Editor: **Philippe Kruchten**
University of British Columbia
pbk@ece.ubc.ca

# Where's the Theory for Software Engineering?

Pontus Johnson, Mathias Ekstedt, and Ivar Jacobson

**MOST ACADEMIC DISCIPLINES** are very concerned with their theories. Standard textbooks in subjects ranging from optics to circuit theory to psychology to organizational theory to international relations either present one single theory as the subject's core or discuss a limited set of alternative theories to explain the discipline's essence to its students. A prime example is the central role of Maxwell's equations in the subject of electrical engineering. It's difficult to fathom what electrical engineering would be today without those four concise equations. A quite different example is the contested Domino theory, which heavily influenced American foreign policy in the 1950s to 1980s by speculating that one nation's embrace of communism would entail the conversion of surrounding countries in a domino effect. Even though electrical engineering and political science are different in almost all respects, they're both highly interested and invested in their theories.

## What Is Software Engineering Theory?

Software engineering, however, isn't overly concerned with its core theory. If asked, the community surely couldn't give a coherent answer about which is the most important one. Candidates might include theories with significant scope, such as formal systems theory, decision theory, organization theory, or theory of cognition. Collections of propositions might also be suggested, such as Alan Davis's *201 Principles of Software Development* (McGraw-Hill, 1995), Frederick P. Brooks's propositions in *The Mythical Man-Month* (Addison-Wesley, 1975), or *SWEBOK* (A. Abran et al., eds., IEEE, 2004). Specialized models such as Cocomo might also be candidates. We suspect that you'll disagree with most of these proposals, but that just proves our point about the lack of consensus. Still, why are so many other fields explicit with their theories while software engineering is not?

Before discussing this question, we need to—in impossibly few words—describe what we mean by that multifaceted word "theory." A good definition comes from a thoughtful article published in *Management Information Systems Quarterly* (S. Gregor, "The Nature of Theory in Information Systems," vol. 30, no. 3, 2006). According to author Shirley Gregor, there are many definitions for the term, but most theories share three characteristics: they attempt to generalize local observations and data into more abstract and universal knowledge; they typically represent causality (cause and effect); and they typically aim to explain or predict a phenomenon. Considering the purpose of theory, Gregor proposes four goals. The first is to simply describe the studied phenomenon; *SWEBOK* could serve as an example. The second goal is to explain the how, why, and when of the topic; theory of cognition, for example, is aimed at explaining the workings of the human mind. The third goal is to not only explain what has already happened but also to predict what will happen next; in software engineering, Cocomo attempts to predict the cost of

*continued on p. 94*



0740-7459/12/$31.00 © 2012 IEEE

*continued from p. 96*

software projects. The fourth goal of theory according to Gregor is to prescribe how to act based on predictions; Davis's 201 principles exemplify prescriptions.

## Three Arguments

Returning to the main question—why the software engineering community seems so uninterested in discussing its theories—we can imagine three arguments: software engineering doesn't need theory, software engineering already has all the theory it needs, and software engineering can't have any significant, defining theories. We don't believe that these arguments are valid, but let's consider them individually.

### Software Engineering Doesn't Need Theory

Software engineering is doing fine without explicit theories, so why change a winning formula? First, software engineering *isn't* doing fine. Reports about failed IT projects have been published on a regular basis for decades now. Second, all engineering fields need theory. To build something good, you must understand the how, why, and when of building materials and structures. Indeed, you have to predict in the design stage the qualities of the end product if you want to avoid the painstaking labor of trial and error. In the words of Kurt Lewin, "There is nothing so practical as a good theory" (*Field Theory in Social Science*, Harper & Row, 1951). Third, for the many software engineering researchers employed at universities around the world, a researcher without a theory is like a gardener without a garden. According to philosopher Thomas Kuhn, the maturity of scientific disciplines can be measured by the unity of their theories (*The Structure of Scientific Revolutions*, Univ. of Chicago Press, 1962). In the most established disciplines engaged in what Kuhn calls normal science, a paradigmatic theory defines a whole field (for example, Maxwell's equations, Einstein's theory of relativity, and Darwin's theory of natural selection). In a less mature phase, called pre-paradigm, a small number of theories, typically with ambitious explanatory scopes, compete for academic hegemony. This is the case in psychology, where cognitive theories challenge psychodynamic theories, and in international relations, where realist and liberalist theories battle for dominance. Kuhn doesn't offer a name for the phase before the pre-paradigm, in which there exists a large number of unrelated theories, because he considers this something less than science.

> To build something good, you must understand the how, why, and when of building materials and structures.

### Software Engineering Already Has Its Theory

A discipline's significant theories should be able to provide answers to that discipline's significant questions. Considering software engineering, one of the most hotly debated questions concerns the choice of software development method. Although there are many opinions on the subject, we can name very few theories that attempt to answer the question. And to the extent that such theories exist, they aren't, as in other disciplines, given names, presented in textbooks, or debated at conferences. The same goes for other significant questions of software engineering, such as which programming language to use, how to specify system requirements, and so on. Note that many proposed software development methods, programming languages, and requirements specification languages exist, but very few explicit theories explain why or predict that one method or language would be preferable to another under given conditions.

### Software Engineering Can't Have a Theory

Software engineering is a practical engineering discipline without scientific ambitions where rules of thumb and guidelines assume the role of theory. We can counter this argument by reiterating the tight connection between engineering and science. A typical definition of engineering is the one found in *Encyclopedia Britannica*: "the application of science to … the uses of humankind." Thus, there's no engineering without science. Second, it isn't true that there is no theory in the software engineering community. In a sense, theory is abundant. To the previously mentioned propositions, we could add Kent Beck's suggestion that the change cost curve could be logarithmic rather than exponential (*Extreme Programming Explained*, Addison-Wesley, 1999), David Parnas's principle of information hiding ("On the Criteria to Be Used in Decomposing Systems Into Modules," *Comm. ACM*, 1972), Conway's law, Edsger Dijkstra's theory of cognitive limits as presented in the classical article "Go To Statement Considered Harmful" (*Comm. ACM*, 1968), stepwise refinement, and so on. But all of these theories are small and most

are casual, proposed by the authors but rarely subjected to extended studies, and they explain only a limited set of phenomena. Furthermore, most of these theories aren't subject to serious academic discussion; they aren't evaluated or compared with respect to traditional criteria of theoretical quality such as consistency, correctness, comprehensiveness, and precision.

As should be evident by now, we don't believe that there's any rational reason for the lack of theoretical interest in software engineering. It's surely historical; born in the hurly burly of software practice, explanation and prediction were often merely glanced at through the car window in the race between problem and solution. Today, however, tens of thousands of software engineering researchers are employed in the universities of the world, spending innumerable manhours on software engineering research, but theory is still on the sidelines. To our knowledge, very few explicit attempts propose general theories of software engineering. Interesting avenues to watch on that front seem to be the Semat initiative (www.semat.org) and GUTSE (Grand Unified Theory of Software Engineering; http://books.google.com/books?id=TLcceL3NEiMC).

And make no mistake, theory is important. Without the predictive and prescriptive support of theory, software engineering would be relegated to the horribly costly design process of trial and error. With theory, we rise from the drudgery of random action into the sphere of intentional design. Software engineering is already full of implicit theory. We just need to bring it out into the open and subject it to the serious scientific treatment it deserves.

**PONTUS JOHNSON** is a professor at KTH Royal Institute of Technology in Stockholm, Sweden. Contact him at pontusj@ics.kth.se.

**MATHIAS EKSTEDT** is an associate professor at KTH Royal Institute of Technology in Stockholm, Sweden. Contact him at mathias.ekstedt@ics.kth.se.

**IVAR JACOBSON** is an international honorary advisor at Peking University, Beijing, and he holds an honorary doctorate degree from San Martin de Porres University, Peru. He is most well known for his seminal contributions to the domain of software engineering, including use cases, UML, and now as a founder of Semat. Contact him at ivar@ivarjacobson.com.



*IEEE Software* (ISSN 0740-7459) is published bimonthly by the IEEE Computer Society. IEEE headquarters: Three Park Ave., 17th Floor, New York, NY 10016-5997. IEEE Computer Society Publications Office: 10662 Los Vaqueros Cir., Los Alamitos, CA 90720-1314; +1 714 821 8380; fax +1 714 821 4010. IEEE Computer Society headquarters: 2001 L St., Ste. 700, Washington, DC 20036. Subscription rates: IEEE Computer Society members get the lowest rate of US$56 per year, which includes printed issues plus online access to all issues published since 1984. Go to www.computer.org/subscribe to order and for more information on other subscription prices. Back issues: $20 for members, $193 for nonmembers (plus shipping and handling).

**Postmaster:** Send undelivered copies and address changes to *IEEE Software*, Membership Processing Dept., IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08854-4141. Periodicals Postage Paid at New York, NY, and at additional mailing offices. Canadian GST #125634188. Canada Post Publications Mail Agreement Number 40013885. Return undeliverable Canadian addresses to PO Box 122, Niagara Falls, ON L2E 6S8, Canada. Printed in the USA.