CLASSIFICATION OF RESEARCH EFFORTS IN REQUIREMENTS ENGINEERING

Pamela Zave

AT&T Laboratories—Research
700 Mountain Avenue, Room 2B-413
Murray Hill, New Jersey 07974, USA
+1 908 582 3080
pamela@research.att.com

21 February 1997

CLASSIFICATION OF RESEARCH EFFORTS IN REQUIREMENTS ENGINEERING

I. PURPOSE OF THE CLASSIFICATION SCHEME

Requirements engineering is the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behavior, and to their evolution over time and across software families.

The subject of requirements engineering is inherently broad, interdisciplinary, and open-ended. It concerns translation from informal observations of the real world to mathematical specification languages. For these reasons, it can seem chaotic in comparison to other areas in which computer scientists do research.

This paper presents a classification scheme for research efforts in requirements engineering. For those readers who are not familiar with requirements engineering, it is intended to provide an overview and a coherent framework for further study. For those readers who do research in requirements engineering, it is offered in the hope that it will:

- delineate the area and encourage research coverage of the whole area;
- provide structure to encourage the discovery and articulation of new principles;
- assist in grouping similar things, such as competing solutions to the same problem (these groupings would be a great help in comparing, extending, and exploiting results).

The great difficulty in constructing such a classification scheme is the heterogeneity of the topics usually considered part of requirements engineering. They include:

Tasks that must be completed: elicitation of information from clients, validation, specification.

Problems that must be solved: barriers to communication, incompleteness, inconsistency.

Solutions to problems: formal languages and analysis algorithms, prototyping, metrics, traceability.

Ways of contributing to knowledge: descriptions of current practice, case studies, controlled experiments.

Types of system: embedded systems, safety-critical systems, distributed systems.

A typical list of research topics in requirements engineering contains all these entries and more. It is intended to be comprehensive, but it is also confusing.

The obvious way out of this difficulty is a classification scheme with several orthogonal dimensions. The more dimensions the more precision, at the expense of making the scheme too complex to use. I have compromised by settling on two dimensions, which are presented separately in the next two sections.

I have referenced a number of papers that illustrate the categories and issues discussed. A reference is nothing more than an example; it is certainly not a claim that the referenced paper is the best or only work in its category! In addition, Section IV presents some examples that do not fit neatly into the nominal categories, and shows how the classification scheme sheds light on them as well.

II. FIRST DIMENSION: PROBLEMS OF REQUIREMENTS ENGINEERING

The first dimension is very particular to requirements engineering. It is an attempt to characterize the work that needs to be done. It must somehow cover necessary tasks, recognizable problems, and proposed solutions, without confusing the three.

Basing this primary dimension on solutions to problems seems like a bad idea, because it would discourage developing alternative solutions to problems, or comparing different solutions to the same problem.

Tasks and problems are both plausible starting points, and indeed overlap quite a bit. A task can always be described as a problem ("How can this task be accomplished satisfactorily?") and a problem can always be described as a task ("Find a solution to this problem."). I prefer to use problems because they are more stable than tasks. After all, the best solutions to problems make certain tasks unnecessary! In other words, a method is a proposed solution to a problem, and a method dictates which tasks are performed.

Here is the first dimension of the classification scheme. Explanatory notes are interspersed.

1. Problems of investigating the goals, functions, and constraints of a software system

This topic includes all the problems of gathering information, analyzing information, and generating alternative strategies. The longer requirements engineers work on solving these problems, the bigger the scope of their work, because their stock of information and alternatives is always increasing.

1.1. Overcoming barriers to communication

Requirements engineers have to talk to a wide range of people, with diverse backgrounds, interests, and personal goals. How can they communicate well with people whose backgrounds, interests, and goals are different from their own? And who may not know what they want from a computer system? Ethnographic techniques are sometimes proposed as a solution to this problem [Goguen & Linde 92, Sommerville et al. 92].

1.2. Generating strategies for converting vague goals (e.g., "user-friendliness," "security," "accuracy," "reliability") into specific properties or behavior

For example, prototyping is often proposed for exploring the friendliness of a user interface. There are also product-oriented [Harrison & Barnard 92] and process-oriented [Chung & Nixon 95] approaches to this problem (see Section III for a definition of these terms).

1.3. Understanding priorities and ranges of satisfaction

Many requirements are not absolute; they can be satisfied partially, or only if resources permit. Requirements engineers must obtain the information necessary to decide when and how to satisfy these requirements [Yen & Tiao 97].

1.4. Generating strategies for allocating requirements among the system and the various agents of its environment

The true requirements always refer to the real world in which the computer system will become embedded. Before the software can be specified, goals, functions, and constraints must be allocated to the various components and agents that will contribute to satisfying them [Alford 77, Dardenne et al. 93, Feather 87, Johnson 88].

1.5. Estimating costs, risks, and schedules

This is the other half of the information needed to handle optional requirements, which are generally satisfied depending on development resources. Requirements engineers must estimate the resources needed, and be aware of how reliable their estimates are [Matson et al. 94, Mukhopadhyay & Kekre 92].

1.6. Ensuring completeness

How can requirements engineers be sure that they haven't left out any important people, viewpoints, issues, facts, etc. out of their investigations? This is "completeness" in an informal sense [Reubenstein & Waters 91].

2. Problems of specifying software system behavior

This topic includes all the problems of synthesizing information and choosing among alternatives, to create a precise and minimal software specification. The longer requirements engineers work on solving these problems, the smaller the scope of their work, because they are discarding alternatives and irrelevant information.

2.1. Integrating multiple views and representations

The results of investigation are likely to be diverse and to contain conflicts. Understanding, communication, and negotiation are useful for reconciling conflicting viewpoints [Easterbrook 92]. Formal methods are useful for composing diverse notations and for monitoring inconsistencies [Nuseibeh et al. 94, Zave & Jackson 93].

2.2. Evaluating alternative strategies for satisfying requirements

Work on 1.2, 1.3, and 1.4 may generate alternatives, from which the specific system behavior must be chosen. Many of the papers cited in those sections also include evaluation strategies.

2.3. Obtaining complete, consistent, and unambiguous specifications

This is "completeness" in the formal sense of having no missing parts [Heimdahl & Leveson 96, Heitmeyer et al. 96].

2.4. Checking that the specified system will satisfy the requirements

There are a variety of approaches to this well-known problem. They include inspections [Porter et al. 95], execution and testing of the specification [Zave & Schell 86], and verification [Coen-Porisini et al. 94, Du Bois et al. 97].

2.5. Obtaining specifications that are well-suited for design and implementation activities

This is the problem of building into the specification qualities that will ensure successful software development. Sometimes designs [Lor & Berry 91] or test cases [Weyuker et al. 94] can be generated automatically or semi-automatically from a specification. Designs can also be checked for consistency with the specification [Lefering 92].

3. Problems of managing evolution of systems and families of systems

The first two major topics treat requirements engineering as if it were an isolated and unique phase of development. Of course, that is untrue. As systems evolve, they undergo many phases of requirements engineering. The requirements engineering of each member of a family should not be independent of other family members. This topic is concerned with the coordination of distinct requirements-engineering phases. It is concerned with how to make the work done in a phase reusable, and how to reuse it in other phases.

3.1. Reusing requirements engineering during evolutionary phases

In other words, this is the problem of ensuring that the artifacts of requirements engineering are maintainable. Some proposed solutions to this problem are traceability (recording the relationship between aspects of system behavior and the requirements that motivated them) [Leite & Oliveira 95, Ramesh et al. 95] and specification modularity.

3.2. Reusing requirements engineering for developing similar systems

In other words, this is the problem of ensuring that the artifacts of requirements engineering apply to families of systems. One example of a solution to this problem is conceptual modeling of an entire application domain [Lam et al. 97, Maiden & Sutcliffe 92, Ryan & Mathews 92]. Another example is separation of user-interface concerns from other concerns, so that the same "look and feel" can be provided across a product line.

3.3. Reconstructing requirements

This problem occurs when you want to reuse the artifacts of requirements engineering, but they are missing. It calls for reverse engineering of requirements. Very little work has been done on this problem.

III. SECOND DIMENSION: CONTRIBUTIONS TO SOLUTIONS IN REQUIREMENTS ENGINEERING

The second dimension could also apply to other areas of software engineering. It is an attempt to characterize the ways that research can contribute to solving problems. This dimension assumes that, as software engineers, we can seek to understand social factors but we can only hope to influence technical practices.

A. Report on the state of the practice

This establishes a baseline from which others can work [Lubars et al. 92].

B. Proposed process-oriented solution

Some problems must be solved manually, because we do not know how to solve them automatically. We can contribute to solving these problems by providing orderly methods and heuristics for making the decisions involved [Goguen & Linde 92, Jackson 83]. These contributions are "process-oriented solutions," because they focus on the manual process of requirements engineering.

C. Proposed product-oriented solution

Some problems can be solved automatically, in which case the emphasis is on formal representations and algorithmic manipulations of them. These contributions are "product-oriented solutions," because they focus on representation and manipulation of the products of requirements engineering [Heimdahl & Leveson 96, Lefering 92, Reubenstein & Waters 91].

Research on prototyping user interfaces would be classified 1.2, because it is addressing the problem of how to make a system user-friendly. As an example of the difference between contributions B and C, if the research emphasizes representation and automated implementation of interface choices and policies, then it would be a contribution of type C. If the research emphasizes working with users to determine their preferences, then it would be a contribution of type B.

As another example of the difference between B and C, of the two cited solutions to problem 1.1, one [Goguen & Linde 92] is a contribution of type B, and the other [Sommerville et al. 92] is a contribution of type C.

D. Case study applying a proposed solution to a substantial example

A case study provides important evidence, but it is necessarily anecdotal [van Lamsweerde et al. 95]. Ideally it would be done in preparation for a more systematic and objective evaluation of the proposed solution, as in E.

E. Evaluation or comparison of proposed solutions

To belong in this category, evaluation of a single proposed solution should be objective in some way ("I tried it and I liked it" is not enough) [Maiden & Sutcliffe 92, Zave 91]. Naturally, a comparison of several solutions is more likely to be systematic and objective. A controlled experiment with quantitative results is the ideal contribution in this category [Porter et al. 95].

F. Proposed measurement-oriented solution

It is now widely accepted that an organization can improve its problem-solving simply by monitoring and measuring how well it solves problems, and then tracking those measurements over time. Thus measurement of the success of requirements-engineering activities can be viewed as a problem-solving technique in its own right, as well as a means of comparing other solutions. For example, measurements of previous development projects help solve problem 1.5. Measurements of customer satisfaction help solve problems 1.1, 1.2, 1.3, and 2.2. A readability metric might help solve problem 2.4. Measurement can help solve 2.2 by checking the domain assumptions that were and are used to make strategic choices [Fickas & Feather 95].

IV. OTHER EXAMPLES

When a paper spans many categories in one dimension, it is usually narrowly focused in another dimension.

Sometimes the other dimension is also in this classification scheme. For example, [Reubenstein & Waters 91] and [Porter et al. 95] both make contributions of a very specific kind. But their contributions—an intelligent automated assistant and rigorous evaluation of inspection techniques, respectively—address many requirements problems simultaneously, in a wide-spectrum fashion.

Sometimes the focused dimension is not in the classification scheme. I have deliberately neglected problem solutions, so a paper focused on a solution technique might address several problems. For example, automated translation of natural-language specifications into formal specifications [Ishihara et al. 92] is a solution that might alleviate problems 1.1, 1.6, 2.1, 2.3, or 2.4. As such, it can be compared for effectiveness to drastically different solutions to these problems, such as ethnography and executable specifications.

Another neglected dimension is that of application domain. For example, the A-7 method [Heninger 80, Parnas & Clements 86, Parnas & Madey 95, van Schouwen et al. 92] is a comprehensive requirements method for real-time process-control systems. It attempts to solve (or at least alleviate) almost all requirements problems within the limits of that application domain.

ACKNOWLEDGMENT

This classification scheme was developed and refined while I was program chair of the Second IEEE International Symposium on Requirements Engineering. I would like to thank everyone who participated.

REFERENCES

[Alford 77]
Mack W. Alford. A requirements engineering methodology for real-time processing requirements. *IEEE Transactions on Software Engineering* III(1):60-69, January 1977.

[Chung & Nixon 95]
Lawrence Chung and Brian A. Nixon. Dealing with non-functional requirements: Three experimental studies of a process-oriented approach. In *Proceedings of the Seventeenth International Conference on Software Engineering*, pages 25-37. ACM Press, ISBN 0-89791-708-1, 1995.

[Coen-Porisini et al. 93]
Alberto Coen-Porisini, Richard A. Kemmerer, and Dino Mandrioli. A formal framework for ASTRAL intralevel proof obligations. *IEEE Transactions on Software Engineering* XX(8):548-561, August 1994.

[Dardenne et al. 93]
Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. *Science of Computer Programming* XX:3-50, 1993.

[Du Bois et al. 97]
Philippe Du Bois, Eric Dubois, and Jean-Marc Zeippen. On the use of a formal RE language: The generalized railroad crossing problem. In *Proceedings of the Third IEEE International Symposium on Requirements Engineering*, pages 128-137. IEEE Computer Society, ISBN 0-8186-7740-6, 1997.

[Easterbrook 92]
Steve Easterbrook. Domain modelling with hierarchies of alternative viewpoints. In *Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 65-72. IEEE Computer Society, ISBN 0-8186-3120-1, 1992.

[Feather 87]
Martin S. Feather. Language support for the specification and development of composite systems. *ACM Transactions on Programming Languages and Systems* IX(2):198-234, April 1987.

[Fickas & Feather 95]
Stephen Fickas and Martin S. Feather. Requirements monitoring in dynamic environments. In *Proceedings of the Second IEEE International Symposium on Requirements Engineering*, pages 140-147. IEEE Computer Society, ISBN 0-8186-7017-7, 1995.

[Goguen & Linde 92]
Joseph A. Goguen and Charlotte Linde. Techniques for requirements elicitation. In *Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 152-164. IEEE Computer Society, ISBN 0-8186-3120-1, 1992.

[Harrison & Barnard 92]
Michael Harrison and Phil Barnard. On defining requirements for interaction. In *Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 50-54. IEEE Computer Society, ISBN 0-8186-3120-1, 1992.

[Heimdahl & Leveson 96]
Mats P. E. Heimdahl and Nancy G. Leveson. Completeness and consistency in hierarchical state-based requirements. *IEEE Transactions on Software Engineering* XXII(6):363-377, June 1996.

[Heitmeyer et al. 96]
Constance L. Heitmeyer, Ralph D. Jeffords, and Bruce G. Labaw. Automated consistency checking of requirements specifications. *ACM Transactions on Software Engineering and Methodology* V(3):231-261, July 1996.

[Heninger 80]
Kathryn L. Heninger. Specifying software requirements for complex systems: New techniques and their application. *IEEE Transactions on Software Engineering* VI(1):2-13, January 1980.

[Ishihara et al. 92]
Yasunori Ishihara, Hiroyuki Seki, and Tadao Kasami. A translation method from natural language specifications into formal specifications using contextual dependencies. In *Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 232-239. IEEE Computer Society, ISBN 0-

8186-3120-1, 1992.

[Jackson 83]
Michael Jackson. *System development*. Prentice-Hall International, 1983.

[Johnson 88]
W. Lewis Johnson. Deriving specifications from requirements. In *Proceedings of the Tenth International Conference on Software Engineering*, pages 428-438. IEEE Computer Society, ISBN 0-8186-0849-8, 1988.

[Lam et al. 97]
W. Lam, J. A. McDermid, and A. J. Vickers. Ten steps towards systematic requirements reuse. In *Proceedings of the Third IEEE International Symposium on Requirements Engineering*, pages 6-15. IEEE Computer Society, ISBN 0-8186-7740-6, 1997.

[Leite & Oliveira 95]
Julio Cesar Sampaio do Prado Leite and Antonio de Padua Albuquerque Oliveira. A client oriented requirements baseline. In *Proceedings of the Second IEEE International Symposium on Requirements Engineering*, pages 108-115. IEEE Computer Society, ISBN 0-8186-7017-7, 1995.

[Lefering 92]
Martin Lefering. An incremental integration tool between requirements engineering and programming in the large. In *Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 82-89. IEEE Computer Society, ISBN 0-8186-3120-1, 1992.

[Lor & Berry 91]
Kar-Wing Edward Lor and Daniel M. Berry. Automatic synthesis of SARA design models from system requirements. *IEEE Transactions on Software Engineering* XVII(12):1229-1240, December 1991.

[Lubars et al. 92]
Mitch Lubars, Colin Potts, and Charles Richter. A review of the state of the practice in requirements modeling. In *Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 2-14. IEEE Computer Society, ISBN 0-8186-3120-1, 1992.

[Maiden & Sutcliffe 92]
N. A. M. Maiden and A. G. Sutcliffe. Requirements engineering by example: An empirical study. In *Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 104-111. IEEE Computer Society, ISBN 0-8186-3120-1, 1992.

[Matson et al. 94]
Jack E. Matson, Bruce E. Barrett, and Joseph M. Mellichamp. Software development cost estimation using function points. *IEEE Transactions on Software Engineering* XX(4):275-287, April 1994.

[Mukhopadhyay & Kekre 92]
Tridas Mukhopadhyay and Sunder Kekre. Software effort models for early estimation of process control applications. *IEEE Transactions on Software Engineering* XVIII(10):915-924, October 1992.

[Nuseibeh et al. 94]
Bashar Nuseibeh, Jeff Kramer, and Anthony Finkelstein. A framework for expressing the relationships between multiple views in requirements specification. *IEEE Transactions on Software Engineering* XX(10):760-773, October 1994.

[Parnas & Clements 86]
David Lorge Parnas and Paul C. Clements. A rational design process: How and why to fake it. *IEEE Transactions on Software Engineering* XII(2):251-257, February 1986.

[Parnas & Madey 95]
David Lorge Parnas and Jan Madey. Functional documentation for computer systems engineering. *Science of Computer Programming* XXV:41-61, October 1995.

[Porter et al. 95]
Adam A. Porter, Lawrence G. Votta, Jr., and Victor R. Basili. Comparing detection methods for software requirements inspections: A replicated experiment. *IEEE Transactions on Software Engineering* XXI(6):563-575, June 1995.

[Ramesh et al. 95]
B. Ramesh, T. Powers, C. Stubbs, and M. Edwards. Implementing requirements traceability: A case study. In *Proceedings of the Second IEEE International Symposium on Requirements Engineering*, pages 89-95. IEEE Computer Society, ISBN 0-8186-7017-7, 1995.

[Reubenstein & Waters 91]
Howard B. Reubenstein and Richard C. Waters. The requirements apprentice: Automated assistance for requirements acquisition. *IEEE Transactions on Software Engineering* XVII(3):226-240, March 1991.

[Ryan & Mathews 92]
Kevin Ryan and Brian Mathews. Matching conceptual graphs as an aid to requirements re-use. In *Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 112-120. IEEE Computer Society, ISBN 0-8186-3120-1, 1992.

[Sommerville et al. 92]
Ian Sommerville, Tom Rodden, Pete Sawyer, Richard Bentley, and Michael Twidale. Integrating ethnography into the requirements engineering process. In *Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 165-173. IEEE Computer Society, ISBN 0-8186-3120-1, 1992.

[van Schouwen et al. 92]
A. John van Schouwen, David Lorge Parnas, and Jan Madey. Documentation of requirements for computer systems. In *Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 198-207. IEEE Computer Society, ISBN 0-8186-3120-1, 1992.

[van Lamsweerde et al. 95]
Axel van Lamsweerde, Robert Darimont, and Philippe Massonet. Goal-directed elaboration of requirements for a meeting scheduler: Problems and lessons learnt. In *Proceedings of the Second IEEE International Symposium on Requirements Engineering*, pages 194-203. IEEE Computer Society, ISBN 0-8186-7017-7, 1995.

[Weyuker et al. 94]
Elaine Weyuker, Tarak Goradia, and Ashutosh Singh. Automatically generating test data from a boolean specification. *IEEE Transactions on Software Engineering* XX(5):353-363, May 1994.

[Yen & Tiao 97]
John Yen and W. Amos Tiao. A systematic tradeoff analysis for conflicting imprecise requirements. In *Proceedings of the Third IEEE International Symposium on Requirements Engineering*, pages 87-96. IEEE Computer Society, ISBN 0-8186-7740-6, 1997.

[Zave 91]
Pamela Zave. An insider's evaluation of PAISLey. *IEEE Transactions on Software Engineering* XVII(3):212-225, March 1991.

[Zave & Jackson 93]
Pamela Zave and Michael Jackson. Conjunction as composition. *ACM Transactions on Software Engineering*

and Methodology II(4):379-411, October 1993.

[Zave & Schell 86]
Salient features of an executable specification language and its environment. *IEEE Transactions on Software Engineering* XII(2):312-325, February 1986.