The Brave New World of Design Requirements: Four Key Principles

Matthias Jarke[1], Pericles Loucopoulos[2], Kalle Lyytinen[3], John Mylopoulos[4], and William Robinson[5]

[1] RWTH Aachen University, Germany
[2] University of Loughborough, U.K.
[3] Case Western Reserve University, USA
[4] University of Toronto, Canada
[5] Georgia State University, USA

Abstract. Despite its undoubted success, Requirements Engineering (RE) needs a better alignment between its research focus and its grounding in practical needs as these needs have changed significantly recently. We explore changes in the environment, targets, and the process of requirements engineering (RE) that influence the nature of fundamental RE questions. Based on these explorations we propose four key principles that underlie current requirements processes: (1) intertwining of requirements with implementation and organizational contexts, (2) dynamic evolution of requirements, (3) architectures as a critical stabilizing force, and (4) high levels of design complexity. We make recommendations to refocus RE research agenda as to meet new challenges based on the review and analysis of these four key themes. We note several managerial and practical implications.

1 Introduction

The genesis of Requirements Engineering (RE) research around 30 years ago was motivated by practitioner who noticed the urgent need for disciplined RE in large software projects [1, 2]. Much of RE research since then has focused on artifacts that help capture, share, represent, analyze, negotiate, and prioritize requirements as a basis for design decisions and interventions (for recent reviews see e,g, [21, 22, 23]). This is evidenced by the volume and impact of a plethora of requirements topics papers published in top the level software engineering and computing conferences and journals (for a survey see [4, 23] . Due to its practical origins it is not surprising that some of its findings, like the use of business and system modeling (ERD, use cases), risk driven methodologies, structured requirements documents, and simple requirements tracing, have found their way into practice[3].

Yet, the environment in which RE is practiced has changed dramatically. Partly, this is due to increases in computing speed, lowering of computing cost, and advances in functionality, which has made software common in all walks-of-life. Partly, this is due to the changes in technological, task, and organizational environments where

B. Pernici (Ed.): CAiSE 2010, LNCS 6051, pp. 470–482, 2010.

software is either produced or deployed. The field's focus and scope has shifted from engineering of individual systems and components towards the generation and adaptation of software intensive ecosystems. Accordingly, a term *design requirements* rather than *software requirements* is needed as an inclusive term to denote all common sets of requirements issues within these ecosystems that need to be addressed at the crossroads of business development, software engineering, and industrial design. This shift has created a strong need to re-think and re-align RE practices to meet the new challenges. Both academia and industry need to understand more deeply issues that underlie current RE and address associated challenges. We posit that answers cannot come just from doing more of the same—i.e., traditional RE research focusing on notations and tools alone. The research scope of RE has to become more interdisciplinary, and it needs to carefully evaluate some of its critical assumptions. This essay aims to identify some of these challenges based on a detailed field and content analysis of extensive expert discussions and feedback on current RE practices [4]. Based on these explorations we propose four principles that underlie future requirements processes and influence their successful resolution. Finally, new research challenges and practical implications are identified.

2 The Changing Nature of Requirements

Current RE landscape is marked by new challenges and opportunities [3]. Its environment, target technologies, processes, and fundamental problems have undergone a tectonic shift. The environment of RE now involves elements that were not there 20 or 30 years ago [5]. First, the economics of RE has changed. Large systems like ERP systems need more rigorous ROI, but at the same time horizons for ROI have reduced to 18-20 months thanks to massive reuse and COTS deployment. Second, there is practically no green-field software development; RE acts more like the Roman god of gates—Janus, with one face, looking at new business and technological challenges and opportunities and another face gazing at existing (technological, organizational, social and political) environments and resource sets. Third, the scaling towards software intensive ecosystems results in exceedingly complex and non-linear dynamic dependencies between system components and their natural, technical and social environment—"green IT" being just one of the latest buzzword that characterize this trend. Fourth, speed and agility, time to market, low-cost iterative, or even end-user development have become critical factors leading to search for new design trade-offs between efficiency, openness, and flexibility. This has also increased outsourcing and off-shoring, which requires disciplined evolution and management of explicit specifications as a basis for delegation and framing design problems. Fifth, RE now cuts across industrial design (e.g., pervasive applications), media design (e.g., e-commerce and media applications), interaction design (e.g., new modalities of interaction in mobile computing, telematics etc.), and business process design (e.g., open business platforms), and regulatory and juridical issues (e.g. management and control of multiple licenses in software platforms). Overall, design requirements need to capture and coordinate increasingly diverging and dynamic needs of users and other stakeholders during the evolution of a product, a service, or a platform.

Table 1. Summary of Critical Design Requirements Issues (adopted from [4])

	Critical Requirements Issues	Brief description
Target platform	**Business process focus**	Requirements focus simultaneously on the business process, and requirements for technological artifact driven by that business process.
	Systems transparency	Requirements involve the demand for a seamless user experience across applications.
	Integration focus	Requirements focus on integrating applications rather than development of new ones (i.e., less green-field development).
	Packaged software	Purchase of commercial off-the-shelf (COTS) software and components rather than internal development. This has lead to market driven vendor-led requirements and knowledge brokering.
Development process	**Distributed requirements**	In addition to increasingly diverse stakeholders, requirements processes are distributed across multiple organizations, groups and social worlds globally.
	Centrality of architecture	Architectural considerations and associated evolutionary paths take a central role and drive business, product and application requirements.
	Layers of requirements	Requirements need to be iteratively developed across multiple levels of abstraction, design focus, or temporal horizon.
	Interdependent Complexity	While some forms of design complexity have been reduced (loosely coupled components), the overall interaction complexity of the design ecology has risen enormously.
	Fluidity of design	Requirements process must accommodate the need for continued evolution of the artifact and the solution after initial implementation.

What are the critical issues that emerge during RE in this brave new world? Table 1 presents nine critical issues that were solicited in a field study in Fortune 500 companies [4]. These are divided into the changing nature of the object of RE (target platform), and the process of RE (development process). Overall, these issues resonate well with the debate Simon engages in his design classic, The *Sciences of the Artificial* [6]. On the one hand, software designs resemble increasingly continuous and dynamic searches for satisficing solutions—not an optimized and fixed solution at one time point conforming to a fixed set of requirements. On the other hand, they go beyond Simon's original model in that they emphasize sense-making in shifting and complex environments [7] and associated problem framing over focused problem solving in a bounded context. To wit, these changes in the environment and the object and process of RE are changing the three classic RE problems as follows:

First, the *design requirements problem* already pointed out in [1, 2] can be stated as follows: *What is the emergent behavior and dynamics of the software artifact and its environment in their evolutionary trajectory?* Now users, designers and other stakeholders need to ask: will the system continue to satisfy emergent goals, and what

those goals could be expected to be during the artifact's life-time; in contrast to the older problem: What are the goals of the system and what is expected to do

Second, the *specification problem* can be stated as follows: *How can designers anticipate and represent the emergent behaviors of the system and its components and how does the resulting system behavior conform and relate to the emerging environments and the notations used to represent and predict it?* Accordingly, now designers need to ask how they can represent, communicate and analyzed increasingly complex and dynamics systems and their emergent requirements in contrast to the older problem: how to represent the system components, their relationships and behaviors and guarantee that they meet functional and non-functional requirements?

Third, the *predictability problem of designs can be stated as follows: How does the artifact and its behavior change the environment as to make our predictions of the system behaviors faithful?* In other words, now designers need to attend more to the dynamic composition of the system and environment and do they together differ from the environment alone, and can he/she accordingly predict faithfully the impact of the system on the environment, and vice versa? This is a different problem from those faced earlier where the system was assumed to *not* affect the environment, or the environment the system, but in some rare cases [14].

3 Four Requirements Principles

Past RE research has been informed by a few key principles such as those of: separation of what, why and how [1], information hiding (see e.g. [2], and the principle of abstraction (see e.g. [22]). These principles reduced design complexity by localizing design decisions. They also helped reduce their interference and analyze and predict system behaviors and structure from specific viewpoints [21, 22].

But, what are the key principles that will underlie successful design and RE in the brave new world we face? What principles will help us address the design requirements problem, the specification problem, and the predictability problem in the new context? We propose next four principles that were gleaned from our analysis of expert opinions and related discussions, and a review of the literature dealing with design dynamism and complexity. These four principles are:

Intertwine Requirements and Contexts: The necessity to *intertwine* design and requirements with design and implementation across multiple dimensions.

Rationale: This principle observes the design requirements problem and the new demands for understanding evolutionary trajectories.

Evolve Designs and Ecologies: The necessity to view design and design processes as *evolving* elements in an *ecology*.

Rationale: This principle observes the new design requirements and specification problem and increasing demands to analyze evolutionary principles of a large set of heterogeneous elements comprising the requirements space.

Manage through architectures: *Architectures* have a critical role of as enablers and constraints in the constant creation and shaping of design ecologies.

Rationale: This principle recognizes the specification problem and the shifting focus towards ecologies where increased emphasis must be placed on antecedent factors that affect the organization and evolution of the ecology.

Recognize complexity: The heightened *interaction complexity* of requirements processes demands new ways to approach design problems and manage requirements.

Rationale: This principle recognizes the predictability problems and that the new interaction complexity requires designers to heed on the external relationships of the software and their evolution as reflected in the requirements.

We will next describe each principle in more detail in terms of the content of the principle, related research questions, and emerging research.

3.1 Intertwine Requirements and Contexts

The debate about the role of requirements is as old as the field itself. Whilst a rough consensus has been reached that requirements are a pre-requisite for downstream development, there is a great deal of controversy on how 'problem' and 'solution' spaces interplay during the evolution. One school of thought regards the influence of implementation on requirements as being harmful [8]. They argue that understanding the system's context, such as its organizational and social factors and goals can provide a sufficient set of functional and non-functional requirements, which can then be mapped onto appropriate implementation models. This school regards requirements as the "downward" bridge between the 'subject' and 'system' worlds by assuming that there exists a high degree of stability on business, organizational, and community goals. An opposing view stresses the need for revisiting requirements as implementation progresses and emphasizes the dynamics and intertwining of these activities [9].

The review of existing practice [4] shows that implementation and its requirements specification are now necessarily intertwined. In fact, many requirements emerge from existing solution spaces. Accordingly, the concept needs to be extended to the whole system context. In addition, the salient factors shaping RE seem to be innovation and effective differentiation. The interplay between the two worlds has thus become more intricate, complex, dynamic, and generative. In these innovation-driven settings, requirements become part of both the business solution and the system solution, and they constantly bridge new solutions to organizational and societal problems. The evolving designs need to reduce the distance between a problem and a solution through novel and dynamic thinking, acting, and innovating. In such a design-thinking culture, design requirements become increasingly central and need to be understood as part of a multi-system, socio-technical ecology, which drives organizational innovation. Therefore, software requirements need to be dynamically situated between these spaces as they intertwine organizational and implementation considerations, providing leverage to influence both.

Due to this constant intertwining some systems and ecologies may be reaching a practical *world-model limit*. While prior design efforts could rely on an adequate, stable, world-model as the basis for specifying nearly stable software designs, now, software must be agile—rapidly evolving to meet changing needs. The level of stability of the world-assumptions is less limited in context-aware, customer-focused applications. The unavoidable intertwining between requirements and contexts will make

designers constantly seek correspondence between the models in software and its world context. Only software embedding an adequate, flexible, and evolvable world-model is likely to survive. The idea of evolutionary software and variability selection aims to partially meet this need. But, little attention has been given to the challenge of formulating evolutionary world models that form the basis for the necessarily simplifying, but evolving model assumptions in the software. Software developers must thus monitor and evolve their understandings of the world, and sustain an adequate correspondence between the world and the modeled world.

Overall, RE processes face a new kind of uncertainty that goes beyond traditional RE uncertainty characterized by: (1) requirements identity (knowing requirements), (2) requirements volatility, and (3) requirements complexity [10]. In addition, designers need to devote their efforts: (1) *requirements fidelity uncertainty*, which denotes the uncertainty about the level of intertwining between the world and the software model. Examples of techniques that help mitigate fidelity uncertainty are exception and event-based analysis; software tailoring and user-based development, and case-based learning; and (2) *requirement monitoring uncertainty*, which denotes uncertainty of the level and mode of observation, and analysis necessary to assess the world, the model, the requirements, and their alignment. Examples of monitoring include ethnographic methods, business activity monitoring (BAM), and software instrumentation. These two new levels of uncertainty highlight the need for increased run-time monitoring to maintain the fidelity of the world-model intertwining with requirements.

3.2 Evolve Designs and Ecologies

Meeting stakeholder needs is fundamental to requirements activity. When requirements increasingly intertwine with organizational and implementation concerns, they will constantly and non-linearly evolve as part of the "ecology". Traditional causes of software evolution have been classified into: (1) the software, (2) the documentation, (3) the properties of the software, and (4) customer-experienced functionality[11]. Evolution has been studied mainly as a software design problem and it has been addressed by improving methodological support (e.g., how can development activities most effectively incorporate evolution?) and its management (e.g., how one can one record and trace software releases or link the code to changing domain knowledge?). Now, the reality of an ever incomplete and evolving design needs to be addressed. We need to ask: what are the principles that guide developers evolve 'incomplete designs' so that they remain functionally adequate, but offer flexibility? What are appropriate co-evolutionary design methods to achieve these goals? How does one determine the impact of co-evolutionary design change?

Activities in open source development, such as inter-project merging and the creation of new software artifacts, for example, compound the need for new frameworks to cope with requirements evolution. Another example is agile methods and scenario based modeling which offer a means to better cope with the fast paced evolution of requirements ecologies [12]. Likewise research into co-evolution and co-design [13] has addressed drivers and interaction laws that deal with the intertwining of contexts and requirements. Yet, such studies are in early stages, and agile methods only deal with micro-level evolution of local tasks, but ignore their recursive nature as the

change propagates across the higher levels of architectures and systems. Here we clearly need longitudinal studies of the dynamics of software ecologies and how different causes ranging from technological, user level learning, organizational policies, market based, and regulatory changes intertwine and generate new evolutionary paths.

3.3 Managing through Architectures

Architecture is concerned with blueprints that connect high level organizational, business, or implementation considerations with a long-term evolutionary perspective. Organizations now increasingly conform to business or information architectures that provide stability, scale and change to their data, business rules and decision models. Designers have, for some time relied on implementation architectures while evolving their designs. In its variety of forms an architecture provides the stepping-stone necessary to understand and evolve any system functionality across different domains. Through release planning, requirements play a central role in systems evolution, where architectures provide "nearly" fixed points of reference to moderate, constrain and enable evolution. Such dependence on architecture is inherent in Lehman's law that "the incremental growth (growth rate trend) of evolutionary software systems is constrained by the need to maintain familiarity" [14].Architectural dependencies arise also in approaches like IKIWISI (I'll Know It When I See It), and COTS (Commercial off-the-shelf) based software deployment [15].

Though RE research in the past has paid significant attention to 'software architectures', it offers limited insight to the role of architectures in the new RE terrain. Many of the past studies focus on organizing sets of design elements and their components in the context of a single system. Accordingly, they approach architectural design akin to generating a blueprint for a single house. In the context of dynamic software ecologies, such an analogue fails. In the brave new world, managing through architectures is about generating and evaluating multiple and multifaceted plans similar to urban planning. Like urban planning blueprints, architectural models provide the key artifact for coordinating components, functionalities, and their evolution. As in urban planning, the architectures in RE may have alternative and overlaying *variation points* that influence the evolution of the software ecology. As in urban planning, architectures embody specific business models or design visions, and come in different forms in different design contexts. By doing so, they integrate the needs of multiple stakeholder groups with varying roles. Finally, like urban plans they involve the same level of complexity and interdependencies. Therefore, in the new RE, we see an increasing need to understand the variation between types of architectural models needed and how they relate to specific families of systems and their ecologies.

Recent attempts to deal with architectural considerations include the transfer of industrial concepts, such as product lines to the software field [16]. They help manage and cope with continuous and rapid change in software by defining the scale, scope, and direction of its variance and selection. Other architectural models such as business architectures [3] help stakeholders to envision the impact of proposed changes on business by providing contextual information that allows for selecting variation points across multiple stakeholders. Yet, many research challenges remain in taking the advantage of the idea of architecture: How do architectures influence the evolution of requirements and their identification? What is the nature of requirements discovery

and elicitation under varying architectural principles? Is it possible or even desirable to construct a single common ontology of business, information and technology architectures? How to relate different architectural presentations and reason around them? How can architectures help in flexible composition of systems and ecologies?

3.4 Recognize and Mitigate against Design Complexity

Complexity is borne out of the existence of multiple uncertain futures that relate to software and their evolving ecologies [17]. A mix of human, social, political, economical, technological, and organizational factors has a bearing on the level of complexity associated with RE. Overall a new sort of interaction or systemic complexity[1] needs to be reckoned and managed during the RE, where design becomes a problem solving and framing process with inherent uncertainties driven by the partial unknowns. Dealing with such design complexity impacts two areas of RE: (1) the strategic decision-making in generating and selecting requirements, and understanding their impact on the ecology; and (2) selecting tractable design-approaches that make complex system designs possible. The former is the concern of how to relate complexity with stakeholders within their 'subject' world, whereas the latter influences behaviors within the 'design' world.

In the brave new world of RE the implementation of requirements impacts not only on the technical systems, but also on their organizational and social settings that increases interaction complexity. In addition, the increased variety of requirements that emanate from diverse communities need to be negotiated, evaluated, and selected compounding the complexity. Qualitative and often structural conceptual models like goal or business models, whilst rich and useful in representation and analysis for design are less helpful for stakeholder evaluation and understanding the interaction complexity. Due to the design complexity, it is also difficult for some stakeholders to visualize and understand the system's behavior. It is tempting to think that "stakeholders understand a description when they don't really understand it at all". Therefore, many questions remain poorly understood concerning design complexity: What is the nature of design complexity and increased interaction complexity, and how can we identify, analyze and measure it?

One way of coping with design complexity is through architectural designs and control that allow 'nearly' decomposable system designs. This mitigates complexity by ensuring that interactions among components are weak, though not negligible. Thus, designing a nearly decomposable system in the face of uncertain requirements becomes a difficult satisficing problem. Perhaps, not surprisingly, a new look at design methodologies can play the central role here. An emergent design methodology will be an improvement over conventional *a priori* methodologies. Open source systems, following nontraditional methodologies, for example evolve systems faster than traditional life cycle and requirements driven development approaches [18]. The co-design and co-evolution of the system and its stakeholders seems to play here a pivotal role, as does the fact that "open source systems entail internal architectures with orthogonal features, sub-systems, or modules, as well as external system release

[1] This should not be confused with computational complexity as defined by well-known complexity notions like NP hard problems.

Table 2. Four key requirements principles

Principle	Description	Rationale
Intertwine Requirements and Contexts	Requirements are interdependent with their social and technical contexts. As boundary objects in the intersection of the technical and social domains, design requirements seek to constantly resolve the gap between problems and solutions. Specification and implementation intertwining is long recognized, but the social context and specification intertwining is growing in importance. *RE Problem addressed: design requirements problem.*	Intertwining between business, organizational, community context and requirements is as important as it is between requirements and software. *New RE Issues : Fluidity of designs, , Business process focus; Integration focus, Distribution of requirements*
Evolve Designs and Ecologies	Design ideas and artifacts evolve, from stakeholder preferences to the implementations. Evolution needs to be managed through selectively freezing some aspects while changing other aspects thus allowing increased variation, dynamic selection and diffusion of structures and behaviors. *RE Problem addressed: design requirements problem, specification problem.*	Everything evolves, but at different rates. Design around relatively fixed evolutionary paths that allow for increased but controlled variation and effective selection and diffusion. *New RE issues : Fluidity of designs, Layers of requirements, Distribution of requirements, Packaged software*
Manage through architectures	Architecture is the least evolving and most widely referenced anchor of the design, be it business architecture, or implementation architecture. *RE Problem addressed: specification problem*	If well-designed, the architecture (business or software) evolves slowly, and influences and interacts with many requirements. We know poorly however how architectures shape, allow and constrain evolution. *New RE issues : Interdependent complexity, business process focus, Centrality of architecture*
Recognize and mitigate against design complexity	The necessity to consider simultaneously a large number of issues and their non-linear interactions during design raises design and requirements complexity beyond what a single designer can understand or visualize. *RE Problem addressed: predictability problem.*	Historically, tools aided a single designer or a small group in design decision making. New design tools need to be extended to monitor and analyze the dynamic design evolution, highlighting its trajectory and helping negotiate at the team and community level. *New RE issues : Interdependent Complexity, Fluidity of designs, Business process focus, Layers of requirements*

architectures that span multiple deployment platforms"[18]. A better understanding of complexity can be obtained, if system descriptions are tested against concepts familiar to stakeholders, and multiple scenarios are played out by fixing key parameters as proposed by the architecture. Experimenting with different scenarios has proved a powerful means for discovering and refining requirements with heightened complexity [19]. Still, we need to examine: What types of interdependencies influence and affect system change and create higher levels of design complexity? How can architectural models be exploited mitigate against design complexity, and to what extent they are a cause of it?

3.5 Summary of Four Key Design Principles

The four principles are summarized in Table 2 together with the rationale for using each principle, as well as what critical RE issues motivate each. We do not claim that they all apply in all design contexts, or even that some will apply in all contexts. In contrast, when e.g. designs are expected to be fluid, when they integrate with business process, or involve significant distribution of requirements we can expect the principle of intertwining of requirements and contexts to be instrumental. Finally, these are similar to *threshold* indicators as used in claiming that abstraction is important, when the size and the number of dependencies within the software system goes beyond a certain threshold.

4 Implications

Over its thirty-year history, the idea of design requirements has changed from single, static and fixed-point statements of desirable system properties into dynamic and evolving rationales that mediate change between the dynamic business environments and the design and implementation worlds. As Fred Brooks noted in the Dagstuhl workshop: "Design is not about solving fixed problems; it is constant framing of solution spaces". This evolution has now probably reached a new turning point characterized by unprecedented scale, complexity, and dynamism. This calls for new ways to think about requirements and their role in the design. Like earlier turning points, such as the software crisis in the 1970s, it will demand a resolute and careful intellectual response. The four requirements principles discussed have numerous implications for research of which only a tip of the iceberg has been addressed. There are also multiple implications for RE practice both at the management and at the engineering level. For the sake of brevity, we discuss below three closely related practical strategies being pursued and exposed by the four principles.

Service orientation and task distribution: The life cycle of systems must now be aligned more closely with the business process lifecycle. Service oriented architectures, possibly combined with model-driven code and test generation, are now reasonably well established at the programming level to do the job. The situation is quite different, however, at the level of business services, despite their ongoing standardization. The decomposition of monolithic business process systems into freely configurable business services has turned out to be far more complex task than expected due to the need to make business semantics explicit that were hitherto hidden in the code.

This is, however, not only true for runtime service configurations, but also when outsourcing and especially when off-shoring. The design challenge is how to tackle the domain of business semantics, which is often culture dependent. Intercultural competencies become a must for requirements engineers in such settings, where pilot cases can be a promising means to establish cultural understanding. Legal aspects are also forming an increasingly important aspect, not only in terms of how to protect intellectual property (IP)—what should I *not* offshore, if IP handling is doubtful? —but also in terms of protecting oneself against being sued by customers due to imprecise contractual agreements or.e.g not honoring open source licenses in some parts of the code.

Importance of the Edge: In the increasingly complex environments the distinction between users and developers is vanishing, and the user networks and the developer networks are becoming increasingly fuzzy and intermingled. Many contributors to system designs, and even implementations are no longer located in the kernel, but at the network edge. This situation— characterized as an evolution from *user* to *citizen*— enables greater diversity of system variants and system uses, especially in cross-cultural environments. Web-based social networks have proven to be the infrastructure of choice for such settings, and quick and dirty testing of incremental changes in limited market portions is the necessary requirements testing strategy. Accordingly, design incentives now go beyond monetary ones. *Bricoleurs* at the edge must be harnessed to contribute to usability and functionality beyond the initial enthusiasm. Thus, transparency, accountability, and maintenance of a core vision in ecologies become more important. Requirements traceability within such evolutionary processes often involves runtime monitoring towards the requirements. Finally, new design goals enter the stage. For example, industrial design ideals such as innovativeness or aesthetics of the user experience play often a larger role than pure functionality.

Capability-based platforms: For the past twenty years, the dominant trend design has been business process modeling and optimization that go together within monolithic COTS, or software product lines. As noted, we witness a move towards capability-based evolutionary platforms defined by consumption and production networks. Such networks need to analyze or define their core capabilities, and seek opportunities (often initiated from the edge, as noted above) to exploit for market or process innovations in a speedy and flexible manner. Capability-based platforms define core capabilities that can be competitively delivered with and by the networks. These platforms also hold the networks together. By fixing 'core' requirements and the related architecture for the efficient core implementation, the platform designer makes bets on assumptions about the speed of change related to different requirements sets associated with the platform (and the network). For example, platform strategies have been a critical success factor in the automotive industry, but this advantage can turn into a deadly trap, when economic considerations mandate that the company must be split in a manner orthogonal to the original platform and related network. Software platform strategies go thus beyond the idea of product lines as a means to manage efficiently variability in software. They need to consider distinctions between core processes (supported in the platform) and context processes (around it). These distinctions must be based on a careful analysis of market power and network strength, and anticipate technology evolution in the underlying technological standards and architectures.

To sum up, the good news is that the RE has never been more important and its criticality will continue to grow. The bad news is that RE is a different beast now. Accordingly, we need to consider RE in new ways that take into consideration the need for the alignments with business processes, the need to understand core capabilities, the need to ensure legal protection or, the need to create user buy-in, or the need to minimize training costs. In consequence, we need to expand RE research into new directions —including complexity science, industrial design, organization design, and economics- and engage these fields in a honest intellectual exchange why and how design requirements matter in the design of complex software and world.

Acknowledgments

We thank Sean Hansen, Nicholas Berente, Dominik Schmitz, and Anna Glukhova for helping to organize the workshops, and the workshop participants for inspiring discussions and Sol Greenspan for constructive comments. This research was in part funded by National Science Foundation's "Science of Design" initiative, Grant Number: CCF0613606, by DFG project CONTICI and by BMBF project ZAMOMO.

References

1. Ross, D.T., Schoman Jr., K.E.: Structured analysis for requirements definition. Transactions on Software Engineering SE-3(1), 6–15 (1977)
2. Frederick, J., Brooks, P.: The Mythical Man-Month. Addison Wesley, Reading (1995)
3. Lyytinen, K., et al.: Design Requirements Engineering: A Ten-Year Perspective. In: Design Requirements Workshop, Cleveland, OH, USA, June 3-6 (2007); Revised and Invited Papers, p. 495. Springer, Heidelberg (2009)
4. Hansen, S., et al.: Principles of Requirements Processes at the Dawn of 21st Century. Ingénierie des Systèmes d'Information 13(1), 9–35 (2008)
5. Schuler, D., Namioka, A.: Participatory design. In: Proc. Lawrence Erlbaum Assoc. (1993)
6. Simon, H.: The Sciences of the Artificial. MIT Press, Cambridge (1996)
7. Schon, D.: The reflective practitioner: How professionals think in action. Basic Books, New York (1983)
8. Bowen, J.P., Hinchey, M.G.: Ten Commandments of Formal Methods (1995)
9. Swartout, W., Balzer, R.: On the inevitable intertwining of specification and implementation. CACM 25(7), 438–440 (1982)
10. Mathiassen, L., et al.: A Contigency Model for Requirements Development. Journal of the Association for Information Systems 8(11), 569–597 (2007)
11. Chapin, N., et al.: Types of software evolution and software maintenance. Journal of Software Maintenance and Evolution Research and Practice 13(1), 3–30 (2001)
12. Cockburn, A.: Agile Software Development. Addison-Wesley, Reading (2002)
13. Berger, C., et al.: Customers as co-designers. Manufacturing Engineer 82(4), 42–45 (2003)
14. Lehman, M.M.: Software Evolution. Encyclopedia of Software Engineering 2, 1507–1513 (2002)
15. Nuseibeh, B.: Weaving together requirements and architectures. Computer 34(3), 115–119 (2001)
16. Pohl, K., et al.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer, Heidelberg (2005)

17. Godet, M.: Scenarios and strategic management. Butterworth-Heinemann, Butterworths (1987)
18. Scacchi, W.: Understanding Open Source Software Evolution. In: Software Evolution and Feedback, Theory and Practice. Wiley, New York (2006)
19. Carroll, J.M.: Scenarios and design cognition. In: Proceedings of IEEE Joint International Conference on Requirements Engineering, pp. 3–5 (2002)
20. Glasser, B., Strauss, A.: The development of grounded theory. Alden, Chicago (1967)
21. Zave, P.: Classification of research efforts in requirements engineering. ACM Comp. Surv. 29(4), 315–321 (1997)
22. van Lamsweerde, A.: Requirements Engineering in the Year 00: A Research Perspective. In: Proceedings of the 22nd International Conference on Software Engineering, pp. 5–19 (2000)
23. Cheng, B., Atlee, J.: Current and Future Research Directions in Requirements Engineering. In: Lyytinen, K., Loucopoulos, P., Mylopoulos, J., Robinson, W. (eds.) Design Requirements Engineering – A Ten-Year Perspective. LNBIP, vol. 14. Springer, Heidelberg (2009)

Appendix: Data Collection and Analysis

The new requirements engineering challenges reported in this essay were discussed in a series of workshops, designed to accomplish three objectives: (1) engage separate research communities in a dialogue, (2) strengthen design science principles, and (3) open new vistas for the research on design requirements. The first workshop was held in the United States in June 2007, while the second workshop was held in October 2008 in Germany (see [3] for more detailed description). The discussions were recorded in the workshop Wikis (see e.g. http://weatherhead.case.edu/requirements; http://www.dagstuhl.de/Materials/index.en.phtml?08412). In addition we undertook a field study to understand the perspectives of practitioners on successful requirements practices, anticipated developments, and new challenges in design environments [5].