Is Requirements Engineering Inherently Counterproductive?

Paul Ralph
Dept. of Computer Science
University of Auckland
Auckland, New Zealand
paul@paulralph.name

Rahul Mohanani
Dept. of Information Processing Science
University of Oulu
Oulu, Finland
rahul.mohanani@oulu.fi

***Abstract*—This paper explores the possibility that requirements engineering is, in principle, detrimental to software project success. Requirements engineering is conceptually divided into two distinct processes: sensemaking (learning about the project context) and problem structuring (specifying problems, goals, requirements, constraints, etc.). An interdisciplinary literature review revealed substantial evidence that while sensemaking improves design performance, problem structuring reduces design performance. Future research should therefore investigate decoupling the sensemaking aspects of requirements engineering from the problem structuring aspects.**

***Index Terms*—Requirements Engineering, Sensemaking, Problem Structuring, Domain Knowledge, Design.**

I. Introduction

Many people believe that requirements engineering (RE) should produce an "unambiguous", "consistent", "complete", "feasible", "traceable" and "verifiable" requirements specification [1, p. 11]. While some eschew a comprehensive system requirements specification in favor of user stories [2], scenarios [3], use cases [4] or other documents, no one is championing "ambiguous", "inconsistent", "incomplete," and "confused" requirements specifications. Rather, it appears obvious that a clearer, more structured account of what is needed or wanted will improve the likelihood of success.

But is this true? Or is it an example of oversimplifying and over-rationalizing complex phenomena [5]? To investigate this, we pose the following research question.

Research Question: *Does any existing theoretical or empirical research support the proposition that Requirements Engineering is fundamentally detrimental to software engineering success?*

Here, *software engineering success* refers to the net impact of a software project or product on its stakeholders over time [6]. Requirements Engineering, in contrast, is more difficult to define (see below).

We approach the research question by reviewing existing empirical studies of problem solving, problem structuring, creativity, goal understanding and sensemaking from cognitive science, psychology, sociology, management and software engineering (§II). We then return to our research question and recommend areas for future research (§III).

II. Literature Review And Analysis

A) What do we mean by Requirements Engineering?

RE has been defined as the subfield of software engineering pertaining to "eliciting", "modeling", "analyzing", "communicating", "agreeing" and "evolving" requirements [7]. The problem with this definition is that many projects appear to proceed with no meaningful requirements [8] or little agreement on goals and requirements [9]. Analysts, moreover, do not "elicit" requirements because requirements are not hiding – preformed – in the minds of project stakeholders. Rather, analysts are usually faced with an ambiguous problematic context, replete with conflicting perceptions, viewpoints, values and goals [9]. RE thus includes at least two related but distinct processes (Figure 1), as follows.

1. *Sensemaking*, which involves learning, understanding and organizing beliefs about the context [10].
2. *Problem structuring*, which involves assigning socially constructed constraints, parameters, options, states, choices and other specifics to the context [11].

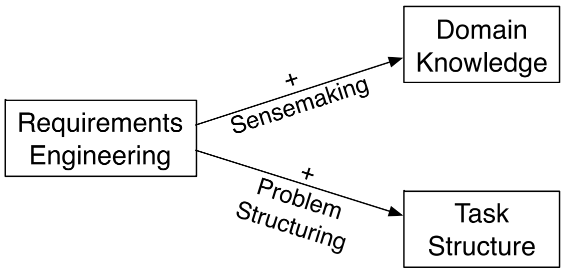

FIGURE 1. Initial Model of RE Effects

Problem structuring involves *inventing* goals, problems, constraints and other structure elements [12]. Problem structure does not exist in an objective reality waiting for discovery; it is created by human actors [13].

Differentiating the two aims of RE forms the basis of our inquiry and raises two new questions: How do domain knowledge (developed through sensemaking) and task structure (developed through problem structuring) affect software engineering success?

B) Domain Knowledge

Domain knowledge comprises memory categories learned from a specific environment and used to solve problems and make decisions [14]. Domain experts can retrieve relevant domain knowledge and strategies with minimal cognitive effort and can exert greater cognitive control over their performance [15]. Greater domain knowledge improves problem comprehension [16] and working memory [17]. It facilitates mental simulation, allowing designers to explore the solution space without overlooking important concerns outside the current level of detail and therefore iterate on initial designs more effectively [18]. Domain knowledge is therefore critical for cognitive tasks [19] including idea generation [20], design [21] and problem solving [22], [23].

While most empirical studies appear to indicate that domain knowledge and design performance are positively related, higher domain knowledge can sometimes lead to lower performance, for instance, by reducing divergent thinking [24], encouraging more narrowly defined problems [25] or promoting mental set fixation [15] leading to less creative solutions. It can also impair judgment by interfering with mental heuristics [26].

C) Task Structure

We often conceptualize problems on a spectrum from well-structured to ill-structured. "Well-structured problems are constrained problems with [correct,] convergent solutions that engage the application of a limited number of rules and principles within well-defined parameters. Ill-structured problems possess multiple solutions, solution paths, fewer parameters which are less manipulable, and contain uncertainty about which concepts, rules, and principles are necessary for the solution or how they are organized and which solution is best" [27, p. 65].

Simon [28], [29] argues that despite the fact that most design problems are ill-structured, "problem-solving theory that is based upon the solution of well-structured problems should serve as the basis for all problem solving" [30, p. 7]. Empirical evidence contradicts this prevalent view [30], [31].

Performance solving well-defined problems is unrelated to performance solving ill-defined problems [32]. Solving well- and ill-defined problems involves different psychological processes [27], [33] and neurophysiological structures [34]. "The associative engine and neural structures that support imprecise, ambiguous, abstract, indeterminate representations are lateralized in the right prefrontal cortex, while the inference engine and neural structures that support precise, unambiguous, determinant representations are lateralized in the left prefrontal cortex" [34, p. 1].

Moreover, myriad empirical studies have found that higher task structure does not necessarily improve outcomes. For example, nonspecific or "open" goals lead to more learning [35]-[37] and increase unconscious assimilation of problem-relevant information [38]. More abstract task framing leads to more novel creations than more specific task framing [39]. Presenting conflicting objectives leads to more effective solutions [40]. In summary, "over-concentration on problem definition does not lead to successful design outcomes" [41, p. 439]. These findings all suggest a negative relationship between task structure and design performance.

D) Mechanisms for Structure Impeding Design

How can clearer, more structured problems possibly impair design? At least three mechanisms have been proposed:

1. Nonspecific goals lead to more learning because they reduce cognitive load [35], [36] (since the problem solver has fewer parameters to remember) or encourage hypothesis testing [37].
2. Designers often fixate on early solution ideas [42], existing solutions [43], [44] and bogus requirements [45].
3. Designers adopt different cognitive strategies for ill-structured and well-structured problems. Designers employ at least four cognitive strategies, which differ by their degree of information gathering, solution generation and use of prior knowledge [46]. Better designers appear to gather a moderate amount of information and process it quickly into their problem schema [35], [41], [46].

While the precise mechanism by which increasing task structure decreases design performance remains unclear, design expertise appears to moderate the relationship. Expert designers disregard task structure [47] – they treat all problems as nonspecific and ill-structured, focusing on solution generation rather than problem analysis [41], [48].

E) Proposed Model of RE Effects

To summarize, RE involves both making sense of an ambiguous context and assigning structure to the context. RE therefore increases both domain knowledge and task structure. The balance of evidence suggests that while increased domain knowledge improves design performance, increased task structure degrades design performance (Figure 2).

The proposed model's implications are entangled with common misunderstandings about design work. Unlike designers in other fields, software developers sometimes differentiate between *design*, a step between requirements and implementation in an idealized systems lifecycle, and *development*, the complete process of conceptualizing, creating and maintain software intensive systems [49]. This distinction is incommensurate with the vast body of empirical research on how designers actually work [13], [48], [50]. Expert designers explore problematic contexts *by* generating solution concepts [13], [51]. They intentionally maintain ambiguity and continue problem framing throughout the development process [52]. Problem framing and designing solutions are one and the same process – coevolution [49], [51]. Lifecycle views of software development therefore incorrectly categorize problem structuring with sensemaking instead of with design. Consequently, structuring tasks are incorrectly assigned to managers and analysts instead of architects and developers. Requirements analysts cannot structure problems for designers because problem structuring is inextricably entangled with design [13]. This realization motivates several recommendations, below.

III. Conclusion and Recommendations

While much research has investigated particular methods, tools and techniques for requirements engineering, this paper questions whether RE is inherently counterproductive. It divides RE into two related but distinct processes (sensemaking and problem structuring), and reviews the

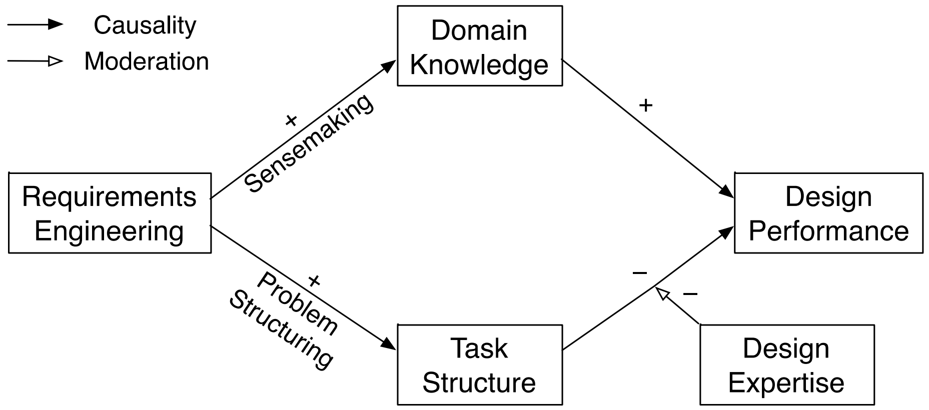

FIGURE 2. Dichotomous Effects of Requirements Engineering

literature on how each process affects outcomes. While higher domain knowledge is mostly associated with higher design performance and more success, more structured tasks often lead to lower design performance and less success. This suggests that the specifying, structuring aspects of RE are counterproductive while the learning, sensemaking aspects of RE are beneficial.

Practically speaking, our findings suggest the following recommendations for requirements analysts, developers, project managers and project clients.

1. Avoid over-structuring, oversimplifying and over-rationalizing problem contexts.
2. Recognize, analyze and accept ambiguity and uncertainty as unavoidable and even beneficial.
3. Expect problem structuring and high-level design to occur simultaneously.
4. Assign problem structuring and high-level design to the same individuals or teams.

These findings and recommendations should be considered in light of two main limitations. Figure 2 is an illustration of our thesis rather than a comprehensive theory of RE. It does not include negotiation or agreement and shows only the analyst's perspective. Furthermore, much of the reviewed literature concerns non-software artifacts and may not perfectly generalize to the software context. With these limitations in mind, we recommend the following areas for future research.

1. Decoupling sensemaking and problem structuring.
2. Techniques, tools and practices for representing, analyzing and addressing ill-structured design tasks; cf. [27], [53], [54].
3. Direct testing of the proposed Model of RE Effects within software engineering contexts.
4. Instruments for measuring domain knowledge, task structure and design performance in software engineering.
5. The relationship between problem structuring and law, particularly accommodating ambiguity in software contracts and public procurement.
6. The potential for specifically asking for creative solutions to mitigate over-structuring; cf. [55].

In conclusion, poor design performance cannot be overcome by proposing better problem structuring methods, practices or representations. More structure is inherently deleterious to design performance and software engineering success. The solution is therefore less structure rather than a better structuring method. Consequently, we call for more research on decoupling the useful, sensemaking aspects of RE from the counterproductive structuring aspects of RE.

REFERENCES

[1] ISO/IEC/IEEE Standard 29148:2011(E), "Systems and software engineering – Life cycle processes – Requirements engineering."

[2] M. Cohn, *User Stories Applied*. Addison Wesley, 2004.

[3] J. M. Carroll, "Five reasons for scenario-based design," *Interacting with Computers*, vol. 13, no. 1, pp. 43–60, 2000.

[4] A. Cockburn, *Writing Effective Use Cases*. Addison-Wesley, 2000.

[5] M. W. Lewis, "Exploring Paradox: Toward a More Comprehensive Guide," *The Academy of Management Review*, vol. 25, no. 4, pp. 760–776, 2000.

[6] P. Ralph and P. Kelly, "The Dimensions of Software Engineering Success," *Proceedings of the 2014 International Conference on Software Engineering*, Hyderabad, India: ACM, 2014.

[7] B. Nuseibeh and S. Easterbrook, "Requirements engineering: a roadmap," *Proceedings of the Conference on The Future of Software Engineering*, 2000, pp. 35–46.

[8] P. Ralph, "The Illusion of Requirements in Software Development," *Requirements Engineering*, vol. 18, no. 3, pp. 293–296, 2013.

[9] P. Checkland, *Systems Thinking, Systems Practice*. Chichester: Wiley, 1999.

[10] K. E. Weick, K. M. Sutcliffe, and D. Obstfeld, "Organizing and the Process of Sensemaking," *Organization Science*, vol. 16, no. 4, pp. 409–421, 2005.

[11] K. Dorst, "Design Problems and Design Paradoxes," *Design Issues*, vol. 22, no. 3, pp. 4–17, 2006.

[12] L. Nguyen and G. Shanks, "A framework for understanding creativity in requirements engineering," *Information and Software Technology*, vol. 51, no. 3, pp. 655–662, 2009.

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

[14] C. M. Ford, "A Theory of Individual Creative Action in Multiple Social Domains," *The Academy of Management Review*, vol. 21, no. 4, pp. 1112–1142, Oct. 1996.

[15] M. Chi, “Two approaches to the study of experts' characteristics,” in *The Cambridge Handbook of Expertise and Expert Performance*, Cambridge University Press, 2006, pp. 21–30.

[16] D. H. Jonassen, “Toward a design theory of problem solving,” *Educational Technology Research and Development*, vol. 48, no. 4, pp. 63–85, Dec. 2000.

[17] D. Hambrick, “Effects of Domain Knowledge, Working Memory Capacity, and Age on Cognitive Performance: An Investigation of the Knowledge-Is-Power Hypothesis,” *Cognitive Psychology*, vol. 44, no. 4, pp. 339–387, Jun. 2002.

[18] B. Adelson and E. Soloway, “The Role of Domain Experience in Software Design,” *IEEE Transaction on Software Engineering*, vol. 11, no. 11, pp. 1351–1360, Nov. 1985.

[19] J. F. Voss, T. R. Greene, T. A. Post, and B. C. Penner, “Problem-Solving Skill in the Social Sciences,” *Psychology of Learning and Motivation*, vol. 17, pp. 165–213, 1983.

[20] E. F. Rietzschel, B. A. Nijstad, and W. Stroebe, “Relative accessibility of domain knowledge and creativity: The effects of knowledge activation on the quantity and originality of generated ideas,” *Journal of Experimental Social Psychology*, vol. 43, no. 6, pp. 933–946, Nov. 2007.

[21] V. Popovic, “Expertise development in product design—strategic and domain-specific knowledge connections,” *Design Studies*, vol. 25, no. 5, pp. 527–545, Sep. 2004.

[22] J. M. Schraagen, “How Experts Solve a Novel Problem in Experimental Design,” *Cognitive Science*, vol. 17, no. 2, pp. 285–309, Apr. 1993.

[23] J. McDermott, “Domain knowledge and the design process,” *Design Studies*, vol. 3, no. 1, pp. 31–36, Jan. 1982.

[24] A. M. Kilgour, “Improving the creative process: Analysis of the effects of divergent thinking techniques and domain specific knowledge on creativity,” *International Journal of Business and Society*, vol. 7, no. 2, pp. 79–107, 2006.

[25] J. Wiley, “Expertise as mental set: the effects of domain knowledge in creative problem solving.,” *Mem Cognit*, vol. 26, no. 4, pp. 716–730, Jul. 1998.

[26] G. Gigerenzer, “How to Make Cognitive Illusions Disappear: Beyond ‘Heuristics and Biases’,” *European Review of Social Psychology*, vol. 2, pp. 83–115, 1991.

[27] D. H. Jonassen, “Instructional design models for well-structured and III-structured problem-solving learning outcomes,” *Educational Technology Research and Development*, vol. 45, no. 1, pp. 65–94, 1997.

[28] A. Newell and H. Simon, *Human Problem Solving*. Prentice-Hall, Inc., 1972.

[29] H. A. Simon, *The Sciences of the Artificial*, 3rd ed. Cambridge, MA, USA: MIT Press, 1996.

[30] K. Dorst, “Design Problems and Design Paradoxes,” *Design Issues*, vol. 22, no. 3, pp. 4–17, 2006.

[31] P. Ralph, “Introducing an Empirical Model of Design,” in *Proceedings of The 6th Mediterranean Conference on Information Systems*, Limassol, Cyprus: AIS, 2011.

[32] G. Schraw, M. E. Dunkle, and L. D. Bendixen, “Cognitive processes in well-defined and ill-defined problem solving,” *Applied Cog. Psych.*, vol. 9, no. 6, pp. 523–538, Dec. 1995.

[33] T. Love, “Philosophy of Design: A Meta-theoretical Structure for Design Theory,” *Design Studies*, vol. 21, pp. 293–313, 2000.

[34] Goel, V. “Creative brains: designing in the real world." *Frontiers in Human Neuroscience*, vol. 8, pp. 1–14, 2014. V.

[35] J. Wirth, J. Künsting, and D. Leutner, “The impact of goal specificity and goal type on learning outcome and cognitive load,” *Computers in Human Behavior*, pp. 1–7, Jan. 2009.

[36] R. Vollmeyer and B. D. Burns, “Goal Specificity and Learning with a Hypermedia Program,” *Experimental Psychology*, vol. 49, no. 2, pp. 98–108, Apr. 2002.

[37] B. D. Burns and R. Vollmeyer, “Goal specificity effects on hypothesis testing in problem solving,” *Quarterly Journal of Experimental Psychology*, vol. 55, no. 1, pp. 241–261, Feb. 2002.

[38] J. Moss, K. Kotovsky, and J. Cagan, “The influence of open goals on the acquisition of problem-relevant information,” *Journal of Experimental Psychology: Learning, Memory, and Cognition*, vol. 33, no. 5, pp. 876–891, 2007.

[39] T. Ward, M. Patterson, and C. Sifonis, “The Role of Specificity and Abstraction in Creative Idea Generation,” *Creativity Research Journal*, vol. 16, no. 1, pp. 1–9, 2004.

[40] A. B. Butler, L. L. Scherer, and R. Reiter-Palmon, “Effects of Solution Elicitation Aids and Need for Cognition on the Generation of Solutions to Ill-Structured Problems,” *Creativity Research Journal*, vol. 15, no. 2, pp. 235–244, 2003.

[41] N. Cross, “Expertise in design: an overview,” *Design Studies*, vol. 25, no. 5, pp. 427–441, Sep. 2004.

[42] R. Guindon, “Knowledge exploited by experts during software system design,” *International Journal of Man-Machine Studies*, vol. 33, no. 3, pp. 279–304, Sep. 1990.

[43] D. G. Jansson and S. M. Smith, “Design fixation,” *Design Studies*, vol. 12, no. 1, pp. 3–11, 1991.

[44] A. T. Purcell and J. S. Gero, “Design and other types of fixation,” *Design Studies*, vol. 17, no. 4, pp. 363–383, 1996.

[45] R. Mohanani, P. Ralph, and B. Shreeve, “Requirements Fixation,” in *Proceedings of the 2014 International Conference on Software Engineering*, Hyderabad, India: ACM, 2014.

[46] C. Kruger and N. Cross, “Solution driven versus problem driven design: strategies and outcomes,” *Design Studies*, vol. 27, no. 5, pp. 527–548, Sep. 2006.

[47] Ö. Akin, “Expertise of the architect,” in *Expert Systems for Engineering Design*, M. Rychener, Ed. New York: Carnegie Mellon University, Engineering Design Research Center, 1988, pp. 171–196.

[48] N. Cross, K. Dorst, and N. Roozenburg, *Research in design thinking*. Delft University Press, 1992.

[49] P. Ralph, “The Sensemaking-Coevolution-Implementation Theory of Software Design,” *Science of Computer Programming*, in press.

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

[51] K. Dorst and N. Cross, “Creativity in the design process: Co-evolution of problem-solution,” *Design Studies*, vol. 22, pp. 425–437, Sep. 2001.

[52] B. Lawson, *Design in Mind*. Architectual Press, 1994.

[53] M. Basadur, S. J. Ellspermann, and G. W. Evans, “A new methodology for formulating ill-structured problems,” *Omega*, vol. 22, no. 6, pp. 627–645, 1994.

[54] M. D. Mumford, R. Reiter-Palmon, and M. R. Redmond, *Problem construction and cognition: Applying problem representations in ill-defined domains.* Ablex Publishing, 1994.

[55] C. E. Shalley, “Effects of productivity goals, creativity goals, and personal discretion on individual creativity,” *Journal of Applied Psychology*, 1991.