

Methodologies of Requirements Engineering Research and Practice: Position Statement

Roel J. Wieringa

Department of Computer Science, University of Twente, P.O. Box 217, 7500 AE
Enschede, the Netherlands <http://www.csw.utwente.nl/~roelw>

Abstract. In this position paper I argue that RE practice is the problem analysis part of a design problem, and that this problem analysis part is a knowledge problem in which the requirements engineer tries to build a theory of a problem domain. RE research is a knowledge problem too, in which the researcher tries to build partial theories of a class of domains. In both cases, models can help to find theories, because models are entities that are more accessible, or more understandable, than their subjects and therefore help to find a theory of their subject. I compare a number of concepts closely related to that of a model, including that of paradigm, benchmark and exemplar. Finally, I argue that the evaluation of RE theories, at whatever level, is not different from the evaluation of any other theory: They should be repeatably and intersubjectively true.

In this position paper I argue that requirements engineering (RE) practice and RE research are both knowledge problems, but at two different levels of abstraction. Due to space restrictions I argue this in a series of probably controversial claims that will hopefully provoke discussion. To make them easily identifiable, the claims are written in sans serif font.

1 Requirements “Engineering” is a Knowledge Problem

A *problem* is a difference between what is perceived and what is desired, that we want to reduce (Dewey 1933). An *action problem* is a desire to change the world; a *knowledge problem* is a desire to increase our knowledge about the world. A characteristic difference between action problems and knowledge problems is that action problems involve criteria, in other words norms, by which the desired change is evaluated (Wieringa 1996, page 52). Knowledge problems are free of norms; or rather, the only norm is that the desired knowledge be true.

Claim 1

Action problems, in turn, can be classified into two kinds. A *design problem* is a desire to specify a change and an *implementation problem* is a desire to implement a specified change. To solve a design problem, we must do two things: Analyze the problem and specify a solution. I define *requirements engineering* as design problem analysis. It involves, among others, identifying the problematic phenomena, identifying the norms with respect to which they are problematic, analyzing causal relationships between the phenomena, etc.

In practice, design problems, implementation problems and knowledge problems occur in a bundle and we cannot separate the problem-solving process in a sequence of tasks each of which is devoted to one of these problems only. My analysis is a rational reconstruction of the logical structure of a process in which many things happen simultaneously (Wieringa 1996, pages 368, 372). Requirements engineering usually occurs simultaneously with other design and implementation tasks.

2 Requirements Engineering as Theory-Building

A consequence of my view is that requirements “engineering” is not engineering at all! Engineering is designing and implementing a solution to an action problem. RE, by contrast, is theory-building, where a *theory* is a coherent description of the causal relationships between the phenomena in a particular domain (Kaplan 1998). RE is building a theory about the domain of a design problem. Claim 2

It is useful to distinguish three kinds of theories. A *universal theory* is claimed to be valid for the entire universe, past, present and future. Theories in physics and chemistry are of this kind. Requirements engineers never produce this kind of theory. Claim 3

A *partial theory* is claimed to be valid for a class of domains. However, the class is not specified independently from the theory: If it were, we could embed the partial theory T into a universal theory of the form “If C then T ”. For partial theories, such an independent criterion C for the applicability of T is not known. For example, let T be “repetitive manual procedures often cause errors”. Not all repetitive manual procedures cause errors, and there is no independent criterion C that characterizes exactly the cases in which they do. So the applicability of a partial theory depends upon unknown contextual factors. Requirements engineers may use partial theories, e.g. from organizational science, to help them focus on possible causes of problems in a problem domain. In addition, over years of experience with similar problem domains, they build up partial theories themselves.

A *domain theory* is a description of causal relationships in a particular problem domain. When analyzing a problem domain, a requirements engineer builds up a domain theory describing the domain phenomena and their causal relationships. For example, an analysis of productivity problems in a medical laboratory may reveal that a manual patient administration causes errors in invitation letters, which in turn cause inconvenience to the patients invited by these letters to come to an appointment. These causal relationships are not universal: The very same phenomena may have different causal relationships in other medical laboratories.

Requirements engineering research aims to find partial theories and requirements engineer practice aims to find particular domain theories. These two levels are distinct but may occur intertwined. For example, RE research may consist of performing a series of particular RE projects, in the course of which domain Claim 4

theories are built, and drawing general conclusions from these in the form of a partial theory.

3 Models, Paradigms, and Problem Frames

Theories can be developed using models. The word “model” has many meanings, ranging from scale models to diagrams. Here, I restrict its meaning to what I regard as its core. An entity M is a *model* of an entity S if the study of M yields knowledge about S (Apostel 1961). We call S the *subject* of M . Classical examples are the use of water waves as a model of sound waves and the use of a hydraulic network as a model of an electrical network. In RE, models are used to build problem domain theories.

Claim 5

Models help us building a theory of a subject if the subject is unknown or difficult to access. For example, to analyze problem A, the requirements engineer may use problem B as a model of A. Problem B may be one that he or she has been involved in, or that one analyzed by others and described in books or papers. Or problem B may be a part of A, analyzed in detail, and then used as a model of A.

Paradigms too help us constructing theories. The term “paradigm” was introduced by Kuhn (1970) to describe normal science. Masterman (1970) shows that Kuhn uses the word “paradigm” in three different ways. A *construct paradigm* is an example found in all recognized textbooks, or in a classic work continuously referred to. It is used for explanation to new members of the field.

A *sociological paradigm* is not a particular example, but a kind of example that is used by members of a scientific community, and is a defining characteristic of that community. If you are a member of that community, you use this kind of example. Examples from physics are Ptolemaic astronomy and Copernican astronomy. Examples in RE are goal-oriented RE and ethnographic approaches to RE.

Claim 6

A *metaphysical paradigm* is a way of seeing phenomena. In software engineering, object-oriented adepts see objects everywhere, while others see aspects everywhere. In RE, some people see goals everywhere, and others see implicit meanings and hidden intentions in every problem domain.

Claim 7

Finally, problem frames help us build a theory of the problem domain. As shown by Schön (1983) and Cross (2001), experienced designers frame a problem in the light of a preferred solution direction. Jackson (2000) presents a number of RE problem frames, where each frame gives a view on the problem from the perspective of a particular kind of solution. For example, a required behavior problem is a problem whose solution is a controller, and an information display problem is a problem whose solution is an information display.

Claim 8

A problem frame can be described by a particular example, such as the sluice control example, that then can be used as a model of the problem at hand. To the extent that this model is used for this kind of problem by many practicing requirements engineers, it is a construct paradigm as defined above. If the problem frame example is a defining characteristic of a community of

requirements engineers, it has become a sociological paradigm. This is not the way these example applications of problem frames should be used. A problem frame should be used because it highlights a structure in the problem at hand, not because the requirements engineer is a member of a community that is in the habit of using this problem frame. By the same token, the use of problem frame examples as metaphysical paradigms goes against the spirit of problem frames. Claim 9

4 Implications for Requirements Engineering Research

Research is solving knowledge problems. In *empirical research*, we search for causal relationships of the form “If a occurs, then b is caused, and b has properties c”. Negative causal claims can also be informative: “If a occurs, then b is *not* caused”.

In *formal research*, we search for logical relationships of the form “if a is true, then b is true, and b has properties c” b may be a positive or negative proposition.

Empirical RE research should result in universal or partial theories about domain classes, about structures found in domains, causal relationships between phenomena, relationships between norms and possible operationalized indicators of those norms, etc. RE research will not yield many universal theories, but partial theories will be very useful. Formal RE research should result in notations and techniques to analyze domains. Claim 10

In no case can the outcome of research be a method. Research never results in prescriptions. But relevant RE research can be used to decide upon a method to use, using an argument of the following form: “I want to satisfy norm N. Research has shown that if I do a, then b will occur and b has properties c that conform to N. So I decide to do a.” Claim 11

The theme of this workshop is the evaluation of research results. There is only one criterion to evaluate research results: Is it repeatably and intersubjectively true? Repeatability makes the result independent from the time of production. Intersubjectivity is, by definition, independence from the researcher who establishes the result. A proposition is intersubjectively true if its truth is independent from the person who is evaluating its truth value. Intersubjectivity is as far as we can go in our quest for objectivity, the difference being that intersubjectivity still allows for collective delusion. (We should of course always strive to eliminate such delusions.) A consequence of this view is that when explaining a result, researchers should also explain how they obtained their results so that others can try to reproduce or refute them. Claim 12

The difference between RE research and software engineering (SE) research is that SE research is design research, for it is about software solutions. However, design research too solves knowledge problems and, as all research, this results in empirical or formal knowledge. For example, the researcher may propose a new solution technique and claim that it has certain properties. One way to stimulate the achievement of repeatable and intersubjective validity of such a claim is to use *benchmarks*, which are sets of operations to be performed by the Claim 13

Claim 14

researcher to substantiate his or her claims (Sim et al. 2003). For example, if the researcher claims that a certain distributed database query optimization has a better performance than other algorithms, he or she can test it empirically on a benchmark distributed database and compare the results with those of other algorithms. (Alternatively, the researcher can perform a formal analysis to prove the claim.) Software pattern research is (or should be) of this kind.

So benchmarks play a role in software engineering research and other kinds of design research. They do not play a role in requirements engineering research because RE research is about problem analysis, not about solution design. *Exemplars* are similar to benchmarks in that they are intended to compare research results, but without the implication that this comparison will be quantitative (Feather et al. 1997).

Claim 15

References

- Apostel, L. (1961), Towards a formal study of models in the non-formal sciences, in H. Freudenthal, ed., 'The Concept and Role of the Model in the Mathematical and the Natural and Social Sciences', Reidel, pp. 1–37.
- Cross, N. (2001), Design cognition: results from protocol and other empirical studies of design activity, in C. Eastman, W. McCracken & W. Newstetter, eds, 'Design Knowing and Learning: Cognition in Design Education', Elsevier, pp. 79–103.
- Dewey, J. (1933), *How We Think: A Restatement of the Relation of Reflective Thinking to the Educative Process*, D.C. Heath and Company.
- Feather, M., Fickas, S., Finkelstein, A. & Lamsweerde, A. v. (1997), 'Requirements and specification exemplars', *Automated Software Engineering*.
- Jackson, M. (2000), *Problem Frames: Analysing and Structuring Software Development Problems*, Addison-Wesley.
- Kaplan, A. (1998), *The Conduct of Inquiry. Methodology for Behavioral Science*, Transaction Publishers. First edition 1964 by Chandler Publishers.
- Kuhn, T. (1970), *The Structure of Scientific Revolutions*, second, enlarged edn, University of Chicago Press.
- Masterman, M. (1970), The nature of a paradigm, in I. Lakatos & A. Musgrave, eds, 'Criticism and the Growth of Knowledge', Cambridge University Press, pp. 59–89.
- Schön, D. (1983), *The Reflective Practitioner: How Professionals Think in Action*, Arena.
- Sim, S., Easterbrook, S. & Holt, R. (2003), Using benchmarking to advance research: A challenge to software engineering, in '25th International Conference on Software Engineering (ICSE 2003)', IEEE Computer Society Press, pp. 74–83.
- Wieringa, R. (1996), *Requirements Engineering: Frameworks for Understanding*, Wiley.