refines its mental picture of the design object, based on its mental picture of context, and the inverse
Implementation	the process where the design agent generates or updates the design object using its mental picture of the design object

TABLE I. CONCEPTS AND RELATIONSHIPS OF SCI THEORY, DEFINED