IEEE Copyright Notice

Published in: ***Proceedings of the ACM/IEEE International Conference on Software Engineering (ICSE'07 Future of Software Engineering),*** May 2007

"Research Directions in Requirements Engineering"

Cite as:

Betty H. C. Cheng and Joanne M. Atlee. 2007. Research Directions in Requirements Engineering. In 2007 Future of Software Engineering (FOSE '07). IEEE Computer Society, Washington, DC, USA, 285-303.

BibTex:

@inproceedings{Cheng:2007:RDR:1253532.1254725,
 author = {Cheng, Betty H. C. and Atlee, Joanne M.},
 title = {Research Directions in Requirements Engineering},
 booktitle = {2007 Future of Software Engineering},
 series = {FOSE '07},
 year = {2007},
 pages = {285--303}
}

DOI: https://doi.org/10.1109/FOSE.2007.17

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 current requirements engineering (RE) research and identifies future research directions suggested by emerging systems. First, the paper overviews the state of the art in RE research. The research is considered with respect to technologies developed to address specific requirements 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 of the future. Third, we review several strategies for performing RE research, to help delineate the scope of future research directions. Finally, within the context of these RE research challenges and 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 [124, 127]. *Software requirements* comprise these needs, and *requirements engineering (RE)* is the process by which the requirements are determined. Successful RE involves understanding the needs of users, customers, and other stakeholders; understanding the contexts in which the to-be-developed software will be used; modeling, analyzing, negotiating, and documenting the stakeholders' requirements; validating that the documented requirements match the negotiated requirements; and managing requirements evolution[1]. It is important to note that requirements encompass more than desired functionality. Users increasingly demand systems that are usable, reliable, secure, and economical. Moreover, product developers want to be able to adapt and enhance products rapidly, in response to changing needs and environmental conditions. As such, requirements activities are multi-disciplinary, drawing on research and experience in computer science, mathematics, engineering, human-computer interaction, and social and cognitive sciences.

In this paper, we offer our views on the research directions in requirements engineering. The paper builds on Nuseibeh and Easterbrook's paper [124], hereafter referred to as the "2000 RE Roadmap Paper", from the Future of Software Engineering track at ICSE 2000 [66]. Whereas the 2000 RE Roadmap Paper focused on current research in requirements engineering, this paper concentrates on research directions and identifies RE challenges posed by emerging and future software needs. We start, in Section 2, with an overview of the essential difficulties in requirements engineering. In Section 3, we provide a summary of the state of the art of RE knowledge and research, and in Section 4, we enumerate general research strategies for advancing the state of the art. The strategies range from revolutionary research to empirical evaluation to codifying proven solutions. In Section 5, we highlight what we consider to be RE research *hotspots*: the most pressing needs and grand challenges in RE research. Some hotspot topics are natural extensions to existing research technologies, whereas others arise as RE aspects of predicted software needs. We conclude with strategic recommendations for improving the research infrastructure for RE researchers, so that they can make better progress on addressing these problems.

2. Why Requirements Engineering is Hard

In general, the research challenges faced by the requirements engineering community are distinct from those faced by the general software engineering community, because requirements reside primarily in the problem space

[1] In addition, there are a number of software-engineering activities that are based on requirements information, such as cost estimation, project planning, and requirements-based derivations of architectures, designs, code, and test cases. Although these activities are "related" to a system's requirements, they play at most a minor role in determining and agreeing on the system's requirements; as such, we consider them to be outside the scope of requirements engineering.

whereas other software artifacts reside primarily in the solution space. That is, *requirements* descriptions, ideally, are written entirely in terms of the environment, describing how the environment is to be affected by the proposed system. In contrast, other software artifacts focus on the behavior of the proposed system, and are written in terms of internal software entities and properties. Stated another way, requirements engineering is about defining precisely the *problem* that the software is to solve (i.e., defining *what* the software system is to do), whereas other SE activities are about defining and refining a proposed software *solution*.

Several consequences follow from this distinction that cause requirements engineering to be inherently difficult:

- Requirements analysts start with ill-defined, and often conflicting, ideas of what the proposed system is to do, and must progress towards a single, detailed, technical specification of the system.
- Requirements artifacts have to be understood and usable by domain experts and other stakeholders, who may not be knowledgeable about computing. Thus, requirements notations and processes must maintain a delicate balance between producing descriptions that are intuitive and suitable for a non-computing audience and producing technical documents that are precise enough for downstream developers.
- The requirements problem space is less constrained than the software solution space (in fact, it is the problem definition that helps to delimit the solution space). As such, there are many options to consider and decisions to make about requirements, such as negotiating requirements, prioritizing requirements, identifying the system boundaries, resolving conflicts, setting acceptance criteria, and so on.
- One means of simplifying the problem space is to constrain the environmental conditions under which the system is expected to operate (though this constraint is relaxed in self-managing systems – see Section 5.5). In such cases, reasoning about requirements involves reasoning about the combined behavior of the proposed system and the environment.
- Reasoning about the environment includes identifying not only assumptions about the normal behavior of the environment, but also about possible threats or hazards that the environment could pose to the system.

Due to all of the above, requirements activities, in contrast to other software-engineering activities, may be more iterative, involve many more players with more heterogeneous backgrounds and expertise, and require a more intensive effort to integrate disparate needs into a consensus-based specification.

3. State of the Art of RE Research

In this section, we summarize the state of the art of RE knowledge and research, as a baseline from which to explore future research directions. This section can be viewed as an update to the 2000 RE Roadmap Paper [124], in that it incorporates advances made in the intervening seven years. For consistency with the 2000 RE Roadmap Paper, we adopt much of the terminology used in that paper.

To provide a visual map of RE research results, we organize 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. The research space is roughly decomposed into four categories of requirements problems (elicitation, modeling, analysis and validation, and requirements management) and three types of research contributions (notations, techniques, and evaluation studies). This decomposition is comparable to the top-level decomposition in Zave's proposed scheme for classifying RE research [182]. The resulting matrix is shown in Table 2.

The rest of this section is organized by requirements problem, and thus overviews the contents of the matrix by column. The exception is that all of the evaluation-based research (listed in the bottom row of the matrix) is discussed collectively at the end of the section.

Elicitation. Requirements elicitation comprises activities that enable the understanding of the goals, objectives, and motives for building a proposed software system. Elicitation also involves identifying the requirements that the resulting system must satisfy in order to achieve these goals. The requirements to be elicited may 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 innovation [2] (e.g., mass-market software). As such, most of the research in elicitation focuses on techniques for improving the precision, accuracy, and variety [72] of the requirements details:

- Techniques for *identifying stakeholders* [150] help to ensure that everyone who may be affected by the software is consulted during elicitation.
- *Analogical techniques*, like norms, metaphors [133], and personas [10, 38], help stakeholders to consider more deeply and be more precise about their requirements.
- Observation, apprenticeship [19], ethnography [156], and other *contextual techniques* [36, 159] 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.

[2] Innovations are inventions that people are willing to purchase.

Table 1. Matrix Summarizing Research in Requirements Engineering

Problems	Requirements Contributions		
	Notations	**Methodologies, Patterns, Strategies**	**Analyses, Tools**
Elicitation	Goals [21, 170] Policies [20] Scenarios [1, 35, 49] Agents [106, 180] Anti-models [155, 164, 171] Nonfunctional requirements [31, 70]	Identifying stakeholders [150] Metaphors [130, 133] Persona [10, 38] Contextual requirements [36, 159] Inventing requirements [72, 115]	Animation [82, 113, 168] Prototyping [51] Simulation [162] Invariant generation [91]
Modeling	Object models [86] Behavioral models [90, 165] Domain descriptions [12] Logics [52] Notation Semantics [117, 122, 161, 135]	RE reference model [75, 76, 128] Model elaboration [167] Viewpoints [125, 153] Patterns [56, 87, 97, 169] Modeling facilitators [7, 34, 95, 126] Formalization heuristics [20, 68] Methodologies [16]	Model merging [145, 163] Model synthesis [4, 41, 107, 166, 179] Model composition [79]
Requirements Analysis		Negotiation [85] Aligning requirements with COTS [6, 142] Conflict management [141]	Linguistic analysis [17, 30, 143, 173] Ontologies [93] Checklists [174] Conflict analysis [28, 78] Obstacle analysis [112, 172] Risk management [64] Causal order analysis [13] Prioritization [119] Variability analysis [74, 108, 109] Requirements selection [137, 158]
Validation & Verification	Property languages [15, 105] Model formalisms [24, 53] Object models [86]		Simulation [162] Animation [82, 113, 168] Invariant generation [91] Consistency checking [60, 81, 120] Inspections [62, 129] Model checking [29, 57, 157] Model satisfiability [86]
Requirements Management	Variability modeling [23, 40, 138, 148]	Scenario management [3] Feature management [176]	Traceability [33, 80, 144, 149] Impact analysis [101] Stability analysis [25]

- *Feedback techniques* use models, model animations [82, 113, 168], prototypes [51], 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 [115], help to identify nonessential requirements that make the final product more appealing.

Modeling. In requirements modeling, a project's requirements or specification is expressed in terms of one or more modeling notations. Modeling notations help to raise the level of abstraction in requirements descriptions by providing a vocabulary and structural rules that more closely match – better than natural language does – the entities, relationships, behavior, and constraints of the problem being modeled. Requirements models have a variety of uses. Early-phase requirements models help to catalyze discussion, and are useful in exploring and learning about the stakeholders' needs. Such exploratory models, like use cases, scenarios, enterprise models, and some policy [20] and goal notations [21], tend to be informal and intuitive, to facilitate early feedback from stakeholders. Early models also need to be inexpensive to create and maintain, so that specifiers can keep them up-to-date as the requirements evolve. In contrast, late-phase requirements models tend to be more precise, complete, and unambiguous. The process of creating precise models helps to evoke details that were missed in the initial elicitation. The resulting (more complete) models can be used to communicate the requirements to downstream developers.

Each modeling notation is designed to elicit or record specific details about the requirements, such as what data the software is to maintain, functions on the data, responses to inputs, or constraints on data or behavior. Of these, scenario-based notations [1, 4, 41, 49, 163, 166, 167, 179] 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 thus they lend themselves to a plethora 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* [75, 76, 128] decompose requirements-related descriptions into the stakeholders' *requirements*, the *specification* of the proposed system, and assumptions made about the system's *environment*. In addition, they establish correctness criteria for verifying that the specified system will meet the requirements. In contrast, the *viewpoints* approach [125, 153] retains each stakeholder's requirements in separate models, and the synthesis of a consistent global model that captures all of the stakeholders' concerns is delayed until conflicts can be resolved knowledgeably.
- *Model patterns* [56, 97, 169] encode common solutions to complex modeling problems. The RE community is also working on tools [34, 95, 126] to help specifiers apply these patterns.
- *Model transformations* combine or manipulate existing models to derive new models. For example, model synthesis [4, 41, 107, 166, 179] and model composition techniques [79] integrate complementary submodels into a composite model. In contrast, model merging techniques [145, 163] unify different views of the same problem. There is also preliminary work on heuristics for formalizing natural-language policies [20] and goal models [68].

Several of the above-mentioned projects directly address challenges raised in the 2000 RE Roadmap Paper [124]. For example, heuristics for formalizing informal models and tools that map constrained natural-language expressions to formal representations [34, 95, 126] 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 [135, 61, 71, 117, 161]. In addition, significant advances have been made in the modeling and analysis of nonfunctional requirements [31] and in establishing objective fit criteria for how well an eventual system must achieve various nonfunctional properties [70]. On the other hand, there has been little progress on new notations for modeling environment descriptions and assumptions [12]. Instead, existing notations like functions [128], 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 techniques for detecting errors in models, where an "error" can be ambiguity [17, 63, 93, 147, 173], inconsistency [60, 81, 120, 26], an unknown interaction among requirements [28, 78, 141], a possible obstacle to requirements satisfaction [112, 172], or missing assumptions [13].

Requirements validation ensures that models and documentation accurately express the stakeholders' needs. Unlike the above analyses, which check that a software specification adheres to precise well-formedness criteria, validation is normally a subjective evaluation of the specification with respect to informally described or undocumented requirements. As such, validation usually requires stakeholders to be directly involved in reviewing the requirements artifacts [143]. Research in this area focuses on improving

the information provided to the stakeholder for feedback, including animations [82, 113, 168], simulations [162], and derived invariants [91].

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 [29, 57, 157] checks behavioral models against temporal-logic properties about execution traces; and model satisfiability [86] checks that there exist valid instantiations of constrained object models, and that operations on object models preserve invariants.

The notations listed in the "Analysis, Validation, Verification" column of Table 2 represent formalisms for verifying requirements. These notations are used to simplify and abstract the structure of a requirements model to be verified [24, 53, 86], or to express verifiable properties of the requirements model [15, 105]. In contrast to modeling notations, their primary purpose is to facilitate automated verification rather than to communicate or document requirements.

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 engineering process. 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, visualization, and analysis techniques to help managers select an optimal combination of requirements to be implemented [74, 109, 137, 119, 158]; identify acceptable off-the-shelf solutions [104, 142]; and determine the maturity and stability of elicited requirements, so that the requirements most likely to change can be isolated [25]. Other researchers are investigating how to improve the ability to estimate the cost of implementing the selected requirements [37], or the impact of new or modified requirements [101]. 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 [33, 80, 144, 149, 116]. Lastly, the basic management of requirements has become a challenge and has inspired research on techniques to organize large numbers of requirements [3] that are globally distributed [45], and that are at different phases in development in different product variants [176].

Evaluation Studies. Most RE research evaluation takes the form of proofs-of-concept or pilot studies. However, evaluation results that are significant enough to be major research contributions on their own (such as those listed in Table 2) tend to be case studies or field studies that evaluate how well research ideas work when applied to industrial-strength problems[3] [85, 110, 114, 118, 178] or in industrial settings [42, 44, 67]. Several recent research projects evaluate how well requirements notations and techniques apply to, or can be adapted to, domain-specific problems, such as security [171], semantic webs [54], and user interfaces [18]. Additionally, there have been a few comparative studies that compare the effectiveness of competing elicitation techniques [50, 58], specification notations![11], and inspection techniques [132]. Finally, there have also been some post-mortem analyses on how requirements evolved in real-world systems [9, 111].

4. Research Strategies

In this section, we discuss ways of advancing the state of the art of RE research. We review several major strategies for conducting research, and look at how each have or might be applied to requirements-related research. The strategies range from inventing new disruptive ideas and technologies, to improving on existing research results, to adapting previous results to a new context, to evaluating or comparing technologies. Each strategy attempts to achieve a slightly different research objective, but all contribute in some way to advancing the state of the art, either by adding new knowledge or by improving the maturity of previous work.

Our collection of research strategies is synthesized from a number of different sources, including Shaw's overview of criteria for good research in software engineering [152], Redwine and Riddle's review of software technology maturation [136], Basili's review of research paradigms [14], and the combined experience of both authors. Table 2 introduces and briefly defines the eight research strategies that we discuss below. The strategies are listed in order of increasing maturity of the research results.

Paradigm Shift. A *paradigm shift* is a revolutionary solution that introduces radically new ideas or technologies to tackle a new or existing problem. A paradigm shift may be called for when researchers can no longer make progress on an important problem by extending or adapting existing solutions. Typically, there are two means by which a paradigm shift happens: push and pull. A paradigm shift is

[3] We use the term "industrial-strength" problems/projects to refer project data that has characteristics of industrial data, such as size, complexity, and/or domain-specific properties.

Table 2. Enumeration of research strategies

Research Strategy	Definition
Paradigm Shift:	Dramatically change the way of thinking, on the order of a revolution in knowledge or technology. The change typically ignites some disruption in practices, while the benefits of the paradigm shift become recognized and the technology is adopted.
Leverage other disciplines:	Leverage and recast principles, practices, processes, or techniques from another discipline.
Leverage new technology:	Make advances by leveraging new tools or technology.
Evolutionary:	Make progressive improvements to existing research solutions and techniques.
Domain-specific:	Develop a new solution or technique that applies narrowly to a specific problem domain.
Generalization:	Generalize an existing solution or technique, so that it applies to a broader class of problems or data.
Engineering:	Develop processes or strategies that make it easier or cheaper to apply research solutions in practice.
Evaluation:	Evaluate existing research solutions – with respect to specified metrics, real or realistic problems, current practices, or related research results.

pushed onto a community when a new technology serendipitously makes major advances towards solving a problem for which it was not originally intended. A classic example of such a shift is the World Wide Web, which has significantly changed the way that society communicates and the way that services are delivered to consumers. An example of a paradigm shift that is currently underway is the shift toward global software development and, by extension, global requirements engineering; we discuss this further in Section 5.6.

Alternatively, a paradigm shift can be *pulled* when there is a real or a perceived crisis that cannot be solved by improving current ideas and techniques [102]. For example, object-based design conventions were invented in response to serious concerns about how to structure programs and data in a way that promoted modularity. As the design conventions gained popularity, they evolved into object-oriented programming methodologies, were codified in new design methods, and were eventually supported by new programming language constructs.

Paradigm shifts are rare and are usually unplanned; but when they occur, they can have tremendous impact on a field. A paradigm shift starts with some innovative change in the way that a particular problem is thought about. The change leads to disruptive innovations, which usually must mature before their benefits are recognized and appreciated enough to motivate rapid and widespread adoption. During the adoption process, practitioners may experience real or perceived setbacks as they acclimate to the new paradigm. In the case of the shift to object-based technology, programmers who were proficient in languages such as C and Cobol took several years to migrate to the object-based mindset and fully leverage its benefits – even with the availability of object-based languages, such as C++ and Java. Of course, nowadays, object-oriented techniques are prominent in the development of new application software.

Leverage other disciplines. An RE researcher can leverage another discipline by identifying the analogous relationships between the two disciplines and then recasting promising knowledge, philosophies, principles, or practices from the other discipline into solutions that are appropriate for requirements problems. For example, software engineering, as a discipline, emerged when researchers and practitioners attempted to manage the "software crisis" by borrowing and adapting from the engineering profession several ideas about design principles, development processes, and discipline. The concept of genetic algorithms leverages ideas from biology, in that the algorithms "evolve" by using feedback from previous computations to improve future computations. Sutcliffe et al. [121] use genetic algorithms to select an optimal set of components that satisfy a given set of fitness criteria for reliability requirements. As a third example, Sutcliffe and Maiden's work [160] in establishing a domain theory for RE draws heavily from cognitive science and the human use of analogical reasoning.

Leverage technology. Technological advances in computing and related fields can be combined and adapted to apply to problems in requirements engineering. In general, artificial intelligence, library science, information sci-

ence, cognitive psychology, linguistics, statistics, and mathematics are all fertile areas for ideas and techniques that are suitable for such adaptation. For example, Ambriola and Gervasi [7], and separately Overmeyer et al. [126], use natural-language processing techniques to parse textual requirements descriptions and to generate corresponding semi-formal models, such as data-flow diagrams and communication diagrams. They and other researchers [7, 17, 63, 147, 173] use linguistic-analysis techniques to detect possible ambiguities and unintended inconsistencies in textual or use-case requirements. Hayes et al. [80] and Cleland-Huang et al. [32] use information-retrieval techniques to automatically retrieve traceability links among requirements.

Evolutionary research. The antithesis of a paradigm shift is evolutionary research, in which the state of the art advances via incremental improvements to existing technologies. Although emerging software needs may pose new research challenges that the RE community will be called on to address, most 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 (as overviewed in Section 3), which were created in response to the problems that today's practitioners face.

In many ways, evolutionary research is about moving research technologies down the research-strategy ladder listed in Table 2. Existing notations and techniques can be extended, adapted, or generalized to address a broader class of problems. Current technologies can be supported by new methodologies, patterns, strategies, and tools that ease their use and help to promote their adoption by practitioners. Empirical research can determine the problems and contexts for which a technology is most effective, and can identify aspects that could be further improved.

Domain-specific. A researcher can sometimes make better progress by narrowing the scope of a requirements problem and studying it in the context of a particular application domain. For example, there is a paradigm shift towards more domain-specific specification languages that provide native facilities for describing important entities and behaviors in that domain and provide macros for eliding recurrent requirements details. Along these lines, the International Telecommunication Union (ITU) has standardized a number of specification, design, and testing languages; design methodologies; and interface specifications – all of which support software aspects of telecommunication systems.

Generalization. Successful domain-specific or organization-specific techniques can sometimes be generalized to be more broadly applicable. For example, many of the ideas in telecommunications notations, like Message Sequence Charts and the Specification and Description Language, have been incorporated into the more general Unified Modeling Language 2.0. As another example, techniques for avoiding, detecting, and resolving feature interactions, which were originally developed in the context of telephony features [79, 181], are now being studied and applied in other domains, such as Web services [177].

Engineering. A surprising number of research problems arise in the course of trying to apply requirements technologies in practice. Engineering as a research strategy looks at how to simplify and codify RE knowledge and techniques so that they can be readily adopted by practitioners and taught in classrooms. For example, visual formalisms [52, 55, 77, 90] ease the task of creating and reviewing precise specifications. Patterns not only help specifiers to create models [96, 97, 169] and express constraints [56], via instantiation and adaptation, but they also offer some level of uniformity and repeatability of such descriptions. Heuristics and strategies offer advice on how to use particular elicitation [8], modeling [20, 68], verification [92], or management technologies. Methodologies and processes provide guidance on how to integrate RE technologies and techniques to progress from an initial idea to a final specification document [134, 139]. One of the best known engineering-style research projects was 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 [90], hierarchical module structures, abstract interfaces, and new inspection strategies [129].

Evaluation. Proposed RE technologies become theories, solutions, or practices through evaluation-based research that demonstrate effectiveness. Evaluation techniques include experience, collection and analysis of data, field studies, case studies, controlled experiments, and analytical reasoning. Evaluation criteria range from qualitative or statistical metrics, to effectiveness in solving real or realistic problems, to comparisons with competing technologies. A mature RE technology should be evaluated on real-world applications or in an industrial setting, to assess its scalability, practicality, and ease of use [67, 85, 110, 114, 118, 178]. In contrast, comparative studies are effective in evaluating the relative strengths and weaknesses of competing solutions to a problem. Notable comparative studies have investigated the criteria for choosing a specification language [11] and the effectiveness of methods for inspecting requirements documents [132].

Evaluation-based research need not be a massive project. A case-study evaluation may be based on a single study involving an industrial-strength project, on replicated stud-

ies of the same project, on studies of multiple projects, or on a longitudinal study that spans several phases of a project. Even the development of new assessment criteria, such as appropriate benchmarks, are valuable research contributions.

5. RE Research Hotspots

As evidenced by the previous sections, the field of RE research is rich, and the scope of possible research directions is quite large. In addition, new RE research challenges may arise from emerging trends in software systems and predictions about future software needs Current trends and expressed needs include increasing scale of software systems, tighter integration between software and its environment, greater autonomy of software to adapt to its environment, increasing globalization of software development, and more agile and cost-effective development processes. These trends reflect changes in stakeholders' needs, and as such they directly affect RE processes and practices. In some cases, current technologies can accommodate the new trends in requirements. In other cases, the trends pose new research challenges in requirements engineering, or raise the priorities of longstanding research problems.

In this section, we highlight what we consider to be RE research *hotspots*: that is, those RE problems whose solutions are likely to have the greatest impact on software-engineering research and practice. Our list of problems is not meant to be exhaustive. There are multiple strategies for extending and maturing existing research technologies and for confronting new research challenges posed by future software needs. Rather, our list of hotspots is intended to highlight some of the more pressing needs and grand challenges in RE research.

5.1. Scale

Software systems are growing in size. Moreover, the "scale" of large-scale systems no longer refers simply to significant size, as in lines of code. Scale factors also include complexity, level of heterogeneity, number of sensors, and number of decentralized decision-making nodes. Yet another scale factor is variability, as software systems need to accommodate increasingly larger sets of requirements that vary with respect to changes in the software's environment. An example of emerging systems that exhibit many of these new scale factors are the ultra-large-scale (ULS) systems [123] proposed for next-generation military command and control systems. Other potential ULS systems include future intelligent transportation-management systems, critical infrastructure protection systems (e.g., systems managing power grids, bridges, telecommunication systems), integrated health-provising systems, and disaster-response systems.

Developing modeling, abstraction, and analysis techniques that scale well is critical in designing future ULSs. Current modeling paradigms and analysis techniques cannot effectively manage the degrees of scale, complexity, variability, and uncertainty that are exhibited in these systems. Requirements will come from many different stakeholders, involve multiple disciplines (e.g., sensors, scientific computation, artificial intelligence), and perhaps be presented at varying levels of abstraction. Thus, new abstractions, innovative decomposition strategies, standardized composition operators, and increased automation of RE tasks are all needed to cope with the complexity. In addition, better techniques are needed to merge these potentially vastly different types of requirements into a single coherent story. Detecting and resolving feature interactions and conflicts on such a large scale will pose a grand challenge to the RE community. Taken together, these problems call for new paradigms for thinking about, modeling, analyzing, and managing requirements.

5.2. Security

As computing systems become ever more pervasive and mobile, and as they automate and manage more consumer-critical processes and data, they increasingly become the targets 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, and are design or implementation results. Thus, the RE challenge for developing 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 [89].

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, analysis, and security communities to resolve.

5.3. Tolerance

Software is increasingly used to automate critical applications and services, such as transportation vehicles and systems, financial decisions and transactions, medical care, military command and control, and so on; in which security and assurance requirements are paramount. However, given the complexity of such systems, with respect to size, decentralized decision-making, and variability, the SE and RE communities may need to soften their views and expectations for security and correctness. Shaw [151] discusses the need to accept "sufficient correctness" for complex systems, instead of striving for absolute correctness that may lead to brittle systems.

> **Sufficient Correctness:** *The degree to which a system must be dependable in order to serve the purpose its user intends, and to do so well enough to satisfy the current needs and expectations of those users [151].*

When operating in an uncertain and dynamically changing environment, brittle systems tend to fail at the first encounter of adverse conditions. To avoid this problem, requirements elicitation should focus on requirements for acceptable behavior and on what it means for a system to be "healthy" [151]. One approach to relaxing the precision of correctness criteria is to specify (fault) tolerance requirements, which extend the ranges of acceptable behavior. For example, Wassyng et al. [175] have made some preliminary progress on specifying timing requirements in a way that is precise and yet captures allowable tolerances. Alternative approaches include focusing on negative requirements, which represent "unhealthy" conditions or behaviors that the system must avoid, and on requirements for diagnostic and recovery mechanisms.

5.4. Increased Reliance on the Environment

The increase in scale is partly due to the rise of systems of systems, consisting of software, hardware, and people, all of which may be loosely or tightly coupled together. For example, *cyber-physical systems* (CPSs) are a new generation of engineered systems in which computing and communication are tightly coupled with the monitoring and control of entities in the physical world [39]. Example cyber-physical systems include intelligent transportation and vehicle systems; automated manufacturing; critical infrastructure monitoring; disaster response; optimization of energy consumption; smart wearable attire [69] for health care, personal safety, and medical needs; and efficient agriculture [39].

Integrated systems pose particularly thorny requirements problems because of their coupling with and dependence on the physical environment. Such systems recast old RE problems of determining the software system's boundary into more complicated problems of assigning responsibilities: to the software system under consideration, to peer software systems, to hardware interface devices (which are increasingly programmable), and to human operators and users [106]. Moreover, the environment or context in which a software system will run is often the least understand and most uncertain aspect of a proposed system; and RE technologies and tools for reasoning about the integration of physical environment, interface devices, and software system are among the least mature.

In an integrated system, the correctness of a software system depends heavily on its interactions with peer software systems, hardware interfaces, and the physical and human environment. To reason about such a software system, it becomes necessary to formalize the properties of the environments with which the software will interoperate. Towards this end, better abstractions are needed to model the behaviors of physical and human entities and their interfaces with the computing elements. New domain-specific modeling languages may be needed to express these domain abstractions and knowledge; and new languages call for corresponding simulation, verification, and visualization techniques, to validate the modeled environment. Jackson [88] explores a number of challenges in modeling and reasoning about a software system's environment, including formalizing multiple aspects of the environment to support proofs about different aspects of the software, ensuring that the environment is in a "compatible state" when the system is initialized, and working with formalizations that are necessarily "imperfect...[discrete approximations] of continuous phenomena" [88].

Most importantly, there need to be better techniques for integrating models of the environment, interface devices, and software components. Computing devices tend to be modeled using discrete mathematics, such as logics and automata; physical devices tend to modeled using continuous mathematics, such as differential equations; and human-behavior modeling is an open problem, with researchers using a combination of goals, agents, relationship models, and performance moderator functions [154]. Researchers in the verification community are making progress on the modeling, simulation, and reasoning of hybrid models [5], but their work does not accommodate human-behavior models, and the scalability of techniques remains an elusive goal.

5.5. Self-Management

The difficulties of requirements engineering are made worse by the desire to create software systems that accommodate varying, uncertain, incomplete, or evolving requirements. For example, there is growing interest in *self-managing systems*, in which the software system is aware of

its context and is able to react and adapt to changes in either its environment or its requirements [100] – such as a mobile device, whose available services vary with the user's location and with the local service provider(s). Examples of such systems include *self-healing systems* that are able to recover dynamically from system failure, faults, errors, or security breaches; and *self-optimizing systems* that are able to optimize their performance dynamically with respect to changing operational profiles.

Self-management capabilities are essential in software systems that, once deployed, cannot be accessed physically or electronically. For example, a cyber-physical system (CPS) may have large numbers of physically distributed sensors that are placed in difficult-to-reach locations, such as power transformers, nuclear reactor cores, and hazardous or toxic sites. Moreover, these sensors are increasingly programmable. If developers are not able to access remote elements to perform software updates, then the elements will need to update and correct themselves.

In the simplest case, a self-managing system adapts its behavior at run-time by replacing the running system with a new target behavior selected from among a set of predefined behaviors. These systems will require different perspectives on what types of requirements information should be documented, in contrast to traditional approaches, which typically focus on a static set of goals or functionality. The RE research problems posed by such a system include

- Identifying and specifying thresholds for when the system should adapt
- Specifying variable sets of requirements
- Identifying correctness criteria for adaptive systems
- Verifying models of adaptive systems and their sets of possible behaviors
- Monitoring the system and environment, against the current requirements

There has been some preliminary work on modeling and verifying dynamic architectures [2, 27, 98, 99, 103], on specifying adaptive software [183, 184], and on run-time monitoring [65, 140, 146].

However, the above approach assumes that it is possible to predict and predefine the requirements for a complete set of target behaviors. Such predictions may not be possible, if the system is to recover dynamically from unexpected errors or attacks, or adapt at run-time to new environmental conditions or to new requirements that were not anticipated during development. In this case, what is needed is a *self-evolving* system that is able, at run-time, to derive new requirements and behaviors. Thus, the requirements analyst needs to specify how the system's requirements can evolve dynamically; specify abstract adaptation thresholds that allow for uncertainty and unanticipated environmental conditions; and verify the requirements-decision capabilities of the resulting system. Unfortunately, none of the existing modeling and verification techniques address the challenges posed by evolution, uncertainty, and incomplete information.

One research strategy would be to investigate whether ideas from other disciplines, such as biology, could be leveraged. Given that natural organisms are inherently able to respond to adverse and unexpected conditions, biological entities and systems may be suitable metaphors for dynamically adaptive software. *Biomimetics* comprises those techniques that attempt to imitate or simulate the behavior of natural organisms. For example, work at Michigan State University is exploring how digital evolution techniques can be extended to simulate a biological evolution process that discovers new unanticipated behavior, and thus new requirements, for potential target systems of dynamically adaptive systems [73, 94].

5.6. Globalization

Global software development is an emerging paradigm shift towards globally distributed development teams [84]. The shift is motivated by the desire to capitalize on global resource pools, decrease costs, exploit a 24-hour work day, and be geographically closer to the end-consumer [47]. The downside is increased risk of communication gaps. For example, elicitation and early modeling are collaborative activities that require the construction of a shared mental model of the problem and requirements. However, there is an explicit disconnect between this need for collaboration and the distance imposed by global development.

Globalization poses two main challenges to the RE research community. First, new or extended RE techniques are needed to support outsourcing of downstream development tasks, such as design, coding, and testing. Distance aggravates the gap between the requirements and development teams, particularly if the teams are from different organizations, have different cultures, or have different work environments. In particular, because geographic distance reduces team communication [83], ill-defined requirements are at risk of ultimately being misinterpreted, resulting in a system that does not address meet the stakeholders' needs. An a preliminary effort to narrow communication gaps, Bhat et al. [47] have proposed a framework based on a people-process-technology paradigm that describes best practices for negotiating goals, culture, processes, and responsibilities across a global organization.

The second challenge is to enable effective distributed RE. Future requirements activities will be globally distributed, since requirements analysts will likely be working with geographically distributed stakeholders and distributed development teams may work with in-house customers. As such, practitioners need techniques to facilitate and manage

distributed requirements elicitation, distributed modeling, distributed requirements negotiation, and the management of distributed teams – not just geographically distributed, but distributed in terms of time zone, culture, and language. Damian and her group are interested in distributed requirements negotiation, and have investigated how best to use and combine different media technology to facilitate negotiations and quality agreements [44, 46]. Sinha et al. have developed an Eclipse-based tool for distributed requirements engineering collaboration [47].

5.7. Methodologies, Patterns, and Tools

The transfer of RE technologies from research into practice would benefit from better advice on how to apply the technologies more systematically. The goals of this type of engineering-style research are to improve the productivity of the requirements analyst and to improve the quality of the resulting requirements artifacts. For example, just as specification patterns [56] help to ease the creation of logic specifications, research into idioms and patterns for other modeling problems and notations [96, 97, 169] would improve the productivity of modelers.

Similarly, modeling conventions, methodologies, and strategies all help to simplify RE techniques so that the techniques can be used successfully by ordinary practitioners. Because patterns and strategies are, or suggest, partial solutions, they help also to impose some level of uniformity and predictability in the resulting requirements descriptions.

Engineering-style research is also needed to investigate how to integrate requirements technologies into a coherent requirements process. Most research projects focus on a single RE problem, such as elicitation or traceability. As a result, the state of the art in RE research is a collection of notations and techniques that have been researched and evaluated in isolation, with little knowledge of how to combine techniques effectively. For example, despite the significant advances that have been made in requirements modeling and notations, there has been little work on how to interconnect various types of requirements models. What are needed are well-defined ways of interrelating requirements goals, scenarios, data, functions, state-based behavior, and constraints. Broy and his group have made some progress on this problem, in the form of a modeling theory that incorporates many of the above-mentioned modeling elements [22]. As an example of synergy among RE technologies, Ebert [59] shows via an analysis of several industrial-development projects that four product-management techniques – for composing teams, negotiating requirements, planning long-term product and feature releases, and tracking a product's status – are most effective at reducing scheduling delays when the techniques are used together. Further research is needed on how to integrate RE technologies, so that practitioners know how to apply individual technologies effectively and synergistically.

5.8. Requirements Reuse

Another approach to making RE tasks more prescriptive and systematic would be to facilitate the reuse of existing requirements artifacts. The most strategic form of requirements reuse is *product lining,* where related products are treated as a product family, and their co-development is planned from the beginning. The family's common requirements are collected in reusable templates that can be instantiated and adapted to derive the requirements for an individual product. A key RE challenge for product-line development includes strategic and effective techniques for analyzing domains; identifying opportunities for product lining; and identifying the scope, commonalities, and variabilities of a product line. A second challenge relates to how requirements for product lines are documented. Feature models [40, 131] are commonly used to model a product-line core, but they quickly proliferate when used to model product-line instantiations. A promising but untested solution to this challenge is multi-level feature trees [138].

Specification and modeling patterns, discussed in the previous section, are also a form of reuse, in that they codify reusable modeling structures. Problem frames [87] can be considered abstract patterns of context diagrams for common classes of software problems, and thus are also reusable. In addition, it may be possible to identify larger units of reusable requirements for particular domains or particular types of applications. The automotive industry has expressed interest in using such "generic, reusable requirements" in developing complex automotive systems.

A reusable requirement should be accompanied by standard pattern fields, such as context, problem addressed, consequences, properties, and so on. However, this is usually not enough information to facilitate effective use of the pattern or reusable artifact. Adapting an instantiated pattern so that it adequately fits the desired context is still a bit of an art. Pattern use would be easier and more successful if practitioners had better guidance and examples of how to apply and adapt individual patterns.

5.9. Agile RE Processes

5.10. Effectiveness of RE Technologies

The ultimate impact of RE research depends on how relevant the results are to industry's short- and long-term needs. So far, there has been surprisingly little evaluation as to how well RE research results address industrial problems. As mentioned in Section 3, most empirical RE research takes

the form of proof-of-concept studies or pilot studies, both of which qualitatively evaluate how well a proposed solution or technique applies to a single concrete problem. Such studies tend to be aimed at research audiences, and are intended to convince readers that the RE technologies under evaluation advance the state of the art. However, given that most studies report success, how is a practitioner to determine when a study reports a significant enough advance to warrant changes to the state of the practice, and how is a practitioner to select from among competing technologies?

Practitioners need hard evidence that a new technology is cost-effective, in order to justify the overhead, in training and in process documentation, of changing their development processes. In particular, practitioners would benefit greatly from empirical studies that assess the costs and benefits of using proposed technologies, assess the scope of problems to which research results can feasibly be applied, and compare the effectiveness of competing technologies. There have been a few studies along these lines. Damian et al. have conducted a series of surveys that evaluates the impact of requirements-related activities on productivity, risk management, and the quality of both requirements and downstream artifacts [42, 43, 48]. The Comparative Evaluation in Requirements Engineering (CERE) workshops investigate how to facilitate comparative studies, such as the development of suitable benchmarks. The Economics-Driven Software Engineering Research (EDSER) workshops investigate how to improve practitioners' abilities to make economics-based design and process decisions, such as whether or not to adopt new technologies. Such empirical research that evaluates requirements technologies in the context of industrial settings and practices would help to accelerate the transfer of research results into RE practice.

6. Recommendations and Conclusions

In this paper, we have described a number of exciting and challenging research directions in requirements engineering. Some of these directions are natural extensions of work already being performed by RE researchers, whereas others are major discontinuities due to fundamental changes in computing needs. All of the problems described above will require substantial effort in order to make significant progress towards effective solutions. To help alleviate this effort, we offer some recommendations of short- and long-term actions that the RE community could take, to position itself to make more rapid progress on these research problems.

There are five recommendations that the RE community could take immediate action on, to start improving the maturity of current requirements technologies:

- Researchers should work with practitioners. Such partnerships can help to ensure that researchers have a through understanding of the real problems that practitioners face.
- RE researchers should work with researchers and practitioners involved in downstream development, to establish stronger links between their respective artifacts. If the transition between RE tasks and other development tasks were more seamless, management would have a higher opinion of RE efforts, because the contributions that requirements knowledge and artifacts make towards achieving downstream milestones would be more concrete.
- RE researchers should not neglect evaluation and empirical research. For practitioners to consider adopting a given research technique, they much know how the technique compares with other similar techniques. Also, practitioners and their managers need to see that the technique can be applied to problems relevant to their organization, both from domain and scale perspectives.
- Industrial organizations should provide (sanitized) industrial-strength project data to researchers. It is especially critical that industry provide realistic data for ultra-large scale or cyber-physical systems to ensure that researchers tackle problems that are representative of those faced by practitioners. Researchers can use this data to guide the development and validation of their new techniques, thereby yielding more relevant and useful research results that explicitly address industrial needs.
- RE researchers and practitioners, together, should establish repositories of RE artifacts. Such repositories can serve as a resource for practitioners and educators to share best practices and exemplar artifacts. Repositories can also store requirements patterns for potential reuse, case studies that evaluate individual or composite RE techniques, benchmark data for evaluating competing technologies, and tools that support specific RE techniques.

The above actions would help the RE research community to make immediate progress on improving existing knowledge and techniques. In addition, there are some longer-term actions that would help to improve the community's research infrastructure and its ability to confront the challenges posed by emerging systems:

- The RE community needs to be proactive in identifying the RE research problems raised by new computing challenges. New challenges reflect changing stakeholders' needs. As such, RE researchers should be involved in the initial investigations of any new computing challenge, to help tease out the essential goals and

to assess their impact on RE tasks and downstream development activities.

- RE researchers and practitioners should form strategic partnerships. (This need is so critical that we felt it was worth repeating.) As partners, researchers and practitioners can collaborate on a shared vision of the important research problems, the formulation of viable solutions, and the transfer of these innovative solutions into practice. They can also develop a shared understanding of the limits of current technologies in addressing current and future problems.
- Researchers need to think beyond current RE and SE knowledge and capabilities, in order to make significant headway in addressing the challenges posed by emerging systems. They need to be willing to search for new solutions that may lead to paradigm shifts in RE practices, at the risk of possible failure.
- RE researchers should seek out collaborators from other disciplines to leverage successful techniques that could be used to address analogous challenges faced by cyber systems.
- RE academics need to educate the next generation of developers on RE problems and technologies. Unlike fields such as physics and mathematics, whose undergraduate curricula has changed little over the past fifty years, undergraduate education in computer science must evolve to keep up-to-date with advances in practices and current technologies. Students need curricula that combine the study of computing with the study of specialized application domains. They also need computing courses that teach them how to make design decisions that achieve requirements (e.g., modularity vs. performance requirements) in the context of the software's operating environment.

In conclusion, the RE research community has made significant progress along many fronts. At the same time, the demands placed on computing and the cyberinfrastructure have increased dramatically, raising many new critical RE research questions. For these reasons, it is an exciting time to be involved in RE research. Technologies that make significant advances in solving these problems are likely to lead to paradigm shifts that will impact many future generations of developers, computing systems, and the ultimate stakeholders – consumers.

Acknowledgements

The work of the first author has been supported by EIA-0000433, EIA-0130724, CDA-9700732, CCR-9901017, CNS-0551622, CCF-0541131, Department of the Navy, Office of Naval Research under Grant No. N00014-01-1-0744, Siemens Corporate Research, and a grant from Michigan State University's Quality Fund. The work of the second author has been supported by NSERC.

The authors greatly appreciate the feedback on earlier drafts of this paper by Philip K. McKinley. Several other people have also provided valuable input, including Heather Goldsby.

References

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

[2] 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.

[3] 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.

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

[5] R. Alur, T. Henzinger, G. Lafferriere, and G. Pappas. Discrete abstractions of hybrid systems. *Proc. of IEEE*, 88(7), July 2000.

[6] C. Alves and A. Finkelstein. Challenges in COTS decision-making: a goal-driven requirements engineering perspective. In *Proc. of the Int. Con. on Soft. Eng. and Know. Eng.*, pages 789–794, 2002.

[7] V. Ambriola and V. Gervasi. Processing natural language requirements. In *IEEE Int. Conf. on Aut. Soft. Eng.*, pages 36–45, 1997.

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

[9] 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.

[10] 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.

[11] M. A. Ardis, J. A. Chaves, L. J. Jagadeesan, P. Mataga, C. Puchol, M. G. Staskauskas, and J. V. Olnhausen. A framework for evaluating specification methods for reactive systems experience report. *IEEE Trans. on Soft. Eng.*, 22(6):378–389, 1996.

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

[13] 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 ACM SIGSOFT Found. on Soft. Eng. (FSE)*, pages 50–59, 2005.

[14] 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.

[15] P. Bellini, R. Mattolini, and P. Nesi. Temporal logics for real-time system specification. *ACM Comp. Sur.*, 32(1):12–42, 2000.

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

[17] D. Berry and E. Kamsties. Ambiguity in Requirements Specification. *Perspectives on Software Requirements*, chapter 2. Kluwer Academic Publishers, 2004.

[18] 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 IEEE Int. Conf. on Soft. Eng. (ICSE)*, pages 453–462, 2001.

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

[20] 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.

[21] 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.

[22] M. Broy. The 'grand challenge' in informatics: Engineering software-intensive systems. *IEEE Computer*, 39(10):72–80, 2006.

[23] S. Buhne, K. Lauenroth, and K. Pohl. Modelling requirements variability across product lines. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 41–52, 2005.

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

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

[26] L. A. Campbell, B. H. C. Cheng, W. E. McUmber, and R. E. K. Stirewalt. Automatically detecting and visualizing errors in uml diagrams˙ *Requirements Engineering Journal*, 37(10):74–86, October 2002.

[27] 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.

[28] 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.

[29] 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.

[30] F. Chantree, B. Nuseibeh, A. de Roeck, and A. Willis. Identifying nocuous ambiguities in natural language requirements. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 59–68, 2006.

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

[32] J. Cleland-Huang, R. Settimi, C. Duan, and X. Zou. Utilizing supporting evidence to improve dynamic requirements traceability. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 135–144, 2005.

[33] 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.

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

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

[36] 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.

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

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

[39] Cyber-Physical Systems. http://varma.ece.cmu.edu/cps, October 2006.

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

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

[42] 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.

[43] D. Damian, J. Chisan, L. Vaidyanathasamy, and Y. Pal. Requirements engineering and downstream software development: Findings from a case study. *Empirical Soft. Eng.*, 10(3):255–283, 2005.

[44] 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.

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

[46] D. Damian, F. Lanubile, and T. Mallardo. An empirical study of the impact of asynchronous discussions on remote synchronous requirements meetings. In *Fund. Appr. to Soft. Eng.*, pages 155–169, 2006.

[47] D. Damian and D. Moitra, editors. *IEEE Software*, volume 23. IEEE-CS Press, 2006. Special Issue on Global Software Development.

[48] D. Damian, D. Zowghi, L. Vaidyanathasamy, and Y. Pal. An industrial case study of immediate benefits of requirements engineering process improvement at the australian center for unisys software. *Empirical Soft. Eng.*, 9(1-2):45–75, 2004.

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

[50] 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.

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

[52] 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.

[53] 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.

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

[55] 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.

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

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

[58] 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.

[59] C. Ebert. Understanding the product life cycle: Four key requirements engineering techniques. *IEEE Soft.*, 23(3):19–25, 2006.

[60] 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 ACM SIGSOFT Found. on Soft. Eng. (FSE)*, pages 186–195, 2001.

[61] A. Evans, J.-M. Bruel, R. France, K. Lano, and B. Rumbpe. Making UML precise. In *Proceedings of the OOPSLA98 Workshop on Formalizing UML. Why? How?* 1998.

[62] M. Fagan. Advances in software inspections. *IEEE Trans. on Soft. Eng.*, 12(7):744–751, 1986.

[63] 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.

[64] M. S. Feather. Towards a unified approach to the representation of, and reasoning with, probabilistic risk information about software and its system interface. In *Int. Sym. on Soft. Reliab. Eng.*, pages 391–402, 2004.

[65] S. Fickas and M. S. Feather. Requirements monitoring in dynamic environments. In *Proceedings of the Second IEEE International Symposium on Require ments Engineering*, page 140. IEEE Computer Society, 1995.

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

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

[68] 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.

[69] R. K. Ganti, P. Jayachandran, T. F. Abdelzaher, and J. A. Stankovic. SATIRE: A softwarwe architecture for smart atTIRE. In *Proc. of 4th International conference on Mobile Systems, Applications, and Services (Mobisys 2006)*, Uppsala, Sweden, June 2006.

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

[71] M. Gogolla, P. Ziemann, and S. Kuske. Towards an integrated graph based semantics for UML. In P. Bottoni and M. Minas, editors, *GT-VMT'2002 Graph Transformation and Visual Modeling Techniques, Barcelona, Spain, 11-12 October 2002*, volume 72(3) of *ENTCS*. Elsevier, 2002.

[72] L. Goldin and D. Berry. Abstfinder, a prototype abstraction finder for natural language text for use in requirements elicitation: Design, methodology, and evaluation. In *Proceedings of the First International Conference on Requirements Engineering*, pages 19–21, April 1994.

[73] H. Goldsby, D. Knoester, B. H. Cheng, P. McKinley, and C. Ofria. Digitally evolving models for dynamically adaptive systems. Technical Report MSU-CSE-07-4, Computer Science and Engineering, Michigan State University, East Lansing, Michigan, January 2007.

[74] 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.

[75] 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.

[76] 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.

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

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

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

[80] 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.

[81] 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.

[82] 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.

[83] J. Herbsleb and A. Mockus. An empirical study of speed and communication in globally distributed software development. *IEEE Trans. on Soft. Eng.*, 29(6):481–494, 2003.

[84] J. D. Herbsleb. Global software engineering: The future of socio-technical coordination. In *Future of Software Engineering 2007*. 2007.

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

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

[87] M. Jackson. *Problem Frames*. Addison-Wesley, 2003.

[88] M. Jackson. What can we expect from program verification? *IEEE Computer*, 39(10):65–71, 2006.

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

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

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

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

[93] 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.

[94] D. B. Knoester, P. K. McKinley, B. Beckmann, and C. A. Ofria. Evolution of leader election in populations of self-replicating digital organisms. Technical Report MSU-CSE-06-35, Computer Science and Engineering, Michigan State University, East Lansing, Michigan, December 2006.

[95] 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.

[96] 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.

[97] 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.

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

[99] 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.

[100] J. Kramer and J. Magee. Self-managed systems: An architectural challenge. In *Future of Software Engineering 2007*. IEEE-CS Press, 2007.

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

[102] T. Kuhn. *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.

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

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

[105] E. Letier, J. Kramer, J. Magee, and S. Uchitel. Fluent temporal logic for discrete-time event-based models. In *Proc. of ACM SIGSOFT Found. on Soft. Eng. (FSE)*, pages 70–79, 2005.

[106] E. Letier and A. van Lamsweerde. Agent-based tactics for goal-oriented requirements elaboration. In *Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE)*, pages 83–93, 2002.

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

[108] E. Letier and A. van Lamsweerde. Reasoning about partial goal satisfaction for requirements and design engineering. In *Proc. of ACM SIGSOFT Found. on Soft. Eng. (FSE)*, pages 53–62, 2004.

[109] 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.

[110] 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.

[111] 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.

[112] 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.

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

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

[115] 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.

[116] A. Marcus and J. Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *Proceedings of the International Conference on Software Engineering*, pages 125–135, 2003.

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

[118] 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 IEEE Int. Conf. on Soft. Eng. (ICSE)*, pages 295–307, 2003.

[119] 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.

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

[121] 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.

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

[123] 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.

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

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

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

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

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

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

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

[131] K. Pohl and T. Weyer. Documenting Variability in Requirements Artefacts. *Software Product Line Engineering*, chapter 5. Springer, 2005.

[132] 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.

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

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

[135] precise UML (pUML). http://www.cs.york.ac.uk/puml/maindetails.html. Research group created in 1987, with an overall objective to develop the UML as a precise, well-defined modeling language.

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

[137] 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.

[138] 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.

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

[140] W. N. Robinson. Monitoring web service requirements. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 65–74, 2003.

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

[142] 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.

[143] K. Ryan. The role of natural language in requirements engineering. In *Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 240–242, San Diego, CA, 1993. IEEE Computer Society Press, Los Alamitos, CA.

[144] 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.

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

[146] T. Savor and R. Seviora. An approach to automatic detection of software failures in realtime systems. In *Proc. IEEE Real-Time Tech. and Appl. Sym.*, pages 136–147, 1997.

[147] 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.

[148] 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.

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

[150] 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.

[151] M. Shaw. "Self-Healing": Softening precision to avoid brittleness. In *WOSS '02: Workshop on Self-Healing Systems*. November 2002. Position paper.

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

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

[154] B. G. Silverman, M. Johns, J. Cornwell, and K. O'Brien. Human behavior models for agents in simulators and games: part i: enabling science with pmfserv. *Presence: Teleoper. Virtual Environ.*, 15(2):139–162, 2006.

[155] 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.

[156] 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.

[157] 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.

[158] 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.

[159] 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.

[160] A. Sutcliffe and N. Maiden. The domain theory for requirements engineering. *IEEE Transactions on Software Engineering*, 24(3):174–196, 1998.

[161] 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.

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

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

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

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

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

[167] 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.

[168] 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.

[169] 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.

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

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

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

[173] 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.

[174] 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.

[175] A. Wassyng, M. Lawford, and X. Hu. Timing tolerances in safety-critical software. In *Proc. of the Int. Sym. of Form. Meth. Eur.*, pages 157–172, 2005.

[176] 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.

[177] 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.

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

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

[180] E. Yu. Agent orientation as a modelling paradigm. *Wirtschaftsinformatik*, 43(3):123–132, April 2001.

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

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

[183] 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.

[184] Z. Zhou, J. Zhang, P. K. McKinley, and B. H. C. Cheng. TA-LTL: Specifying adaptation timing properties in autonomic systems. In *3rd IEEE Workshop on Engineering of Autonomic Systems (EASe 2006)*, Columbia, Maryland, April 2006.