Design Science Methodology: Principles and Practice

Roel Wieringa
University of Twente
The Netherlands
roelw@cs.utwente.nl

Design scientists have to balance the demands of methodological rigor that they share with purely curiosity-driven scientists, with the demands of practical utility that they share with utility-driven engineers. Balancing these conflicting demands can be conceptually complex and may lead to methodological mistakes. For example, treating a design question as an empirical research question may lead to researcher to omit the identification of the practical problem to solve, to omit the identification of stakeholder-motivated evaluation criteria, or to omit trade-off and sensitivity analysis. This tutorial aims to clear up this methodological mist in the case of software engineering (SE) research.

The core distinction is that between practical problems and knowledge questions. A practical problem is a difference between stakeholder goals and experiences, that they wish to reduce, and a knowledge question is a lack in knowledge, that they wish to reduce [14]. For example, to reduce the number of build failures in distributed SE projects is a practical problem; to ask for the relation between team communication structure and code integration build failures is a knowledge question [17].

In all kinds of scientific research, knowledge questions and practical problems are mutually nested. For example, an empirical research question is a knowledge question for which the researcher must *do* something to answer it, namely to perform research. But to perform empirical research is in turn a practical problem. The research problem must be analyzed, the research must be designed and validated, etc. Conversely, to solve any practical problem, the problem-solver must know what the problem is, which is a knowledge question; and to assess whether a solution design would solve the problem, the problem solver must predict what would happen if the design were implemented in the problem domain, which is another knowledge question. The mutual recursion of practical problems and knowledge questions can be confusing, and may easily cause the researcher to miss relevant problems to solve, or questions to ask.

In design science, the top-level problem in this mutual problem nesting hierarchy is always a practical problem [15]. In software engineering, practical problems are always problems in the design, construction or maintenance of software systems—the software engineering domain. Artifacts designed or investigated in software engineering research can be algorithms, techniques, methods, tools, notations, or even conceptual frameworks used in the software engineering domain [11].

There are two roles that empirical research can play with respect to artifact design, namely *validation* of a designed artifact before it has been transferred to practice, and *evaluation* of the performance of an implemented design after it has been transferred to practice. For example, testing a new fault localization technique before it is transferred to practice is a validation study [9], but a case study of long-term usage of a system for software engineering measurement and analysis in a company [1] is an evaluation study. In validation research there is no practical experience with the artifact yet and any prediction of its future behavior will have some degree of uncertainty. The challenge of validation research is to reduce this uncertainty, for example by scaling up from controlled laboratory conditions to uncontrolled conditions of practice.

In both validation and evaluation, typical design research questions are the same, except that validation questions ask for what will happen and evaluation questions ask for what has happened. Where there is a difference, the questions are given in their validation form:

- How to operationalize a stakeholder goal into measurable design criteria? An example is the operationalization of the concept of flying quality of aircraft [13].
- Design prediction: What will be the effect of this artifact in this problem domain? For example, what is the effect of a new fault localization technique on the cost of fault localization [9]?
- Effect valuation: How well does this effect match stakeholder criteria? Does the use of a tool for capturing socio-technical relationships in software development serve the goals of software engineers [10]?.
- Trade-off analysis: What would be the effect if the artifact design is changed? For example, how does a new fault localization technique perform with respect to alternative techniques [9]?
- Sensitivity analysis: What would be the effect if the problem domain changes? For example, if the tool works for 10 000 line programs, does it still work for 1000 000 line programs?

Research methods to investigate these questions do not differ from research methods used in the natural or behavioral sciences [3, 7, 8], but in validation research there is a progression of methods from the controlled conditions of the lab to the uncontrolled conditions of practice [16]. Some examples of validation research methods are lab experiments, benchmarking, pilot studies, technical action research and user evaluation studies.

The scope of design knowledge transcends the individual case but is rarely universal, which contrasts it with basic science with its universal knowledge claims [5, 12]. Design science theories are theories of practice, which means they incorporate some of the conditions of practice that basic scientists, aiming for universal generalizations, abstract from [4].

Knowledge can accumulate by trying to understand how the interaction between an artifact and a problem domain in particular cases produces effects. We can do this in a bottom-up way by identifying generalizable underlying mechanisms of interaction between an artifact and the domain in which it is inserted. This has been called analytical generalization by some methodologists [6, 18]. We can also use a top-down approach in which a theory from another domain, such as social science or cognitive psychology, is applied to software engineering [11].

The structure of design theories is the same as the structure of any scientific theory but reflects the role of design theories in practical problem solving in the choice of research questions and in the statement of intermediate scope. There are three components.

- Conceptual framework: Constructs including operationalization of stakeholder-motivated criteria
- Design prediction (*Artifact* $\wedge$ *Problem domain causes Effect*) and valuation (*Effect satisfies Criteria*)
- Scope: Range of variation in Artifact and Problem domain that still produces the Effects.

Design theories are not prescriptive, as some methodologists claim [2]. It is the artifact *specification* that is prescriptive, and design theories can be used by a design scientist to justify why a class of artifacts will solve a class of problems.

1. REFERENCES

[1] I. Coman, A. Sillitti, and G. Succi. A case-study on using an automated in-process software engineering measurement and analysis system in an industrial environment. In *ICSE '09: Proceedings of the 2009 IEEE 31st International Conference on Software Engineering*, pages 89–99, Washington, DC, USA, 2009. IEEE Computer Society.

[2] S. Gregor and D. Jones. The anatomy of a design theory. *Journal of the AIS*, 8(5):312–335, May 2007.

[3] B. Kitchenham, S. Pfleeger, D. Hoaglin, K. Emam, and J. Rosenberg. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering*, 28(8):721–733, August 2002.

[4] 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.

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

[6] R. Pawson and N. Tilley. *Realistic Evaluation*. Sage Publications, 1997.

[7] S. Pfleeger. Understanding and improving technology transfer in software engineering. *Journal of Systems and Software*, 47(2–3):111–124, July 1999.

[8] P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14:131–164, 2009.

[9] R. Santelices, J. Jones, Y. Yu, and M. Harrold. Lightweight fault-localization using multiple coverage types. In *ICSE '09: Proceedings of the 2009 IEEE 31st International Conference on Software Engineering*, pages 56–66, Washington, DC, USA, 2009. IEEE Computer Society.

[10] A. Sarma, L. Maccherone, P. Wagstrom, and J. Herbsleb. Tesseract: Interactive visual exploration of socio-technical relationships in software development. In *ICSE '09: Proceedings of the 2009 IEEE 31st International Conference on Software Engineering*, pages 23–33, Washington, DC, USA, 2009. IEEE Computer Society.

[11] D. Sjøberg, T. Dybå, B. Anda, and J. Hannay. Building theories in software engineering. In F. Shull, J. Singer, and D. Sjøberg, editors, *Guide to advanced empirical software engineering*, pages 312–336. Springer, 2008.

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

[13] W. Vincenti. Establishment of design requirements: Fflying-quality specifications for American aircraft, 1918–1943. In *What Engineers Know and How They Know It. Analytical Studies from Aeronautical History*, pages 51–108. Johns Hopkins, 1990.

[14] R. Wieringa, N. Maiden, N. Mead, and C. Rolland. Requirements engineering paper classification and evaluation criteria: A proposal and a discussion. *Requirements Engineering*, 11(1):102–107, March 2006.

[15] R. J. Wieringa. Design science as nested problem solving. In *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology, Philadelphia*, pages 1–12, New York, 2009. ACM.

[16] R. J. Wieringa and J. M. G. Heerkens. Design science, engineering science and requirements engineering. In *16th IEEE International Requirements Engineering Conference, Barcelona, Spain*, pages 310–313, Los Alamitos, 2008. IEEE Computer Society Press.

[17] T. Wolf, A. Schroter, D. Damian, and T. Nguyen. Predicting build failures using social network analysis on developer communication. In *ICSE '09: Proceedings of the 2009 IEEE 31st International Conference on Software Engineering*, pages 1–11, Washington, DC, USA, 2009. IEEE Computer Society.

[18] R. Yin. *Case Study research: Design and Methods*. Sage Publications, 2003. Third Edition.