Towards Middle-Range Usable Design Theories for Software Engineering

Roel J. Wieringa
University of Twente
The Netherlands
r.j.wieringa@utwente.nl

ABSTRACT

In this position paper we argue argue to reduce the ambition for a general theory of software engineering to that of a patchwork of middle-range theories about how artifacts can be used by practitioners. This proposal is placed in the wider context of the role of theory in the engineering sciences. A few suggestions are given for how to proceed along this way.

Categories and Subject Descriptors

D.2.10 [**Software**]: Software Engineering—*Design*

General Terms

Theory

Keywords

Middle-range theory, design theory, usable theory

1. INTRODUCTION

In their call for more theory in software engineering, Johnson et al. [11] mention a few candidate theories, and immediately add that readers may disagree with these proposals. This, they point out, is exactly the problem: Why is there such a lack of consensus about theories in software engineering? And why are so many other fields explicit in their theories while software engineering is not?

These two questions point to some additional underlying problems. First of all, can scientific theories be the subject of consensus? There might be a lack of consensus about my theory why the Dutch lost the last World Championship (i.e. too many prima donnas, no team play). Some people might agree, others might disagree. But is it relevant whether there is a consensus about theories in mechanical engineering? The truth of scientific theories is supposed to be independent from our opinion but dependent on the test of practice. But perhaps the problem is a lack of consensus in software engineering about what theories are.

GTSE'14, June 2, 2014, Hyderabad, India
ACM 978-1-4503-2850-0/14/06
http://dx.doi.org/10.1145/2593752.2593753

A related problem is whether we should aim at *general* theories. In other fields, very few papers, if any, claim general theories. And often, the class of phenomena generalized about is a fuzzy set. In the words of the philosopher of science Cartwright, we live in a dappled world.

> The laws that describe this world are a patchwork, not a pyramid. They do not take after the simple, elegant and abstract structure of a system of axioms and theorems. ... The dappled world is what, for the most part, comes naturally; regimented behaviour results from good engineering. [4, page 1]

As argued below, universality comes at the cost of idealization, which makes theories unusable in practice. If generality means universality, then generality should not be our goal.

But if we drop the goal of universality, the alternative is not to have no theory at all. We cannot restrict ourselves to collecting empirical data without providing a theoretical explanation. We should still provide theoretical explanations of empirical data, without claiming universality for the results. As it turns out,we already have a patchwork of theories, compiled in such books as Davis' *201 Principles of Software Development* [8] and Endres & Rombach's *A Handbook of Software and Systems Engineering* [9] as middle-range theories. One of the recommendations of this paper is to reconceptualize these partial theories and principles as design theories, and to develop these theories further in evaluation and validation research.

In this opinion paper I propose a more realistic concept of scientific theory, characterized by three elements: The theories that we look for are middle-range in scope, they are about artifacts in a context, and they must be usable by practitioners. I briefly discuss what makes a theory scientific (Section 2), why we need patchworks of theories (Section 3), why we need design theories (Section 4) and what it means for them to be usable (Section 5). The final section draws some conclusions.

2. WHAT IS A SCIENTIFIC THEORY?

The briefest definition of the concept of a *theory* is that it is a belief that there is a pattern in phenomena [6, page 55]. After adoption of a theory, what appeared to be a chaotic bundle of of phenomena turns out to exhibit a pattern. Kaplan [14, page 295] contrasts theory with the notion of unreflective habit. Habit exploits regularities in phenomena but never pauses to reflect and identify the underlying pattern.

This includes all kinds of theories, including conspiracy

theories about the causes of the credit crisis, economic theories about the causes of the same crisis, astrology, the theory of classical mechanics, and string theory. What makes a theory scientific?

Briefly, a theory is scientific if it has been submitted to, and survived, two kinds of tests [5, 25]:

- *Empirical testing.* the theory has been submitted to, and survived, tests against experience. This can be done in observational research or in experimental research.

- *Justification to a critical peer group.* The theory has been submitted to, and survived, criticism by competent and critical peers. This is organized in a peer review system, in which the author of a theory submits it to a peer-reviewed publication process, and the quality of the justification of the theory is critically assessed by peers before publication. Critical peers have the competence to criticize a paper, and will try to find flaws in the justification of the theory, much as a lawyer would try to find flaws in the argument of his or her opponent.

  Part of the justification to critical peers is that empirical tests do not depend on the person of the researcher, and hence are repeatable: Critical peers must be able to repeat the empirical tests. They will try to do so.

Surviving criticism and empirical testing is never final. Even for a theory that survived testing and criticism for a long time, it is always possible that someone will find a flaw or that a test will falsify part of the theory. Scientific theories are fallible. This does not mean that scientific theories are automatically rejected when falsified. Popper [27] does not say that the rule of falsification must be applied mechanically, and historical examples show that a falsification may be shelved as a puzzling result to be studied in more detail later. Persistent falsification may be responded to by a redefinition of concepts to define the phenomenon away, or by revising theoretical generalizations so that the phenomenon is not an anomaly anymore [1, 17, 19, 20].

3. WE NEED PARTIAL THEORIES

Theories are scientific when they survive the test of practice and the critique of competent peers. What makes a theory general?

Two elements make a theory general: Concepts and generalizations. Concepts, like *agile development* and *development effort*, refer to classes of phenomena. Generalizations, such as *agile projects deliver useful products faster than non-agile projects*, are claimed to be true of a class of phenomena.

Here we see a difference between basic science and engineering science. Again very briefly put, basic scientific research makes idealizing assumptions that are known to be false but that make patterns of behavior visible [22, 31]. Concepts like that of *point masses*, *frictionless surface*, *perfectly elastic body*, *absolute vacuum*, *rational actor* and *Turing machine* do not exist in the real world, but allow to analyze the patterns in phenomena conceptually or mathematically. This approach to knowledge is characteristic for modern science and has been called Galilean idealization [23, 24]. Laws of nature based on them are true of the idealizations but false of the real world [3]. Universality is obtained at the price of idealization. This means that the laws of basic science cannot be applied in practice [21, 2].

Where basic scientists spend their research budget on approximating the idealizing assumptions of basic laws of nature in the laboratory, engineering scientists spend their research budget on approximating the conditions of practice under which their artifacts must work [15, 18, 30]. So engineering is *not* applied science, at least not applied basic science. Transfer of knowledge from science to engineering is *not* linear [10, 16]. Some authors in the history of engineering suggest that basic science is hardly used in engineering at all [29].

In software engineering we need middle-range theories. A theory is *middle-range* if its generalizations do not have universal scope. The concept of middle-range theory was developed for the social sciences [26], but is also applicable to special sciences such as geology, meteorology and political science, which all produce middle-range theories. Engineering science produces theories of this kind of theory too [31]. We can view theories like COCOMO, the software engineering principles of Davis [8] and the theories listed by Endres & Rombach [9] as middle-range theories.

This means that researchers produce generalizations that make no unrealistic assumptions and that have less-than-universal scope. It also means that practitioners who want to apply these theories to their particular case should assess whether the theory is true for their case. This is a risk assessment. In a sense, a practitioner should build a theory of his or her particular case, based on more general, middle-range knowledge produced by researchers [28].

4. WE NEED DESIGN THEORIES

Design theories are middle-range scientific theories about the interaction of artifacts with their context [32]. The artifact in software engineering can be a method, a way of organizing, a way of coordinating, a technique, a notation, etc. used in the context of a software engineering project. Or it can be a pattern, architecture, algorithm, etc. used in a software system. The generic knowledge question in all these situations is:

- What is the effect of placing this artifact in this kind of context?

Variations of this question are: What happens to the effects if the artifact is changed a bit? How does this compare to the effect of related artifacts in this kind of context? What happens to the effects if the context changes? Is the effect robust when we scale up the context?

Design theories built this way are about the interactions between a class of artifacts and a class of contexts, where these classes are defined without making idealizing assumptions. The price we pay for this is a lack of clarity what exactly the class boundaries are. When is a project truly agile? And does our generalization hold for all "truly agile" projects without exception, regardless of the circumstances? Quite likely, our generalization is incompletely defined, and interesting empirical research can be done to determine its boundaries more clearly.

Another problem is that most of the time most generalizations are unfinished, and practitioners who consider to use them must decide for themselves if this is going to work in their case. What help is this to them? It would be safest for them if we could guarantee universal truths about how

an artifact is going to interact with a context. If we cannot guarantee that, then it is safer for them to simply say: "Works in general, but not here." But then of course they might miss an opportunity for improvement, which is uncomfortable too.

Before we turn to the use of partial generalizations to practitioners, I should remark that the above view of design theories has some similarity with the one by Gregor & Jones [**?**]. In their view as well as in mine, a scientific theory should at least consist of a conceptual framework, generalizations, and a scope claim. However, Gregor & Jones require an array of additional elements, such as design choices, artifact requirements, and artifact variants, that are not part of design theories in my view. Instead, these are elements of what a design theory is about: the interactions between an artifact and a context. A more detailed comparison has been given elsewhere [31].

5. WE NEED USABLE THEORIES

A theory that is universally true could be applied mechanically, without checking the conditions of the context of application first. Alas, the real world is to varied for that. Every case is unique, and universal truth is too strong a condition for many theories.

But design theories can possess many features that nevertheless make it attractive for practitioners to consider using them. Consider a generalization of the form

> "in cases that are architecturally similar to this,
> this artifact has effects like these."

For example, consider the context of a mature software development organization, containing requirements engineers, designers, programmers, testers, and a product manager. Consider as artifact a cross-functional team, which is a weekly meeting in which one representative of each of these stakeholders participates, and in which the coming week's work is discussed and aligned. In a longitudinal case study of a mature software development organization, Damian & Chisa [7] observed that introducing such a team in a development organization had the effect that the amount of rework was reduced. This supports the following design generalization:

> "In mature software development organizations,
> cross-functional teams reduce rework."

This is a design generalization because it is about the effects of introducing an artifact in a context. It is a middle-range generalization because there are exceptions to it, such as cross-functional teams that contain stakeholders who are at odds with each other and do not want to cooperate.

When is such a generalization usable by practitioners? When at least the following is true:

- It must be in the capability of the practitioner to recognize his or her case as being "similar" or "dissimilar" to the cases generalized about. It can be argued that recognition of similarity is more reliable if the similarity is defined in terms of structural properties of the case, such as the entities, actors, and their interactions, that play a role in this case [31].

- It must be in the capability of the practitioner to realize the artifact in his or her context. This includes sufficient resources, time and money.

- The predicted effects must not be interfered with by other mechanisms in the case, that are not mentioned in the theory. This is perhaps the hardest and riskiest part of the practitioner's assessment of usability, because here the theory is confronted with the unique characteristics of a case and there is no uniform way to combine case knowledge with theoretical knowledge. The real world is non-compositional. It is not only full of details, it is also full of unexpected interactions.

- The practitioner must be able to recognize the effects when they occur.

This list shows that usability of a theory by a practitioner in a context depends on the capabilities of the practitioner and the properties of the context.

In addition to a design theory being usable, it must be *useful*. This depends on the normative properties of the context:

- The predicted effects must contribute to stakeholder goals,

- without violating applicable norms or

- exceeding the available budget.

All of this adds up to a case-dependent argument that compares potential benefit (contribution to stakeholder goals) with cost (use of resources) as well as the risk of producing undesirable effects.

In this argument, the fact that generalizations are middle-range does not add any complexity. The same argument would have to be made when applying basic scientific knowledge: Does our case satisfy the idealizing assumptions of the theory? Or rather, does our violation of the idealizing assumptions matter for the effects to be achieved?

6. IMPLICATIONS

Even though I have given a few examples from software engineering, none of the above is specific for software engineering. So what is the relevance for software engineering? The relevance is that I claim the above arguments to be valid for all engineering sciences, including software engineering. If this is so, then this has some consequences for the way we should develop theories of software engineering.

- As engineering researchers we should scale up our artifacts to conditions of practice and develop theories about their effects in a class of real-world contexts. These theories will be middle range and should not make unrealistic assumptions about the context [30].

- Empirical researchers in software engineering should develop more theories than they do now [11, 12, 13, 32]. These theories need not be universal.

- Software engineering researchers should not only propose new theories, they should elaborate and improve existing theories by exploring their scope in empirical research. The result of this kind of theory-development research should be a refined and possibly even reconceptualized middle range theory of some aspect of software engineering.

As a starting point for this kind of theory-development research, we could take the observations, principles, and generalizations of Davis [8] and Endres & Rombach [9], and further develop these in more refined and better-tested middle-range design theories.

7. REFERENCES

[1] W. Bechtel and R. Richardson. *Discovering Complexity: Decomposition and localization as strategies in scientific research*. MIT Press, 2010. Reissue of the 1993 edition with a new introduction.

[2] M. Boon. How science is applied in technology. *International Studies in the Philosophy of Science*, 20(1):27–47, March 2006.

[3] N. Cartwright. *How the Laws of Physics Lie*. Oxford University Press, 1983.

[4] N. Cartwright. *The Dappled World. A Study of the Boundaries of Science*. Cambridge University Press, 1999.

[5] A. Cournand and M. Meyer. The scientist's code. *Minerva*, 14(1):79–96, 1976.

[6] C. Craver. Structure of scientific theories. In P. Machamer and M. Silberstein, editors, *The Blackwell Guide to the Philosophy of Science*, pages 55–79. Blackwell, 2002.

[7] D. Damian and J. Chisan. An empirical study of the complex relationships between requirements engineering processes and other processes that lead to payoffs in productivity, quality and risk management. *IEEE Transactions on Software Engineering*, 32(7):433–453, July 2006.

[8] A. Davis. *201 Principles of Software Development*. McGraw-Hill, 1995.

[9] A. Endres and D. Rombach. *A Handbook of Software and Systems Engineering: Empirical Observations, Laws and Theories*. Pearson Addison Wesley, 2003.

[10] B. Godin. The linear model of innovation: The historical reconstruction of an analytic framework. *Science, Technology and Human Values*, 31(6):639–667, November 2006.

[11] P. Johnson, M. Ekstedt, and I. Jacobson. Where's the theory for software engineering? *IEEE Software*, pages 94–96, September/October 2012.

[12] M. Jørgensen and D. Sjøberg. Generalization and theory-building in software engineering research. In S. Linkman, editor, *8th Internation Conference on Empirical Assessment in Software Engineering EASE 2004 Workshop 26th International Conference on Software Engineering*, volume 1, pages 29–35. IEE Proceedings, 2004.

[13] M. Jørgensen and D. Sjøberg. Generalization and theory-building in software engineering research. *IEE Seminar Digests*, 2004(920):29–35, 2004.

[14] A. Kaplan. *The Conduct of Inquiry. Methodology for Behavioral Science*. Transaction Publishers, 1998. First edition 1964 by Chandler Publishers.

[15] R. Kline. Construing "technology" as "applied science": Public rethoric of scientists and engineers in the United States, 1880 – 1945. *Isis*, 86(2):194–221, 1995.

[16] S. Kline. Innovation is not a linear process. *Research Management*, 24(4):36–45, July/August 1985.

[17] T. Kuhn. *The Structure of Scientific Revolutions*. University of Chicago Press, second, enlarged edition, 1970.

[18] G. Küppers. On the relation between technology and science—goals of knowledge and dynamics of theories. The example of combustion technology, thermodynamics and fluid dynamics. In W. Krohn, E. Layton, and P. Weingart, editors, *The Dynamics of Science and Technology. Sociology of the Sciences, II*, pages 113–133. Reidel, 1978.

[19] I. Lakatos. Falsification and the methodology of scientific research programmes. In I. Lakatos and A. Musgrave, editors, *Criticism and the Growth of Knowledge*, pages 91–196. Cambridge University Press, 1970.

[20] I. Lakatos. *Proofs and Refutations*. Cambridge University Press, 1976. Edited by J. Worall and E. Zahar.

[21] R. Laymon. Applying idealized scientific theories to engineering. *Synthese*, 81:353–371, 1989.

[22] R. Laymon. Experimentation and the legitimacy of idealization. *Philosophical Studies*, 77:353–375, 1995.

[23] E. McMullin. A case for scientific realism. In J. Leplin, editor, *Scientific Realism*, pages 8–40. University of California Press, 1984.

[24] E. McMullin. Galilean idealization. *Studies in the History and Philosophy of Science*, 16(3):247–273, 1985.

[25] R. Merton. The normative structure of science. In *Social Theory and Social Structure*, pages 267–278. The Free Press, 1968. Enlarged Edition.

[26] R. Merton. On sociological theories of the middle range. In *Social Theory and Social Structure*, pages 39–72. The Free Press, 1968. Enlarged Edition.

[27] K. Popper. *The Logic of Scientific Discovery*. Hutchinson, 1959.

[28] P. Van Strien. Towards a methodology of psychological practice: The regulative cycle. *Theory & Psychology*, 7(5):683–700, 1997.

[29] W. Vincenti. *What Engineers Know and How They Know It. Analytical Studies from Aeronautical History*. Johns Hopkins, 1990.

[30] R. Wieringa. Empirical research methods for technology validation: Scaling up to practice. *Journal of Systems and Software*, 2013.

[31] R. Wieringa. *Design Science Methodology for Information Systems and Software Engineering*. Springer, To be published, 2014.

[32] R. Wieringa, M. Daneva, and N. Condori-Fernandez. The structure of design theories, and an analysis of their use in software engineering experiments. In *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 295 –304. IEEE Computer Society, sept. 2011.