Research Directions in Requirements Engineering

Betty H.C. Cheng
Michigan State University
3115 Engineering Building
East Lansing, Michigan 48824 USA
chengb@cse.msu.edu

Joanne M. Atlee
University of Waterloo
200 University Ave. West
Waterloo, Ontario N2L 3G1 CANADA
jmatlee@uwaterloo.ca

Abstract

This paper reviews the current state of the art of requirements engineering (RE) research and identifies RE research challenges for future systems. First, the paper overviews the highlights of RE research over the past two decades; the research is considered with respect to requirements technologie, including notations and methodologies, developed to address specific RE tasks, such as elicitation, modeling, and analysis. Such a review enables us to identify mature areas of research, as well as areas that warrant further investigation. Next, we identify several research challenges posed by emerging systems for the future. In order to help delineate the scope of future RE research directions, we then identify several strategies for performing RE research. (The spectrum of research strategies ranges from empirical research to paradigm shifts.) Finally, within the context of these RE research strategies, we identify "hot areas" of research that address RE needs for emerging systems of the future.

1. Introduction

The success of a software system depends on how well it fits the needs of its users and its environment [111, 114] . *Software requirements* comprise these needs, and *requirements engineering (RE)* is the process by which requirements are elicited, modelled, analyzed, and documented. Requirements encompass more than desired functionality – users increasingly demand systems that are usable, reliable, secure, and economical, while product developers want to be able to adapt and enhance products rapidly, in response to changing both user's needs and environmental conditions. As such, improving the effectiveness of requirements-related activities requires multi-disciplinary research, involving aspects of computer science, mathematics, engineering, human-computer interaction, and social and cognitive sciences.

In this paper, we offer our views of the research directions in requirements engineering. In contrast to Nuseibeh and Easterbrook's roadmap paper from the ICSE'00 track on the Future of Software Engineering [57] (herein referred to as the "2000 Roadmap Paper"), which emphasized current research in requirements engineering, this paper focuses on research directions and identifies challenging RE problems of emerging systems of the future. We start in Section 2 with a summary of the state of the art of RE knowledge and research. Section 3 identifies research challenges for several future and emerging types of systems. In Section 4, we enumerate general paradigms for conducting research, ranging from revolutionary research to empirical evaluation to codifying proven solutions. Section 5 highlights research hotspots that address the RE challenges of emerging systems. We conclude with a summary of near- and long-term RE research needs.

2. State of the Art of RE Research

In general, the research challenges faced by the requirements engineering community tend to parallel those faced by the general software engineering community. Several additional challenges are faced by the RE community. First, RE artifacts and processes have to be understandable and usable by a broader range of stakeholders than other SE tasks. For example, RE notations and processes must satisfy a delicate balance between formal, analyzable artifacts and high-level (often informal), intuitive to understand artifacts. In contrast, notations and artifacts for design, implementation, and testing are typically more formally, technically defined and involve automated processing. Second, the information available for RE tends to be less well-defined and more likely to change than that used for other development tasks, which means that the information available at the beginning of an RE process may be dramatically different than what is derived many iterations later. As such, RE techniques necessarily may be more complex (involving many stages), or several RE techniques have to be combined to handle such a range of information. Finally, RE

Table 1. Matrix Summarizing Research in Requirements Engineering

Research Contributions	Elicitation	Modeling	Analysis, Validation, Verification	Management
Notations	Goals [18, 150] Policies [17] Scenarios [2, 31, 41] Enterprise models [45]	Data models [76] Behavioral models [78, 145] Domain descriptions [10] Logics [47] Anti-models [135, 144, 151] Variability modeling [36, 123, 130] Formalizations [101, 109, 141] Nonfunctional requirements [26, 61]	Logics [47] Model checking [19, 48]	–
Techniques	Identifying stakeholders [132] Metaphors [117, 119] Persona [9] Contextual requirements [32, 140] Animation [71, 97, 148] Prototyping [43] Inventing requirements [99]	RE reference model [64, 65, 115] Model elaboration [147] Viewpoints [112, 134] Model merging [128, 143] Model synthesis [5, 37, 92, 146, 159] Modeling patterns [51, 85, 149] Model composition [68] Modeling facilitators [30, 83, 113] Formalization heuristics [17, 60] Methodologies [14]	Linguistic analysis [56, 129, 153] Ontologies [81] Checklists [154] Consistency checking [55, 70, 107] Interaction analysis [23, 67, 125] Obstacle analysis [96, 152] Causal order analysis [11] Simulation [142] Invariant generation [79] Model checking [25, 52, 137] Model satisfiability [76]	Prioritization [105] Variability Analysis [63, 93] Requirements selection [122, 139] Aligning Req. with COTS [91, 126] Effort estimation [33] Impact analysis [88] Traceability [29, 69, 127, 131] Stability analysis [20] Scenario management [4] Feature management [155] Global RE [40]
Evaluation	Elicitation [42, 39, 94] Negotiation [75]	Air traffic control [98, 158] Aspects [12] Viewpoints [53]	Security [151] Semantic web ontologies [49] User interfaces [15] Inspection methods [118]	RE payoff [38, 54] Requirements evolution [7, 95] Risk management [59] Cost estimation in web app. [104] Req. reuse in automotive software[72]

information is inherently going to be tied more tightly with the application domain, thus potentially requiring domain-specific expertise to be integrated with system development information.

Before exploring future directions, we highlight previous work as a way of summarizing the state of the art of RE knowledge and technology. This section can be viewed as an update to the 2000 Roadmap Paper [111]. For consistency, we adopt much of the terminology used in that paper.

To provide a visual map of RE research, we organize research results in a matrix structure that relates each result to the requirements problem that it addresses and to the contribution it makes towards a solution – a la Zave's proposed scheme for classifying RE research [161]. The research space is roughly decomposed into four categories of requirements problems (elicitation, modeling, validation and analysis, and requirements management) and three types of research contributions (notations, techniques, and empirical evidence). This decomposition is comparable to the top-level decomposition in Zave's classification scheme. The resulting matrix is shown in Table 2.

Elicitation. Requirements elicitation comprises activities that enable the understanding of the goals, objectives, and motives for building a proposed software system. Elicitation also identifies the requirements that the resulting system must satisfy to be considered acceptable. The requirements to be elicited range from modifications to well understood problems and systems (e.g., software upgrades), to hazy understandings of new problems being automated, to relatively unconstrained requirements that are open to invention (e.g., mass-market software). As such, most of the research in elicitation focuses on techniques for improving the precision, accuracy, and variety of the requirements details:

- Techniques for *identifying stakeholders* [132] help to ensure that everyone who may be affected by the software is consulted during elicitation.

- *Analogical techniques*, like norms, metaphors [119], and personas [9, 34], help stakeholders to consider more deeply and be more precise about their requirements.
- Observation, apprenticeship [16], ethnography [136], and other *contextual techniques* [32, 140] analyze stakeholders' requirements with respect to a particular context and environment, to help ensure that the eventual system is fit for use in that environment.
- *Feedback techniques* use models, model animations [71, 97, 148], prototypes [43], mockups, and storyboards to elicit positive and negative feedback on early representations of the proposed system.
- Techniques for *inventing requirements*, like brainstorming and creativity workshops [99], help to identify nonessential requirements that make the final product more appealing.

Modeling. Requirements are presented in one or more modeling notations. Requirements models are used in a variety of ways, ranging from eliciting feedback from stakeholders, to teasing out missing details, to documenting final decisions. Early-phase requirements models are used to explore and learn about the stakeholders' problem. Such exploratory models, like use cases, scenarios, enterprise models, and some policy [17] and goal notations [18], tend to be informal, incomplete, and inexpensive to create and maintain, so that specifiers can keep them up-to-date as the requirements evolve. Late-phase requirements models tend to be more complete, unambiguous, and formal, as these models are used to communicate the requirements to downstream developers. Each modeling notation is designed to elicit or record specific details about the requirements, such as information to be maintained, functions on the data, responses to inputs, or constraints on data or behavior. Of these, scenario-based notations [2, 5, 37, 41, 143, 146, 147, 159] have been the focus of much recent research – perhaps because scenarios are easiest for practitioners and nontechnical stakeholders to use, or perhaps because scenarios are naturally incomplete, and so they lend themselves to lots of research problems.

In addition, there is considerable research on techniques for creating, combining, and manipulating models.

- *Modeling strategies* provide guidelines for structuring models. For example, *RE reference models* [64, 65, 115] decompose requirements-related descriptions into the stakeholders' *requirements*, the *specification* of the proposed system, and assumptions made about the system's *environment*; and they establish correctness criteria for verifying that the specified system will meet the requirements. Whereas in the *viewpoints* approach [112, 134], each stakeholder's requirements are retained in separate models, and the synthesis of a global consistent model of all stakeholders' concerns is delayed until conflicts can be resolved knowledgeably.
- The RE community has started to collect *model patterns* [51, 85, 149] that encode common solutions to complex modeling problems, and to develop tools [30, 83, 113] to help specifiers to apply these patterns.
- Several contributions propose *model transformations* for combining or manipulating models to derive new models. For example, model synthesis [5, 37, 92, 146, 159] and model composition [68] techniques integrate complementary submodels into a composite model, whereas model merging [128, 143] techniques unify different views of the same problem. There is also preliminary work on heuristics for formalizing natural-language policies [17] and goal models [60].

Several of the above-mentioned projects directly address challenges raised in the 2000 Roadmap Paper [111]. For example, heuristics for formalizing informal models and tools that map constrained natural-language expressions to formal representations [30, 83, 113] help to bridge the gap between informal and formal requirements. The gap is also narrowed by research on formalizing the semantics of informal or semi-formal modeling notations [101, 141]. In addition, there has been significant advances in the modeling and analysis of nonfunctional requirements [26] and in establishing objective fit criteria for how well an eventual system must achieve various nonfunctional properties [61]. On the other hand, there has been little progress on new notations for modeling environment descriptions and assumptions [10]; instead, existing notations like functions [115], object models (e.g., UML), operational specifications (e.g., Z), and constraint languages continue to be used.

Analysis, validation, and verification. *Requirements analysis* assesses the quality of requirements models and documentation. Most of the research in this area focuses on new or improved automated techniques for detecting errors in models, where an "error" can be ambiguity [56, 81, 129, 153], inconsistency [55, 70, 107], an unknown interaction among requirements [23, 67, 125], a possible obstacle to requirements satisfaction [96, 152], or missing assumptions [11].

Requirements validation ensures that models and documentation accurately represent the stakeholders' requirements. Unlike the above analyses, which check software specifications against established well-formedness criteria, validation is an attempt to compare a specification against undocumented requirements. As such, it usually requires the direct involvement of stakeholders in reviewing the requirements artifacts. Research in this area focuses on improving the information provided to the stakeholder

for feedback, including animations [71, 97, 148], simulations [142], and derived invariants [79].

In cases where a formal description of the stakeholders' requirements exists, obtained perhaps by validation, *verification* techniques can be used to prove that the software specification meets these requirements. Such proofs often take the form of checking that a model satisfies some constraint. For example, model checking [25, 52, 137] checks behavioral models against temporal-logic properties about execution traces; and model satisfiability [76] checks that there exist valid instantiations of constrained object models, and that operations on object models preserve invariants. The notations listed in this column are notations that simplify and abstract the structure of the model to be verified [19, 48], to facilitate automated verification.

Requirements management. Requirements management is an umbrella activity that comprises a number of activities related to the management of the project or of the requirements phase. Such activities include traceability, impact analysis, cost estimation, risk management, as well as management of requirements variations. Research in this area focuses on easing management tasks and improving analysis techniques. For example, researchers have proposed a number of prioritization [105], visualization [63, 93], and analysis [122, 139] techniques to help managers select an optimal combination of requirements to implement – or to help project managers identify acceptable off-the-shelf solutions [91, 126]. Other researchers are investigating how to improve our ability to estimate the cost [33] of implementing the selected requirements, or the impact of new or modified requirements [88]. Of particular interest are tools and techniques to ease, and partially automate, the task of identifying and documenting traceability links among requirements artifacts and between requirements and downstream artifacts [29, 69, 127, 131]. Stability analysis [20] helps project managers to determine the maturity and stability of the elicited requirements, and to isolate those requirements that are most likely to change. Lastly, the basic management of requirements has become challenging, thus inspiring research on techniques to organize large numbers of requirements [4] that are globally distributed [40], and that are at different phases in development in different product variants [155].

Evaluation. There is surprisingly little empirical research on the effectiveness of requirements technologies. That said, several recent case studies evaluate how well research ideas work when applied to industrial-sized problems [75, 94, 98, 104, 158] or in industrial settings [38, 39, 59]. Additionally, several recent research projects have been evaluating how well requirements notations and techniques apply to, or can be adapted to, domain-specific problems, such as security [151], semantic webs [49], and user interfaces [15]. There have been a few comparative studies that compare the effectiveness of competing elicitation techniques [42, 53] and inspection techniques [118]. Finally, there have also been some post-mortem analyses on how requirements evolved in real-world systems [7, 95].

3 Research Challenges

As described above, the RE community has made significant progress in addressing RE research challenges with respect to today's systems. In this section, we identify several key challenges for future systems.

3.1. Scale

Future systems will not be limited to significant size increases (such as lines of code), but scale factors will also include complexity, the level of heterogeneity, number of sensors, number of decentralized decision-making nodes, etc. Another major challenge with these large-scale systems will be the management of these complex and numerous requirements, where continuous evolution of the requirements will add another level of complexity to the management problem. An example of an emerging system exhibiting many of these new scale factors are the emerging Ultra-Large Scale (ULS) systems [110]. ULS systems will be used to develop military command and control systems; ULS systems are also likely to be a big part of automotive systems of the future, intelligent transportation management systems, critical infrastructure protection systems, etc.

3.2. Cyber-Physical Systems

Increasingly, computing systems interact with the physical world. *Cyber-physical systems* are a new generation of engineered systems that need to be highly dependable, efficiently produced, and capable of advanced performance in information, computation, communication, and control [35]. "A cyber-physical system integrates computing, communication and storage capabilities with the monitoring and/or control of entities in the physical world, and must do so dependably, safely, securely, efficiently and in real-time. [35]" Example cyber-physical systems include intelligent vehicle systems, automated manufacturing, critical infrastructure monitoring, disaster response, optimization of energy consumption, and efficient agriculture [35]. High-level requirements for CPS include dependability, security, safety, efficient, and real-time operation, while being scalable, cost-effective, and adaptive [35]. n processing demands are being imposed by

The biggest factor accelerating the growth of the emerging area of dynamically adaptive systems is the increase in

volume and sophistication of sensors and sensor networks. With this new sensor technology, scientists and practitioners are able to use sensors to monitor critical behavior for applications such as power-grid infrastructures, bridges, and transportation systems. These sensors will be equipped with varying levels of computational capacity and may be placed in locations (e.g., power transformers, nuclear reactor cores, hazardous/toxic sites), where it may be difficult, if not impossible, for humans to access to update with new software and behavior. The CPS systems will need to be able to respond to unexpected conditions, thus requiring the requirements to continuously evolve, even during execution. As such, traditional approaches to requirements modeling and analysis will not be sufficient, since the requirements will be so dynamic in nature. New RE techniques will need to be more *agile* and rigorous given the critical nature of CPS applications.

3.3. Self-Adaptive Systems

With the potential for millions of sensors at the wireless edge of the Internet, combined with the increasing complexity of computing technology, gives rise to the need for computer systems capable of self-management. *Autonomic Computing* [82] refers to systems that adapt, or self-regulate, in response to changing conditions, such as hardware or software failures, fend off attacks, optimize performance, all with minimal human intervention. IBM's original motivation for autonomic computing was to build self-managing capabilities into large farms of servers, thus reducing the demand for specialized systems administrators who have to manage increasingly complex computing infrastructure. Since the original introduction of the term, autonomic computing-related techniques have been explored for other types of applications, such as critical infrastructure protection (e.g., power grid management, bridge monitoring), CPS (e.g., automated manufacturing, autonomic networking [138]), public safety systems, and command and control systems.

Depending on the type dynamic adaptation need at run time to changing environment conditions, user needs, and system requirements, these systems have been given slightly different names. For example, *self-healing* refers to those systems that have to adapt dynamically in response to a system failure, fault, errors, security breaches, where the failures are detected at run time and the failure conditions and types may not be known at system development time. In contrast, *self-managing* or *autonomic* tend to refer to systems that have minimal human guidance during execution, and are intended to be autonomous with the ability to adapt (at run time) to new environmental conditions and new requirements that were not anticipated during development time.

Three key areas of research challenges are posed by dynamically adaptive systems: environment and system monitoring, decision-making for adaptation, and the adaptive mechanisms. For all three areas, the biggest RE challenge is how to handle incomplete and continuously evolving requirements of the system that must respond to changing environmental conditions, user needs, and resource constraints. Because many of the applications for the dynamic adaptation are high assurance, these systems must be formally analyzable for satisfying critical properties before, during, and after adaptation. Much of the preliminary SE research efforts for adaptive systems have focused on architectural frameworks [58](Self-Adaptive Systems article), programming languages, and adaptive mechanisms for dynamically adaptive systems [100]. RE activities largely focus on specifying and/or verifying assurance models and properties for adaptive systems [3, 21, 86, 87, 90, 163, 164]. Most of these techniques do not explicitly address the challenges posed by incomplete information.

3.4. Security

As computing systems become ever more pervasive and mobile, and as they increasingly automate and manage consumer-critical processes and data, they increasingly become the target of security attacks. There has been substantial work on how to improve software security, in the form of solutions and strategies to avoid vulnerabilities, to protect systems and information, and to defend against or recover from attacks. However, most of these solutions are threat specific. Thus, the RE challenge with respect to secure systems is to identify potential security threats, so that designers can select and employ appropriate protections. This task involves significant study, modeling, and analysis of the environment in which the system will operate, and so far there has been little work on domain modeling – despite the fact that its importance was raised almost 15 years ago [77].

Moreover, there is no consensus on how security requirements themselves should be documented. Is security a nonfunctional requirement to be resolved and optimized at design time along with other competing nonfunctional requirements? Or should security requirements be realized as functional requirements, in the manner that user interfaces and timing deadlines are woven into behavioral specifications? These are open questions for the modeling, analyses, and security communities to resolve.

3.5. Globalization

The trend towards globalization is motivated by the desire to capitalize on global resource pools, decrease costs, exploit a 24-hour work day, and to be geographically closer to the end-consumer [62]. All of these factors pose sev-

eral new RE challenges. For example, elicitation and early modeling are collaborative activities that require the construction of a shared mental model of the problem and requirements. There is an explicit disconnect between this need for collaboration and distance imposed by global development.

4. Research Strategies

In order to provide context and a better understanding of the hot spots of RE research for the future (see Section 5), we have identified several major research strategiess that offer complementary approaches for conducting research. Each of these research strategies addresses different types of research questions and have different expectations for the research results. These strategiess are based on a number of different sources, including Shaw's overview of criteria for good research in software engineering [133], Redwine and Riddle's review of software technology maturation [121], Basili's review of research paradigms [13], and the combined experience of both authors. Table 2 introduces and briefly defines the eight research strategies that we have identified thus far, listed in the order of increasing maturity of the RE research results. Next, we describe each of these paradigms in more detail, including examples.

Paradigm Shift. A *paradigm shift* dramatically changes the way of thinking, on the order of a transformation, revolutionary, or metamorphsis [89]. The change typically requires some disruption (i.e., technologically may have to apply new tecnhiques to smaller, simpler problems and then graduate to applicability to problems, size, and scope of interest) before the benefits of the paradigm shift can be perceived. Typically, there are two strategies for bringing about a paradigm shift: push and pull. A paradigm shift can be *pushed* into the community when a technique is serendipitously discovered, where it may make major advances in solving a problem for which it may not have been originally intended. The world wide web is a classic example of such a paradigm shift, where the web has significantly changed the way society behaves for communication and delivery of consumer services. A paradigm shift can be *pulled* when there is either a real or a perceived crisis in which current techniques are inadequate, no matter how the techniques are extended or improved [89]. For example, at the end of the 1960's, the area of software engineering emerged in an attempt to solve the "software crisis" which referred to the significant difficulty in writing correct, easily understood, and machine-verifiable computer software [46]. Key factors contributing to the crisis include the complexity of software and the constant change of requirements, user expectations, operating conditions, etc. The identification of the crisis led to the development of structured programming, software development methodologies and processes.

Leverage other disciplines. In order to leverage another discipline, it is critical for a researcher to see the analogoous relationships between the two disciplines. The US National Science Foundation has solicited proposals to significantly advance (versus making incremental progress) the "Science of Design" discipline [106], where the most recent solicitation explicitly encourages research proposals that imports or adapts from other design fields, such as engineering, biology, architecture, economics, and the arts. Examples include Cleanroom Software Engineering [103], where the term and general idea for the term "cleanroom' was from the electronics domain. A physically clean room is needed for the fabrication of hardware in order to prevent the introduction of defects; the analogy in software was to have a well-defined, incremental process for planning, specifying, designing, verifying, coding, and testing software to achieve certifiable zero-defect software.

Leverage technology. Advances in computing and related fields can be used to make progress in requirements engineering. For example, the field of model checking was originally developed for use in hardware and protocol verification [28, 58]. The success of using model checking for hardware has catalyzed an entire area of research focusing on model checking software systems. In contrast to model checking hardware, where models are typically well-defined and stable, software models are generally more difficult to specify due to vague and changing requirements.

Evolutionary. The antithesis of a paradigm shift would be evolutionary research that improves on existing technologies [73]. Although the research challenges listed in the previous section identify a number of new problems that the RE community will be called upon to address, most new software developed in the near future will resemble the types of systems being developed today. As such, the software community will continue to benefit from improvements to current requirements technologies, which were created to solve the problems that today's practitioners face.

In many ways, evolutionary research is about moving research technologies down the research-paradigm ladder. Existing notations and techniques can be expanded, adapted, or generalized to address problems raised in different software domains. Methodological support, modelling patterns, and analysis strategies can ease the transfer of current technologies from early adopters to ordinary practitioners. Empirical research can identify the problems and contexts for which a technology is most effective, and can indicate aspects that could be further improved.

Research Strategy	Definition
Paradigm Shift:	Dramatically change the way of thinking, on the order of a transformation, revolutionary, or metamorphsis [89]. The change typically requires some disruption before the benefits of the paradigm shift can be perceived.
Leverage other disciplines:	Make advances in one discipline by leveraging and recasting techniques from another discipline.
Leverage technological advances:	Current technological advances are leveraged to enable progress for a particular discipline.
Evolutionary:	This research strategy takes an existing techniques and improves one or more dimensions of the approach to obtain an improved version; this strategy is perhaps one of the most commonly applied.
Domain-specific:	This approach develops a technique that is narrowly applicable to a specific domain.
Empirical:	This strategy proposes a model/approach, develops statistical or qualitative metrics to assess the approach, then applies the approach to case studies, performs assessments according to the metrics, validates the approach, and then repeats the process [13]. The research is typically in the form of single or longitudinal case studies, applying the technique to concrete data.
Engineering:	This research strategy starts with an existing solution, proposes improvements, such as making the approach more systematic, practically usable, amenable to automation, and more analyzable [13].
Generalization:	This approaches take a narrowly defined technique and generalizes it to apply to a broader class of problems or data.

Table 2. Enumeration of research paradigms

Domain-specific. Typically, the more specialized the application domain, the more likely there will be software engineering techniques developed to address the special needs. For example, starting with model-driven development (MDD) techniques [102], Motorola has specialized MDD for the telecommunications domain [156] and developed a proprietary MDA and supporting tools to generate the production-level code for many of their handheld devices. *Question: How can general RE techniques be specialized or customized for a specific application domain?*

Empirical. Empirical research typically involves two complementary approaches. The first is to compare similar techniques against standard criteria (e.g., completeness, correctness, consistency, non-ambiguity, feasibility, traceability) [44, 24, 42, 53, 118]. The second type of empirical research focuses on demonstrating how techniques can be used for real systems, thereby measuring effectiveness of a given RE technique [13, 75, 94, 98, 104, 158, 38, 39, 59].

Engineering. A surprising number of research problems arise in the course of trying to apply requirements technologies in the practice of RE. Engineering as a research paradigm looks at how to simplify and codify RE knowledge and techniques so that they are readily adopted by practitioners and taught to undergraduates. For example, visual formalisms [47, 50, 66, 78] ease the task of creating and reviewing precise specifications. Specification patterns [51, 85, 149, 84] not only help specifiers to create models, via instantiation and adaptation, but they also offer some level of uniformity and repeatability of such models. Methodologies and processes provide guidance on how to solve requirements problems from start to finish [120, 124]. Heuristics and strategies offer advice on how to use particular elicitation [6], modeling [17, 60], verification [80], or management technologies. One of the best known engineering-style research projects was David Parnas et al.'s case study that applied state-of-the-art software engineering practices to (re)develop the engineering artifacts and code for the U.S. A-7 naval aircraft; this work led to research results in tabular specifications [78], hierarchical module structures, abstract interfaces, and new inspection strategies [116].

Generalization. Some domain-specific or organization-specific techniques have been generalized to be more broadly applicable. For example, feature interaction analysis started with the telephony area [160, 68], but is now branching into other areas, such as web services [157] and middleware [162].

5. RE Research Hotpots

This section overviews RE research hotspots in the context of the research challenges of emerging systems overviewed in Section 3. The hotspots are roughly organized according to the research strategies presented in Section 4.

5.1. Paradigm Shifts for ULS Systems

Many of the RE challenges facing current systems are significantly magnified these large-scale systems. Requirements will come from many different stakeholders, perhaps presented at varying levels of abstraction, and involve multiple disciplines (e.g., sensors, scientific computation, artificial intelligence, etc.). New techniques are needed to compose these potentially vastly different types of requirements into a single coherent story. Detecting and resolving feature interactions and conflicts will pose a grand challenge to the RE community.

Preliminary activities have begun to address some of these challenges, while further recognizing the importance that RE will play in the successful development and evolution of these emerging systems. For example, a *60th* anniversary issue of *IEEE Computer* [74] overviewed the achievements and future prospects for software engineering. Two of the articles ([74], Broy and Jackson) identified research challenges posed by "software-intensive systems (SIS)" from the design of embedded systems and verification perspectives, respectively. Developing requirements engineering techniques to manage the scale effectively is critical in designing future SIS, for both modeling and verification purposes. Some progress in these areas include Broy and his group's work on several aspects of managing requirements for complex, software-intensive embedded systems, including the development of the AutoRaid requirements engineering tool [74] (p. 72), based on well-defined modeling theories to help manage complex requirements for software-intensive embedded systems.

In order to address requirements engineering needs for emerging systems, such as ULS, dynamically adaptive, and cyberphysical, while providing security and assurance requires the RE community to address three major challenges. First, new notations and techniques are needed to support the development of new abstractions. In addition, innovative decomposition strategies, simpler composition operators, and more automation of RE tasks are all needed to address the requirements of these types of systems. A constraint on these techniques and notations will be the need to support uncertainty and variability in the information available during the requirements engineering process, which will not be limited to the early lifetime of the system. Instead, the systems of the future will be continuously evolving, thus requirements engineering will be needed before and after deployment, as well as potentially even during execution. These new systems will require a different perspective on what level of information should be modeled, in contrast to the traditional approach to modeling requirements, which typically focus on low-level functional information. New logics may need to be explored that go beyond the traditional two valued logics to represent the uncertainty and continuous variability. In addition, new underlying computational models may need to be developed to reason about and validate the integration of discrete and continuous systems behavior.

Second, given the fact that many of the future emerging systems will be used in critical applications, such as intelligent transportation systems, financial systems, medical care systems, military command and control, etc., security and high assurance will be essential. The big challenge will be the need to alter the view and expectations for the notions of correctness and acceptability. That is, new techniques are needed that can model and even support analysis of systems information with incomplete information, varying levels of abstractions, incomplete or partial information. These criteria might be based on new types logic. These notations and analysis tools should be accessible to the typical RE analyst.

Finally, techniques need to be developed that enable RE tasks to be more prescriptive and systematic with information that is less well-defined and continuously evolving. The analogy is whether the same level of maturity for RE can be achieved that the design field has achieved with architectural styles for designs, design patterns, and design transformations. One step towards this objective is the notion of requirements reuse, which was raised in the 2000 Roadmap Paper, but has received little attention. One way to tackle complexity and size is to attempt to reuse existing requirements that may have tightly or loosely defined refinements for design and code. It may be possible to identify some number of patterns or units of reusable requirements for particular domains or particular types of applications. Each requirement pattern must come with standard pattern fields, such as context, problem addressed, consequences, properties, etc. The automotive industry has expressed interest in using "generic, reusable requirements" for the complex automotive systems. Work with specification patterns [51, 84] and object analysis patterns [85] are recent attempts at reusable requirements-level information.

5.2. Biologically-Inspired Computing

A common challenge to the ULS, adaptive, and cyber-physical systems is the need to respond to unanticipated changes in the environment, new user needs, unexpected failures, security threats, etc. With the current technology, it is impossible to predict or specify the requirements for target systems needed to respond to these changing, even unanticipated operating conditions. As such, one key area to investigate is how techniques from other disciplines, such as biology, can be used to discover the requirements for the adaptive systems. as ULS, cyber-physical, and dynamically adaptive systems. In essence, techniques are needed to preview the possible target systems that should replace a running system in response to unexpected conditions to keep

the system in an acceptable, consistent state.

Given the ability that natural organisms have to respond to adverse and unexpected conditions, research has been performed to explore how biologically-inspired approaches can be used to address the challenges posed by dynamically adaptive systems. *Biomimetics* comprises those techniques that attempt to imitate or simulate the behavior of natural organisms. For example, Sutcliffe and Maiden's work [1] with establishing a domain theory for RE draws heavily from cognitive science and the human use of analogical reasoning. The NASA ANTS (Autonomous Nano Technology Swarm) project involves large collections (i.e., swarms) of mission-independent small, autonomous, self-similar, reconfigurable robot-like entities can collectively perform mission-specific tasks [27], such space exploration, including planetary surfaces. The general objective is to understand and then mimic the behavior of social insect colonies that can perform relatively sophisticated tasks based on efficient social interaction and coordination. Evolutionary computation, such as the use of genetic algorithms, uses an evolutionary process to produce desired behavior. Sutcliffe *et al.* [108] used genetic algorithms to select an optimal set of components that satisfy a number of fitness criteria for reliability requirements.

In contrast to the biomemitic and evolutionary computing approaches where either behavior in nature is being imitated or the possible sets of behavior are known in advance, respectively, new work at Michigan State University is exploring how digital evolution techniques can be extended to provide a biologically-driven evolution process to discover new, unanticipated behavior and thus, new requirements for potential target systems of dynamically adaptive systems.

5.3. Roundtrip Requirements Engineering

For systems that require complex decision-making logic, such as dynamically adaptive systems, cyber-physical systems, ULS systems, it will be necessary to have intelligent decision-making subsystems based on large volumes of heterogeneous sensor data. Most of these systems will require some type of feedback loop to enable the system to dynamically evolve to changing conditions. Techniques from areas like machine learning, data mining, probablistic reasoning, and control theory can all be leveraged to help discover, evolve, and analyze the requirements for these challenges. For example, control theory has been applied to improve testing [8] by computing the level of effort needed to reduce the number of effors and decrease schedule overruns under changing environment conditions. These results are then fed back through the control loop at specific variable points to identify appropriate changes to improve the overall results. Control theory has also been used to predict and avert constraint violations of an adaptive system [22]; memory overflow is an example constraint that undergoes prediction analysis and alternative software components are selected to avert the constraint violation. How can these techniques be extended to establish a feedback loop from early to late requirements.

5.4. Empirical RE Research: Partnerships between Research and Practice

5.5. Engineering Cyber-Physical Systems

The cyber-physical systems require not only innovative techniques to specify the requirements of heterogeneous systems working on discrete and continuous scales, but these techniques must be integrated into a systematic process with sufficiently well-defined steps. Furthermore most of the RE techniques must be amenable to significant automation in order to be scalable to the size and complexity.

One particlar area that is actively being pursued for CPS is how to more effectively leverage commercial off the shelf (COTS) components [35], as a means to move towards an engineering process for developing CPSs. But the software development field is not yet sufficiently mature to support such a direction. Specifically, formally verifiable components that can serve as the building blocks of CPS are needed. These components will likely need to be defined in terms of different granularities depending on level of use (e.g., application, network, middleware). These components should be more easily certifiable as COTS for reuse. In order to make them more accessbile, these components should be indexable by behavior specifications, which becomes an RE challenge for developing easy to use specification languages that are amenable to automated processing for syntactic and semantic-based retrieval techniques. What requirements factors will determine whether it is better to use COTS or to design/implement from scratch? New methodologies are needed to integrate the specification of system components with discrete and continuous behavior.

The physical world brings not only information delivered on a continuous scale (in contrast to the discrete data processed by traditional computing software), but we are also faced with new extremes in terms of size, ranging from nano to monolithic (such as gas turbine engines for jet planes), leading to multi-scale and and multi-level complexity for the systems. (Many of these challenges will overlap with the general scale challenges.)

5.6. RE Challenges Posed by Globalization

Interestingly, this area of research has been largely promoted by industrial needs – global software development is a reality in full swing. In order to maximize the benefits and ensure positive long term effects, it is imperative for global-centric SE techniques to be developed.

Furthermore because many of the downstream development efforts, such as coding and testing are done globally, requirements engineering, including knowledge management [62] (p. 18), is one of the key areas of need. Bhat *et al* have proposed a framework focusing on a people-process-technology paradigm that describe best practices for goals, culture, processes, and responsibility, all of which are shared across the entire global organization [62]. Sinha *et al* have developed an Eclipse-based tool for distributed requirements engineering collaboration [62].

Globalization poses two broad classes of challenges to the RE research community. First, what new RE techniques are needed to enable effective distributed RE. We need RE research to produce techniques to facilitate and manage distributed requirements elicitation, distributed modeling, distributed requirements negotiation/agreement, and management of distributed teams (not just geographically distributed, but distributed in terms of time zone, culture, and language).

Second, new or extended RE techniques are needed to support outsourcing for downstream development activities. Sspecifically, techniques are needed that enable requirements engineers to easily construct complete and unambiguous specifications. These specifications need to be understandable by general practitioners, versus specification language experts.

6. Conclusions

References

[1] The domain theory for requirements engineering. *IEEE Transactions on Software Engineering*, 24(3):174–196.

[2] A. Alfonso, V. Braberman, N. Kicillof, and A. Olivero. Visual timed event scenarios. In *Proc. of the Int. Conf. on Soft. Eng. (ICSE)*, pages 168–177, 2004.

[3] R. Allen, R. Douence, and D. Garlan. Specifying and analyzing dynamic software architectures. In *Proceedings of the 1998 Conference on Fundamental Approaches to Software Engineering (FASE'98)*, Lisbon, Portugal, March 1998.

[4] T. A. Alspaugh and A. I. Antón. Scenario networks for software specification and scenario management. Technical Report TR-2001-12, North Carolina State University at Raleigh, 2001.

[5] R. Alur, K. Etessami, and M. Yannakakis. Inference of message sequence charts. In *Proc. of the Int. Conf. on Soft. Eng. (ICSE)*, pages 304–313, 2000.

[6] A. I. Antón. Goal-based requirements analysis. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 136–144, 1996.

[7] A. I. Antón and C. Potts. Functional paleontology: The evolution of user-visible system services. *IEEE Trans. on Soft. Eng.*, 29(2):151–166, 2003.

[8] J. ao W. Cangussu, R. A. DeCarlo, and A. P. Mathur. A formal model of the software test process. *IEEE Trans. Softw. Eng.*, 28(8):782–796, 2002.

[9] M. Aoyama. Persona-and-scenario based requirements engineering for software embedded in digital consumer products. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 85–94, 2005.

[10] Y. Arimoto, M. Nakamura, and K. Futatsugi. Toward a domain description with CafeOBJ. In *Proc. 23rd JSSST Convention*, 2006.

[11] P. Baker, P. Bristow, C. Jervis, D. King, R. Thomson, B. Mitchell, and S. Burton. Detecting and resolving semantic pathologies in uml sequence diagrams. In *Proc. of SIGSOFT Found. on Soft. Eng. (FSE)*, pages 50–59, 2005.

[12] E. Baniassad and S. Clarke. Theme: An approach for aspect-oriented analysis and design. In *Proc. of the Int. Conf. on Soft. Eng. (ICSE)*, pages 158–167, 2004.

[13] V. R. Basili. The experimental paradigm in software engineering. In *Proceedings of the International Workshop on Experimental Software Engineering Issues: Critical Assessment and Future Directions*, pages 3–12, London, UK, 1993. Springer-Verlag.

[14] B. Berenbach. The evaluation of large, complex uml analysis and design models. In *Proc. of the Int. Conf. on Soft. Eng. (ICSE)*, pages 232–241, 2004.

[15] J. Berstel, G. Roussel, S. C. Reghizzi, and P. S. Pietro. A scalable formal method for design and automatic checking of user interfaces. In *Proc. of the Int. Conf. on Soft. Eng. (ICSE)*, pages 453–462, 2001.

[16] H. Beyer and K. Holtzblatt. Apprenticing with the customer. *Comm. of the ACM*, 38(5):45–52, 1995.

[17] T. D. Breaux and A. I. Antón. Analyzing goal semantics for rights, permissions, and obligations. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 177–188, 2005.

[18] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. TROPOS: an agent-oriented software development methodology. *J. of Auto. Agents and Multi-Agent Sys.*, 8(3):203–236, 2004.

[19] T. Bultan. Action language: A specification language for model checking reactive systems. In *Proc. of the Int. Conf. on Soft. Eng. (ICSE)*, pages 335–344, 2000.

[20] D. Bush and A. Finkelstein. Requirements stability assessment using scenarios. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 23–32, 2003.

[21] C. Canal, E. Pimentel, and J. M. Troya. Specification and refinement of dynamic software architectures. In *Proceedings of the TC2 First Working IFIP Conference on Software Architecture (WICSA1)*, pages 107–126. Kluwer, B.V., 1999.

[22] J. W. Cangussu, K. Cooper, and C. Li. A control theory based framework for dynamic adaptable systems. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 1546–1553, New York, NY, USA, 2004. ACM Press.

[23] P. Carlshamre, K. Sandahl, M. Lindvall, B. Regnell, and J. N. Dag. An industrial survey of requirements interdependencies in software product release plannin. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 84–93, 2001.

[24] Part of the RE Conference.

[25] W. Chan, R. J. Anderson, P. Beame, S. Burns, F. Modugno, D. Notkin, and J. D. Reese. Model checking large software specifications. *IEEE Trans. on Soft. Eng.*, 24(7):498–520, 1998.

[26] L. Chung, B. Nixon, E. Yu, and J. Mylopoulos. *Non-functional Requirements in Software Engineering*. Kluwer, 1999.

[27] P. Clark, M. L. Rilee, S. A.Curtis, W. Truszkowski, G. Marr, C. Cheung, and M. Rudisill. BEES for ANTS: Space mission applications for the autonomous nanotechnology swarm. In *AIAA 1st Intelligent Systems Technical Conference*. American Institute of Aeronautics and Astronautics, 2004.

[28] E. Clarke, O. Grumberg, and D. Long. Verification tools for finite-state concurrent systems. In *A Decade of Concurrency – Reflections and Perspectives*, volume 803. 1994. Lecture Notes in Computer Science.

[29] J. Cleland-Huang, G. Zemont, and W. Lukasik. A heterogeneous solution for improving the return on investment of requirements traceability. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 230–239, 2004.

[30] R. L. Cobleigh, G. S. Avrunin, and L. A. Clarke. User guidance for creating precise and accessible property specifications. In *Proc. of SIGSOFT Found. on Soft. Eng. (FSE)*, pages 208–218, 2006.

[31] A. Cockburn. *Writing Effective Use Cases*. Addison-Wesley, 2001.

[32] T. Cohene and S. Easterbrook. Contextual risk analysis for interview design. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 95–104, 2005.

[33] R. Conradi, B. Anda, and P. Mohagheghi. Effort estimation of use cases for incremental large-scale software development. In *Proc. of the Int. Conf. on Soft. Eng. (ICSE)*, pages 303–311, 2005.

[34] A. Cooper. *The Inmates are Running the Asylum*. Sams, 1999.

[35] Cyber-Physical Systems, October 2006.

[36] K. Czarnecki and U. W. Eisenecker. *Generative Programming*. Addison-Wesley, 2000.

[37] C. Damas, B. Lambeau, and A. van Lamsweerde. Scenarios, goals, and state machines: a win-win partnership for model synthesis. In *Proc. of SIGSOFT Found. on Soft. Eng. (FSE)*, pages 197–207, 2006.

[38] D. Damian and J. Chisan. An empirical study of the complex relationships between requirements engineering processes and other processes that lead to payoffs in productivity, quality, and risk management. *IEEE Trans. on Soft. Eng.*, 32(7):433–453, 2006.

[39] D. Damian, A. Eberlein, M. Shaw, and B. Gaines. An exploratory study of facilitation in distributed requirements engineering. *Req. Eng. J.*, 8(1):23–41, 2003.

[40] D. Damian and D. M. (eds.). Global software development. *IEEE Soft. special issue*, 23(5), 2006.

[41] W. Damm and D. Harel. Lscs: Breathing life into message sequence charts. *Form. Meth. in Sys. Des.*, 19(1):45–80, 2001.

[42] A. Davis, O. Dieste, A. Hickey, N. Juristo, and A. M. Moreno. Effectiveness of requirements elicitation techniques: Empirical results derived from a systematic review. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 176–185, 2006.

[43] A. M. Davis. Operational prototyping: A new development approach. *IEEE Soft.*, 9(5):70–78, 1992.

[44] A. M. Davis. *Software Requirements: Objects, Functions and State*. Prentice Hall PTR, 1993.

[45] N. Desai, A. U. Mallya, A. K. Chopra, and M. P. Singh. Interaction protocols as design abstractions for business processes. *IEEE Trans. on Soft. Eng.*, 31(12):1015–1027, 2005.

[46] E. W. Dijkstra. EWD 340: The humble programmer. *Communications of the ACM*, pages 859–866, 1972.

[47] L. K. Dillon, G. Kutty, L. E. Moser, P. M. Melliar-Smith, and Y. S. Ramakrishna. A graphical interval logic for specifying concurrent systems. *ACM Trans. on Soft. Eng. & Meth.*, 3(2):131–165, 1994.

[48] L. K. Dillon and R. E. K. Stirewalt. Inference graphs: A computational structure supporting generation of customizable and correct analysis components. *IEEE Trans. on Soft. Eng.*, 29(2):133–150, 2003.

[49] J. S. Dong, C. H. Lee, Y. F. Li, and H. Wang. Verifying daml+oil and beyond in z/eves. In *Proc. of the Int. Conf. on Soft. Eng. (ICSE)*, pages 201–210, 2004.

[50] N. Dulac, T. Viguier, N. G. Leveson, and M.-A. D. Storey. On the use of visualization in formal requirements specification. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 71–80, 2002.

[51] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *Proc. of the Int. Conf. on Soft. Eng. (ICSE)*, pages 411–420, 1999.

[52] S. Easterbrook and M. Chechik. A framework for multi-valued reasoning over inconsistent viewpoints. In *Proc. of the Int. Conf. on Soft. Eng. (ICSE)*, pages 411–420, 2001.

[53] S. Easterbrook, E. Yu, J. Aranda, Y. Fan, J. Horkoff, M. Leica, and R. A. Qadir. Do viewpoints lead to better conceptual models? an exploratory case study. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 199–208, 2005.

[54] K. E. Emam and A. Birk. Validating the iso/iec 15504 measure of software requirements analysis process capability. *IEEE Trans. on Soft. Eng.*, 26(6):541–566, 2000.

[55] G. Engels, J. M. Küster, R. Heckel, and L. Groenewegen. A methodology for specifying and analyzing consistency of object-oriented behavioral models. In *Proc. of SIGSOFT Found. on Soft. Eng. (FSE)*, pages 186–195, 2001.

[56] A. Fantechi, S. Gnesi, G. Lami, and A. Maccari. Application of linguistic techniques for use case analysis. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 157–164, 2002.

[57] A. Finkelstein, editor. *The Future of Software Engineering*. Proc. of the Int. Conf. on Soft. Eng. (ICSE), 2000.

[58] *Future of Software Engineering*. IEEE Computer Society, 2007.

[59] B. Freimut, S. Hartkopf, P. Kaiser, J. Kontio, and W. Kobitzsch. An industrial case study of implementing software risk management. In *Proc. of SIGSOFT Found. on Soft. Eng. (FSE)*, pages 277–287, 2001.

[60] A. Fuxman, L. Liu, M. Pistore, M. Roveri, and J. Mylopoulos. Specifying and analyzing early requirements: Some experimental results. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 105–114, 2003.

[61] T. Gilb. *Competitive Engineering: A Handbook for System Engineering, Requirements Engineering, and Software Engineering using Planguage*. Butterworth-Heinemann, 2005.

[62] *IEEE Software*, volume 23, 2006. Special Issue on Global Software Development.

[63] B. Gonzalez-Baixauli, J. C. S. do Prado Leite, and J. Mylopoulos. Visual variability analysis for goal models. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 198–207, 2004.

[64] C. A. Gunter, E. L. Gunter, M. Jackson, and P. Zave. A reference model for requirements and specifications. *IEEE Soft.*, 17(3):37–43, 2000.

[65] J. Hall and L. Rapanotti. A reference model for requirements engineering. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 181–187, 2003.

[66] D. Harel. Statecharts: A visual formalism for complex systems. *Sci. Comp. Prog.*, 8(3):231–274, 1987.

[67] J. H. Hausmann, R. Heckel, and G. Taentzer. Detection of conflicting functional requirements in a use case-driven approach. In *Proc. of the Int. Conf. on Soft. Eng. (ICSE)*, pages 105–115, 2002.

[68] J. D. Hay and J. M. Atlee. Composing features and resolving interactions. In *Proc. of SIGSOFT Found. on Soft. Eng. (FSE)*, pages 110–119, 2000.

[69] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram. Advancing candidate link generation for requirements tracing: The study of methods. *IEEE Trans. on Soft. Eng.*, 32(1):4–19, 2006.

[70] C. L. Heitmeyer, R. D. Jeffords, and B. G. Labaw. Automated consistency checking of requirements specifications. *ACM Trans. on Soft. Eng. & Meth.*, 5(3):231–261, 1996.

[71] C. L. Heitmeyer, J. Kirby, B. G. Labaw, and R. Bharadwaj. SCR*: A toolset for specifying and analyzing software requirements. In *Comp. Aid. Verf.*, pages 526–531, 1998.

[72] N. Heumesser and F. Houdek. Towards systematic recycling of systems requirements. In *Proc. of the Int. Conf. on Soft. Eng. (ICSE)*, pages 512–519, 2003.

[73] IEEE. 1993-present. Consider the history of RE conferences, starting with 1993.

[74] Software engineering: Past and future. *IEEE Computer*, 39(10), October 2006.

[75] H. In, T. Rodgers, M. Deutsch, and B. Boehm. Applying winwin to quality requirements: A case study. In *Proc. of the Int. Conf. on Soft. Eng. (ICSE)*, pages 555–564, 2001.

[76] D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. MIT Press, 2006.

[77] M. Jackson and P. Zave. Domain descriptions. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 54–64, 1993.

[78] R. Janicki, D. L. Parnas, and J. Zucker. Tabular representations in relational documents. *Relational methods in computer science*, pages 184–196, 1997.

[79] R. Jeffords and C. Heitmeyer. Automatic generation of state invariants from requirements specifications. In *Proc. of SIGSOFT Found. on Soft. Eng. (FSE)*, pages 56–69, 1998.

[80] R. D. Jeffords and C. L. Heitmeyer. A strategy for efficiently verifying requirements. In *Proc. of SIGSOFT Found. on Soft. Eng. (FSE)*, pages 28–37, 2003.

[81] H. Kaiya and M. Saeki. Using domain ontology as domain knowledge for requirements elicitation. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 186–195, 2006.

[82] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[83] S. Konrad and B. H. Cheng. Facilitating the construction of specification pattern-based properties. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 329–338, 2005.

[84] S. Konrad and B. H. C. Cheng. Real-time specification patterns. In *Proceedings of the International Conference on Software Engineering (ICSE05)*, pages 372–381, St Louis, MO, USA, May 2005.

[85] S. Konrad, B. H. C. Cheng, and L. A. Campbell. Object analysis patterns for embedded systems. *IEEE Trans. on Soft. Eng.*, 30(12):970–992, 2004.

[86] J. Kramer and J. Magee. The evolving philosophers problem: Dynamic change management. *IEEE Trans. Softw. Eng.*, 16(11):1293–1306, 1990.

[87] J. Kramer and J. Magee. Analysing dynamic change in software architectures: a case study. In *Proc. of 4th IEEE International Conference on Configuratble Distributed Systems*, Annapolis, May 1998.

[88] S. Krishnamurthi, M. C. Tschantz, L. A. Meyerovich, and K. Fisler. Verification and change-impact analysis of access-control policies. In *Proc. of the Int. Conf. on Soft. Eng. (ICSE)*, pages 196–205, 2005.

[89] T. Kuhn. chapter The Nature and Necessity of Scientific Revolutions. University of Chicago Press, 1962. Book chapter transcribed by Andy Blunden in 1998; proofed and corrected March 2005.

[90] S. Kulkarni and K. Biyani. Correctness of component-based adaptation. In *Proceedings of the International Symposium on Component-based Software Engineering*, May 2004.

[91] S. Lauesen. Cots tenders and integration requirements. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 166–175, 2004.

[92] E. Letier and A. van Lamsweerde. Deriving operational software specifications from system goals. In *Proc. of SIGSOFT Found. on Soft. Eng. (FSE)*, pages 119–128, 2002.

[93] S. Liaskos, Alexei, Y. Yu, E. Yu, and J. Mylopoulos. On goal-based variability acquisition and analysis. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 76–85, 2006.

[94] W. J. Lloyd, M. B. Rosson, and J. D. Arthur. Effectiveness of elicitation techniques in distributed requirements engineering. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 311–318, 2002.

[95] R. Lutz and I. C. Mikulski. Operational anomalies as a cause of safety-critical requirements evolution. *J. of Sys. and Soft.*, pages 155–161, 2003.

[96] R. Lutz, A. Patterson-Hine, S. Nelson, C. R. Frost, D. Tal, and R. Harris. Using obstacle analysis to identify contingency requirements on an unpiloted aerial vehicle. *Req. Eng. J.*, 12(1):41–54, 2006.

[97] J. Magee, N. Pryce, D. Giannakopoulou, and J. Kramer. Graphical animation of behavior models. In *Proc. of the Int. Conf. on Soft. Eng. (ICSE)*, pages 499–508, 2000.

[98] N. Maiden and S. Robertson. Developing use cases and scenarios in the requirements process. In *Proc. of the Int. Conf. on Soft. Eng. (ICSE)*, pages 561–570, 2005.

[99] N. Maiden and S. Robertson. Integrating creativity into requirements processes: Experiences with an air traffic management system. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 105–116, 2005.

[100] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng. Composing adaptive software. *IEEE Computer*, 37(7):56–64, 2004.

[101] W. E. McUmber and B. H. Cheng. A general framework for formalizing uml with formal languages. In *Proc. of the Int. Conf. on Soft. Eng. (ICSE)*, pages 433–442, 2001.

[102]

[103] H. Mills, M. Dyer, and R. Linger. Cleanroom software engineering. *IEEE Software*, 4(5), September 1987.

[104] D. L. Moody, G. Sindre, T. Brasethvik, and A. Solvberg. Evaluating the quality of information models: Empirical testing of a conceptual model quality framework. In *Proc. of the Int. Conf. on Soft. Eng. (ICSE)*, pages 295–307, 2003.

[105] A. Moreira, A. Rashid, and J. Araujo. Multi-dimensional separation of concerns in requirements engineering. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 285–296, 2005.

[106] National Science Foundation. Program solicitation, nsf-07-505, science of design program, 2006.

[107] C. Nentwich, W. Emmerich, A. Finkelstein, and E. Ellmer. Flexible consistency checking. *ACM Trans. on Soft. Eng. & Meth.*, 12(1):28–63, 2003.

[108] R. Neville, A. Sutcliffe, and W.-C. Chang. Optimizing system requirements with genetic algorithms. In *IEEE World Congress on Computational Intelligence*, pages 495–499, May 2002.

[109] J. Niu, J. Atlee, and N. Day. Template semantics for model-based systems. *IEEE Trans. on Soft. Eng.*, 29(10):866–882, 2003.

[110] L. Northrup, P. Feiler, R. P. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, D. Schmidt, K. Sullivan, and K. Wallnau. *Ultra-Large-Scale Systems: The Software Challenge of the Future*. Software Engineering Institute, Carnegie Mellon, 2006.

[111] B. Nuseibeh and S. Easterbrook. Requirements engineering: a roadmap. In *Proc. of the Int. Conf. on Soft. Eng. (ICSE)*, pages 35–46, 2000.

[112] B. Nuseibeh, J. Kramer, and A. Finkelstein. Viewpoints: meaningful relationships are difficult! In *Proc. of the Int. Conf. on Soft. Eng. (ICSE)*, pages 676–683, 2003.

[113] S. P. Overmyer, B. Lavoie, and O. Rambow. Conceptual modeling through linguistic analysis using lida. In *Proc. of the Int. Conf. on Soft. Eng. (ICSE)*, pages 401–410, 2001.

[114] D. L. Parnas. Software engineering programmes are not computer science programmes. *Ann. Soft. Eng.*, 6(1):19–37, 1999.

[115] D. L. Parnas and J. Madey. Functional documents for computer systems. *Sci. of Comp. Prog.*, 25(1):41–61, 1995.

[116] D. L. Parnas and D. M. Weiss. Active design reviews: principles and practices. *J. Sys. Soft.*, 7(4):259–265, 1987.

[117] Y. Pisan. Extending requirement specifications using analogy. In *Proc. of the Int. Conf. on Soft. Eng. (ICSE)*, pages 70–76, 2000.

[118] A. Porter and L. Votta. Comparing detection methods for software requirements inspections: A replication using professional subjects. *Empir. Soft. Eng.*, 3(4):355–379, 1998.

[119] C. Potts. Metaphors of intent. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 31–39, 2001.

[120] C. Potts, K. Takahashi, and A. Antón. Inquiry-based requirements analysis. *IEEE Soft.*, 11(2):21–32, 1994.

[121] S. T. Redwine, Jr. and W. E. Riddle. Software technology maturation. In *IEEE International Conference on Software Engineering*, pages 1989–200, May 1985.

[122] R. Regnell, L. Karlsson, and M. Host. An analytical model for requirements selection quality evaluation in product software development. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 254–263, 2003.

[123] M.-O. Reiser and M. Weber. Managing highly complex product families with multi-level feature trees. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 146–155, 2006.

[124] S. Robertson and J. Robertson. *Mastering the Requirements Process*. Addison-Wesley, 1999.

[125] W. N. Robinson, S. D. Pawlowski, and V. Volkov. Requirements interaction management. *ACM Comp. Sur.*, 35(2):132–190, 2003.

[126] C. Rolland and N. Prakash. Matching erp system functionality to customer requirements. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 66–75, 2001.

[127] M. Sabetzadeh and S. Easterbrook. Traceability in viewpoint merging: a model management perspective. In *Proc. of the Int. Work. on Trace. in Emerg. Forms of Soft. Eng.*, pages 44–49, 2005.

[128] M. Sabetzadeh and S. Easterbrook. View merging in the presence of incompleteness and inconsistency. *Req. Eng. J.*, 11(3):174–193, 2006.

[129] P. Sawyer, P. Rayson, and K. Cosh. Shallow knowledge as an aid to deep understanding in early phase requirements engineering. *IEEE Trans. on Soft. Eng.*, 31(11):969–981, 2005.

[130] K. Schmid. The product line mapping approach to defining and structuring product portfolios. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 219–226, 2002.

[131] R. Settimi, E. Berezhanskaya, O. BenKhadra, S. Christina, and J. Cleland-Huang. Goal-centric traceability for managing non-functional requirements. In *Proc. of the Int. Conf. on Soft. Eng. (ICSE)*, pages 362–371, 2005.

[132] H. Sharp, A. Finkelstein, and G. Galal. Stakeholder identification in the requirements engineering process. In *Proc. of the 10th Int. Work. on Datab. & Exp. Sys. Appl.*, pages 387–391, 1999.

[133] M. Shaw. What makes good research in software engineering? *International Journal of Software Tools for Technology Transfer*, 4(1):1–7, 2002.

[134] A. Silva. Requirements, domain and specifications: A viewpoint-based approach to requirements engineering. In *Proc. of the Int. Conf. on Soft. Eng. (ICSE)*, pages 94–104, 2002.

[135] G. Sindre and A. Opdahl. Templates for misuse case description. In *Proc. of the Int. Work. on Req. Eng,: Found, for Soft. Qual.*, pages 125–136, 2001.

[136] I. Sommerville, T. Rodden, P. Sawyer, R. Bentley, and M. Twidale. Integrating ethnography into the requirements engineering process. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 165–181, 1993.

[137] T. Sreemani and J. M. Atlee. Feasibility of model checking software requirements. In *Conf. on Comp. Ass.*, pages 77–88. National Institute of Standards and Technology, 1996.

[138] J. Strassner. Autonomics: A critical and innovative component of seamless mobility. Motorola Technology Position Paper.

[139] A. Sutcliffe, W.-C. Chang, and R. Neville. Evolutionary requirements analysis. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 264–273, 2003.

[140] A. Sutcliffe, S. Fickas, and M. M. Sohlberg. Pc-re a method for personal and context requirements engineering with some experience. *Req. Eng. J.*, 11(3):1–17, 2006.

[141] A. Taleghani and J. M. Atlee. Semantic variations among uml statemachines. In *ACM/IEEE Int. Conf. on Model Driven Eng. Lang. and Sys.*, pages 245–259, 2006.

[142] J. M. Thompson, M. P. E. Heimdahl, and S. P. Miller. Specification-based prototyping for embedded systems. In *Proc. of SIGSOFT Found. on Soft. Eng. (FSE)*, pages 163–179, 1999.

[143] S. Uchitel and M. Chechik. Merging partial behavioural models. In *Proc. of SIGSOFT Found. on Soft. Eng. (FSE)*, pages 43–52, 2004.

[144] S. Uchitel, J. Kramer, and J. Magee. Negative scenarios for implied scenario elicitation. In *Proc. of SIGSOFT Found. on Soft. Eng. (FSE)*, pages 109–118, 2002.

[145] S. Uchitel, J. Kramer, and J. Magee. Behaviour model elaboration using partial labelled transition systems. In *Proc. of SIGSOFT Found. on Soft. Eng. (FSE)*, pages 19–27, 2003.

[146] S. Uchitel, J. Kramer, and J. Magee. Synthesis of behavioral models from scenarios. *IEEE Trans. on Soft. Eng.*, 29(2):99–115, 2003.

[147] S. Uchitel, J. Kramer, and J. Magee. Incremental elaboration of scenario-based specifications and behavior models using implied scenarios. *ACM Trans. on Soft. Eng. & Meth.*, 13(1):37–85, 2004.

[148] H. T. Van, A. van Lamsweerde, P. Massonet, and C. Ponsard. Goal-oriented requirements animation. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 218–228, 2004.

[149] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[150] A. van Lamsweerde. Goal-oriented requirements engineering: a guided tour. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 249–263, 2001.

[151] A. van Lamsweerde. Elaborating security requirements by construction of intentional anti-models. In *Proc. of the Int. Conf. on Soft. Eng. (ICSE)*, pages 148–157, 2004.

[152] A. van Lamsweerde and E. Letier. Handling obstacles in goal-oriented requirements engineering. *IEEE Trans. on Soft. Eng.*, 26(10):978–1005, 2000.

[153] K. S. Wasson. A case study in systematic improvement of language for requirements. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 6–15, 2006.

[154] K. S. Wasson, K. N. Schmid, R. R. Lutz, and J. C. Knight. Using occurrence properties of defect report data to improve requirements. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 253–262, 2005.

[155] M. Weber and J. Weisbrod. Requirements engineering in automotive development experiences and challenges. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 331–340, 2002.

[156] F. Weil. Hardware/software integration through modeling and automatic code generation. Presentation at USC Center for Software Engineering, Annual Research Review, 2006.

[157] M. Weiss and B. Esfandiari. On feature interactions among web services. In *ICWS '04: Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, page 88, Washington, DC, USA, 2004. IEEE Computer Society.

[158] J. Whittle, J. Saboo, and R. Kwan. From scenarios to code: An air traffic control case study. In *Proc. of the Int. Conf. on Soft. Eng. (ICSE)*, pages 490–497, 2003.

[159] J. Whittle and J. Schumann. Gnerating statechart designs from scenarios. In *Proc. of the Int. Conf. on Soft. Eng. (ICSE)*, pages 314–323, 2000.

[160] P. Zave. Feature interactions and formal specifications in telecommunications. *Computer*, 26(8):20–29, 1993.

[161] P. Zave. Classification of research efforts in requirements engineering. *ACM Comp. Sur.*, 29(4):315–321, 1997.

[162] C. Zhang and H.-A. Jacobsen. Resolving feature convolution in middleware systems. *SIGPLAN Not.*, 39(10):188–205, 2004.

[163] J. Zhang and B. H. C. Cheng. Specifying adaptation semantics. In *WADS '05: Proceedings of the 2005 workshop on Architecting dependable systems*, pages 1–7, St. Louis, Missouri, May 2005. ACM Press.

[164] J. Zhang and B. H. C. Cheng. Model-based development of dynamically adaptive software. In *Proceedings of International Conference on Software Engineering (ICSE'06)*, Shanghai,China, May 2006.