

The illusion of requirements in software development

Paul Ralph

Received: 8 July 2012 / Accepted: 28 July 2012
© Springer-Verlag London Limited 2012

Abstract This viewpoint explores the possibility that many software development projects may have no useful requirements. Specifically, for problems (e.g., knowledge worker burnout) with two completely different solutions (e.g., better tool support or hire more employees), an analyst may state a goal (e.g., decrease work hours) but more specific desiderata are contingent on the chosen solution. Furthermore, without fully exploring the design space, the designer cannot be sure whether there exists another approach, which would achieve the goal without any commonality with known approaches. In these situations of sparse requirements, analysts may misrepresent design decisions as requirements, creating an illusion of requirements in software development.

Keywords Requirement · Goal · Design · Fundamentals · Philosophy · Ontology · Epistemology

1 Introduction

It is widely accepted that understanding system requirements is important for software development project success [1, 2]. Put another way, it is widely acknowledged that failing to understand requirements is related to project failure [3, 4]. The idea that software artifacts generally have a set of discoverable, documentable requirements is entrenched in industry standards [5], development processes [2] and educational curricula [6, 7]. More broadly, requirements are a fundamental component of the Rational

Model of Design [8–10], the dominant view of how practitioners approach developing software and information systems. However, utilizing good requirements practices may not be a necessary or sufficient condition for project success [11, 12].

The assumption that software projects have discoverable, documentable requirements has motivated diverse literature on requirements engineering (RE), the process of identifying, analyzing, modeling, verifying and managing requirements. Major contributions include RE approaches (e.g., goal-oriented RE [13], user-centered RE [14]) and requirement types (e.g., non-functional requirements [15], early requirements [16]).

However, at least three software development projects that I have observed or participated in have produced not a single, meaningful requirement. Although they produced statements labeled as requirements, closer inspection revealed that they were something else—goals, design decisions, to-do- or wish-list items. Therefore, the purpose of this viewpoint is to explore the possibility of software projects with few or illusory requirements.

2 Some explicit assumptions

Different authors have defined “requirement” in different ways, alternatively as “a structural or behavioral property that a design object must possess” [17, p. 108], “a statement that identifies a capability or function that is needed by a system in order to satisfy its customer’s needs” ([18], p. 205) and “a property which must be exhibited in order to solve some problem in the real world” ([19], sec. 1.1).

Requirements are often differentiated from goals and design decisions. “A goal is an objective the system under consideration should achieve” [13, p. 2].

P. Ralph (✉)
Department of Management Science, Lancaster University,
Lancaster LA1 4YX, UK
e-mail: paul@paulralph.name

Where “goals describe the desired impacts of a design object on its environment” [17, p. 110], “requirements are usually understood as stating what a system is supposed to do, as opposed to how it should do it” ([20], p. 226). Similarly, Roman argued that “requirements specifications state the desired functional and performance characteristics of some component independent of any actual realization” ([21], p. 14). Likewise, IEEE Standard 830-1998 states that “a requirement specifies an externally visible function or attribute of a system [while] a design describes a particular subcomponent of a system and/or its interfaces with other subcomponents” [5, p. 9]. Understanding the relationships between goals, requirements and design decisions is the essence of the requirements traceability problem (cf. [22, 23]).

These varying definitions clarify the need for more explicit ontological assumptions underlying RE. For the purposes of this paper, I assume that a software development project produces an artifact called the design object having properties called features. Project participants or stakeholders may assign to the design object one or more goals—optative statements describing a change in the environment that the design object is desired to produce. A requirement is a feature of a design object that is necessary to achieve a goal. (While this is an unconventional way of defining requirement, it significantly simplifies explaining the two challenges advanced below, which do not hinge on this particular definition.) For example, suppose a development team is designing a website (the design object) to sell cameras online (the goal). The website will have many features (e.g., having the store name in a bold font) that do not substantially contribute to achieving its goal. However, other features (e.g., shopping cart) may be necessary conditions for achieving the goal—these necessary features are requirements. More precisely, given a goal g , a set of requirements R_g , may be defined as the set of all features necessary for a design object to achieve g . For the purposes of this paper, I do not distinguish between early- and late-stage requirements [16], hard/functional and soft/non-functional requirements [15] or requirements and constraints [17].

3 Ontological and epistemological challenges

Suppose exactly two design objects D_1 and D_2 will achieve a goal g . Features of both design objects are requirements as there is no way to achieve g without them (Table 1). Properties of D_1 but not D_2 are not requirements as g may be achieved without them (building D_2); properties of D_2 but not D_1 are not requirements as g may be achieved without them (building D_1). Properties of neither object are irrelevant. More generally, suppose g may be achieved by n design objects ($D_1, D_2, \dots, D_n$) each with features ($F_1, F_2,$

Table 1 Relationship between properties and requirements

	Property of D_1	Not a property of D_1
Property of D_2	Requirement	Feature
Not property of D_2	Feature	Irrelevant

$\dots F_n$). The set of requirements R_g may then be defined as the intersection of the properties ($R_g = F_1 \cap F_2 \cap \dots \cap F_n$).

Characterizing requirements as such reveals two assumptions. First, the existence of requirements entails overlap among design object features that may achieve a goal ($R_g = F_1 \cap F_2 \cap \dots \cap F_n \neq \emptyset$). In contrast, if the properties do not overlap ($R_g = F_1 \cap F_2 \cap \dots \cap F_n = \emptyset$), there are no requirements (an ontological problem; Fig. 1). Second, stating requirements assumes that all relevant design objects (or classes thereof) have been identified. If another solution, which will achieve g without any properties in common with known solutions, may exist, no properties of the existing solution sets are requirements (an epistemological problem). For example, suppose a company sets out to protect their network against malware, writing a requirements specification for a sophisticated antivirus system. During review, someone asks “Why don’t we just switch to Mac OS or Linux instead as they have fewer problems with viruses?” This reveals ostensible “requirements” as merely features of one solution.

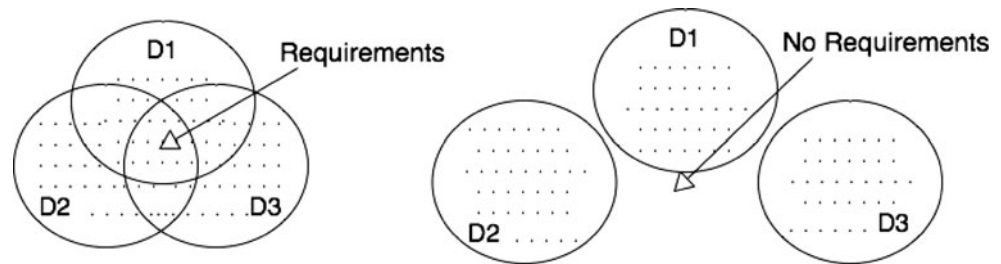
In summary, if the solution space is unknowable or lacks overlap, no requirements can be stated. This is not the same as acknowledging ambiguous, conflicting or incomplete requirements. Either lack of overlap or epistemic uncertainty produces a no-requirements scenario.

4 Limitations and proceeding without requirements

The ontological and epistemological challenges to the requirements concept described above are obviously somewhat simplified. First, the discussion takes a counterfactual approach to causality [24] when a probabilistic approach would be more appropriate (features may increase success probability rather than being “necessary conditions”). Second, the definition of requirement excludes other non-critical desiderata (wants, preferences), which may still be important. Third, identifying a needed feature may not always necessitate understanding the full scope of the design space—a highly credible informant’s comment that “the board will never approve this if it won’t work on an iPhone” may be sufficient to justify stating a requirement.

However, the illustration illuminates two fundamental problems. First, we can conceive of situations (e.g., knowledge worker burnout) where two completely

Fig. 1 Relationship between features (*dots*) and requirements. Features common to all design alternatives are requirements (*left*); where no features overlap, no requirements exist (*right*)



different approaches (e.g., better tool support or hire more employees) may achieve the same goal (e.g., decrease work hours)—in such cases of low solution overlap, few if any requirements can be stated. Second, without fully exploring the design space, we cannot be sure whether there exists another approach, which would achieve the goal without any features of known approaches.

This leaves the intellectual enterprise of RE research with two possibilities. One is that we should expect many software development projects to have few if any legitimate requirements, rendering many requirements engineering approaches ineffective or inappropriate in these contexts. This raises fundamental questions concerning how to adapt existing RE approaches to no-requirements scenarios, or how to proceed without requirements more generally.

Possibility two is that many existing RE approaches operationalize the requirements construct more generally—as desiderata, which may or may not be strictly necessary for success. This is problematic due to the strong denotation and connotation of the term. The word “requirement” denotes a thing that is compulsory. Listing requirements connotes certainty and unambiguousness. For example, when an analyst states that “cross-platform compatibility” is a requirement, novice developers and stakeholders unfamiliar with the challenges of RE are unlikely to interpret this as “the analyst hypothesizes that cross-platform compatibility will increase the probability that the system will achieve its objectives but we will not know for sure until the system is built, if ever” or “all of the plausible design candidates generated so far include cross-platform compatibility, but we have not fully explored the design space.” A wide variety of cognitive phenomena including anchoring bias [25], fixation [26] and confirmation bias [27] suggest that misrepresenting an incidental feature as a requirement will reduce exploration of the design space, curtailing innovation [28].

The argument of this paper may be challenged on at least two grounds. First, one may draw a distinction between mandatory and optional requirements. In response, I reiterate the psychological effect of mislabeling features as “requirements” (above) and suggest that the “optional requirement” label will likely increase confusion. Second, one may argue for conditional requirements, wherein some

features become requirements conditional on other features, for example, “given that the design artifact is a website, it must be HTML5-compliant.” However, this creates a reductionist spiral where virtually all design decisions may be recast as conditional requirements, undermining the distinction between requirements and design decisions and further confusing developers and curtailing innovation.

In conclusion, this paper presents two novel challenges to RE. The ontological challenge posits that where many plausible approaches to achieving a goal are evident, there may be insufficient overlap between approaches to form requirements. The epistemological challenge posits that while all plausible approaches may have sufficient overlap to state requirements, one cannot know that unless all approaches are identified and one is somehow sure that none have been missed. I have formulated these challenges to stimulate debate on fundamental properties and assumptions of requirements in theory and practice. They raise important questions about possible requirement-sparse environments and the implications of goals, features, conjectures and design decisions mislabeled as requirements.

References

- Boehm B (1991) Software risk management: principles and practices. *IEEE Softw* 8(1):32–41
- Jacobson I, Booch G, Rumbaugh J (1999) The unified software development process. Addison-Wesley Longman Publishing Co., Inc., Boston
- Ewusi-Mensah K (2003) Software development failures. MIT Press, Cambridge
- Standish Group (2009) CHAOS summary 2009. Boston, MA, USA http://www.standishgroup.com/newsroom/chaos_2009.php
- IEEE (1998) IEEE Standard 830-1998: recommended practice for software requirements specifications. <http://standards.ieee.org/findstds/standard/830-1998.html>
- Topi H, Valacich JS, Wright RT, Kaiser KM, Nunamaker JF, Sipior JC, Vreede GJd (2010) IS 2010: curriculum guidelines for undergraduate degree programs in information systems. *Communications of association for information systems* 26:Article 18. <http://aisel.aisnet.org/cais/vol26/iss1/18/>
- Joint Task Force on Computing Curricula (2004) Software engineering 2004: curriculum guidelines for undergraduate degree programs in software engineering. In: Díaz-Herrera JL,

- Hilburn TB (eds). <http://sites.computer.org/ccse/SE2004Volume.pdf>
8. Simon HA (1996) The sciences of the artificial, 3rd edn. MIT Press, Cambridge
 9. Brooks FP (2010) The design of design: essays from a computer scientist. Addison-Wesley Professional, Reading
 10. Ralph P (2011) Introducing an empirical model of design. Paper presented at the 6th Mediterranean conference on information systems, Limassol, Cyprus, 3–5 Sep
 11. Davis AM, Zowghi D (2006) Good requirements practices are neither necessary nor sufficient. *Requir Eng* 11(1):1–3
 12. Shenhar AJ, Dvir D, Levy O, Maltz AC (2001) Project success: a multidimensional strategic concept. *Long Range Plan* 34(6):699–725
 13. van Lamsweerde A (2001) A goal-oriented requirements engineering: a guided tour. In: Proceedings of the fifth IEEE international symposium on requirements engineering, Aug, pp 249–262
 14. Sutcliffe A, Thew S, Jarvis P (2011) Experience with user-centred requirements engineering. *Requir Eng* 16(4):267–280
 15. Chung L, Nixon BA, Yu E (2000) Non-functional requirements in software engineering. Kluwer international series in software engineering, vol 5. Springer, Berlin
 16. Fuxman A, Liu L, Mylopoulos J, Pistore M, Roveri M, Traverso P (2004) Specifying and analyzing early requirements in tropos. *Requir Eng* 9(2):132–150
 17. Ralph P, Wand Y (2009) A proposal for a formal definition of the design concept. In: Lyytinen K, Loucopoulos P, Mylopoulos J, Robinson W (eds) Design requirements engineering: a ten-year perspective. Lecture notes on business information processing, vol 14. Springer, Berlin, pp 103–136
 18. Bahill AT, Dean FF (2009) Discovering system requirements. In: Sage AP, Rouse WB (eds) Handbook of systems engineering and management, 2nd edn. Wiley, New York, pp 205–266
 19. Bourque P, Dupuis R (eds) (2004) Guide to the software engineering body of knowledge (SWEBOK). IEEE Computer Society Press, Silver Spring
 20. Yu E (1997) Towards modelling and reasoning support for early-phase requirements engineering. In: Proceedings of the third IEEE international symposium on requirements engineering, pp 226–235
 21. Roman G-C (1985) A taxonomy of current issues in requirements engineering. *Computer* 18(4):14–23
 22. Gotel O, Finkelstein A (1994) An analysis of the requirements traceability problem. In: First international conference on requirements engineering, Colorado Springs, CO, USA, IEEE Computer Society Press, pp 94–101
 23. Ramesh B, Jarke M (2001) Toward reference models for requirements traceability. *IEEE Trans Softw Eng* 27(1):58–93
 24. Gregor S (2006) The nature of theory in information systems. *MIS Q* 30(3):611–642
 25. Parsons J, Saunders C (2004) Cognitive heuristics in software engineering: applying and extending anchoring and adjustment to artifact reuse. *IEEE Trans Softw Eng* 30:873–888
 26. Jansson DG, Smith SM (1991) Design fixation. *Des Stud* 12(1):3–11
 27. Oswald ME, Grosjean S (2004) Confirmation bias. In: Pohl RF (ed) Cognitive illusions: a handbook on fallacies and biases in thinking, judgement and memory. Psychology Press, Hove, pp 79–96
 28. Ralph P (2011) Toward a theory of debiasing software development. In: Wrycza S (ed) Research in systems analysis and design: models and methods. In 4th SIGSAND/PLAIS EuroSymposium 2011. LNBIP, vol 93. Springer, Gdansk, Poland. pp 92–105