Classification of Research Efforts in Requirements Engineering

PAMELA ZAVE

AT&T Laboratories—Research

1. 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 article 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; and
—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 the following.

—*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; and
—*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 4 presents some examples that do not fit neatly into the nominal categories, and shows how the classification scheme sheds light on them as well.

2. FIRST DIMENSION: PROBLEMS

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 and Linde 1992; Sommerville et al. 1992].

(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 and Barnard 1992] and process-oriented [Chung and Nixon 1995] approaches to this problem (see Section 3 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 and Tiao 1997].

(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 1977; Dardenne et al. 1993; Feather 1987; Johnson 1988].

(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 the reliability of their estimates [Matson et al. 1994; Mukhopadhyay and Kekre 1992].

(1.6) *Ensuring completeness.* How can requirements engineers be sure that they have not left out any important people, viewpoints, issues, facts, etc., out of their investigations? This is "completeness" in an informal sense [Reubenstein and Waters 1991].

(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 1992]. Formal methods are useful for composing diverse notations and for monitoring inconsistencies [Nuseibeh et al. 1994; Zave and Jackson 1993].

(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 and Leveson 1996; Heitmeyer et al. 1996].

(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. 1995], execution and testing of the specification [Zave and Schell 1986], and verification [Coen-Porisini et al. 1994; Du Bois et al. 1997].

(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 and Berry 1991] or test cases [Weyuker et al. 1994] can be generated automatically or semiautomatically from a specification. Designs can also be checked for consistency with the specification [Lefering 1992].

(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 and Oliveira 1995; Ramesh et al. 1995] 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 require-

ments 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. 1997; Maiden and Sutcliffe 1992; Ryan and Mathews 1992]. 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.

3. SECOND DIMENSION: CONTRIBUTIONS TO SOLUTIONS

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. 1992].

(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 and Linde 1992; Jackson 1983]. 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 and Leveson 1996; Lefering 1992; Reubenstein and Waters 1991].

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 and Linde 1992] is a contribution of type B, and the other [Sommerville et al. 1992] 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. 1995]. 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 and Sutcliffe 1992; Zave 1991]. 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. 1995].

(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 and Feather 1995].

4. OTHER EXAMPLES

When an article 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 and Waters [1991] and Porter et al. [1995] 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 and in a wide-spectrum fashion.

Sometimes the focused dimension is not in the classification scheme. I have deliberately neglected problem solutions, so an article focused on a solution technique might address several problems. For example, automated translation of natural-language specifications into formal specifications [Ishihara et al. 1992] 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 1980; Parnas and Clements 1986; Parnas and Madey 1995; van Schouwen et al. 1992] 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, M. W. 1977. A requirements engineering methodology for real-time processing requirements. *IEEE Trans. Softw. Eng. III,* 1 (Jan.) 60–69.

CHUNG, L. AND NIXON, B. A. 1995. Dealing with non-functional requirements: Three experimental studies of a process-oriented approach. In *Proceedings of the Seventeenth International Conference on Software Engineering,* ACM Press, ACM, New York, NY, 25–37.

COEN-PORISINI, A., KEMMERER, R. A., AND MANDRIOLI, D. 1994. A formal framework for ASTRAL intralevel proof obligations. *IEEE Trans. Softw. Eng. XX,* 8 (Aug.) 548–561.

DARDENNE, A., VAN LAMSWEERDE, A., AND FICKAS, S. 1993. Goal-directed requirements acquisition. *Science of Computer Programming XX,* 3–50.

DU BOIS, P., DUBOIS, E., AND ZEIPPEN, J.-M. 1997. On the use of a formal RE language: The generalized railroad crossing problem. In *Proceedings of the Third IEEE International Symposium on Requirements Engineering,* IEEE Computer Society, ISBN 0-8186-7740-6, 128–137.

EASTERBROOK, S. 1992. Domain modelling with hierarchies of alternative viewpoints. In *Proceedings of the IEEE International Symposium on Requirements Engineering,* IEEE Computer Society, ISBN 0-8186-3120-1, 65–72.

FEATHER, M. S. 1987. Language support for the specification and development of composite systems. *ACM Trans. Program. Lang. Syst. IX,* 2 (Apr.), 198–234.

FICKAS, S. AND FEATHER, M. S. 1995. Requirements monitoring in dynamic environments. In *Proceedings of the Second IEEE International Symposium on Requirements Engineering,* IEEE Computer Society, ISBN 0-8186-7017-7, 140–147.

GOGUEN, J. A. AND LINDE, C. 1992. Techniques for requirements elicitation. In *Proceedings of the IEEE International Symposium on Requirements Engineering,* IEEE Computer Society, ISBN 0-8186-3120-1, 152–164.

HARRISON, M. AND BARNARD, P. 1992. On defining requirements for interaction. In *Proceedings of the IEEE International Symposium on Requirements Engineering,* IEEE Computer Society, ISBN 0-8186-3120-1, 50–54.

HEIMDAHL, M. P. E. AND LEVESON, N. G. 1996. Completeness and consistency in hierarchical state-based requirements. *IEEE Trans. Softw. Eng. XXII,* 6 (June), 363–377.

HEITMEYER, C. L., JEFFORDS, R. D., AND LABAW, B. G. 1996. Automated consistency checking of requirements specifications. *ACM Trans. Softw. Eng. Method. V,* 3 (July) 231–261.

HENINGER, K. L. 1980. Specifying software requirements for complex systems: New techniques and their application. *IEEE Trans. Softw. Eng. VI,* 1 (Jan.) 2–13.

ISHIHARA, Y., SEKI, H., AND KASAMI, T. 1992. A translation method from natural language specifications into formal specifications using contextual dependencies. In *Proceedings of the IEEE International Symposium on Requirements Engineering,* IEEE Computer Society, ISBN 0-8186-3120-1, 232–239.

JACKSON, M. 1983. *System Development*. Prentice-Hall International.

JOHNSON, W. L. 1988. Deriving specifications from requirements. In *Proceedings of the Tenth International Conference on Software Engineering,* IEEE Computer Society, ISBN 0-8186-0849-8, 428–438.

LAM, W., MCDERMID, J. A., AND VICKERS, A. J. 1997. Ten steps towards systematic requirements reuse. In *Proceedings of the Third IEEE International Symposium on Requirements Engineering,* IEEE Computer Society, ISBN 0-8186-7740-6, 6–15.

DO PRADO LEITE, J. C. S. AND DE PADUA ALBUQUERQUE OLIVEIRA, A. 1995. A client oriented requirements baseline. In *Proceedings of the Second IEEE International Symposium on Requirements Engineering,* IEEE Computer Society, ISBN 0-8186-7017-7, 108–115.

LEFERING, M. 1992. An incremental integration tool between requirements engineering and programming in the large. In *Proceedings of the IEEE International Symposium on Requirements Engineering,* IEEE Computer Society, ISBN 0-8186-3120-1, 82–89.

LOR, K.-W. E. AND BERRY, D. M. 1991. Automatic synthesis of SARA design models from system requirements. *IEEE Trans. Softw. Eng. XVII,* 12 (Dec.) 1229–1240.

LUBARS, M., POTTS, C., AND RICHTER, C. 1992. A review of the state of the practice in requirements modeling. In *Proceedings of the IEEE International Symposium on Requirements Engineering,* IEEE Computer Society, ISBN 0-8186-3120-1, 2–14.

MAIDEN, N. A. M. AND SUTCLIFFE, A. G. 1992. Requirements engineering by example: An empirical study. In *Proceedings of the IEEE International Symposium on Requirements Engineering,* IEEE Computer Society, ISBN 0-8186-3120-1, 104–111.

MATSON, J. E., BARRETT, B. E., AND MELLICHAMP, J. M. 1994. Software development cost estimation using function points. *IEEE Trans. Softw. Eng. XX,* 4 (Apr.) 275–287.

MUKHOPADHYAY, T. AND KEKRE, S. 1992. Software effort models for early estimation of process control applications. *IEEE Trans. Softw. Eng. XVIII,* 10 (Oct.) 915–924.

NUSEIBEH, B., KRAMER, J., AND FINKELSTEIN, A. 1994. A framework for expressing the relationships between multiple views in requirements specification. *IEEE Trans. Softw. Eng. XX,* 10 (Oct.) 760–773.

PARNAS, D. L. AND CLEMENTS, P. C. 1986. A rational design process: How and why to fake it. *IEEE Trans. Softw. Eng. XII,* 2 (Feb.), 251–257.

PARNAS, D. L. AND MADEY, J. 1995. Functional documentation for computer systems engineering. *Science of Computer Programming XXV* (Oct.), 41–61.

PORTER, A. A., VOTTA, JR., L. G., AND BASILI, V. R. 1995. Comparing detection methods for software requirements inspections: A replicated experiment. *IEEE Trans. Softw. Eng. XXI,* 6 (June) 563–575.

RAMESH, B., POWERS, T., STUBBS, C., AND EDWARDS, M. 1995. Implementing requirements traceability: A case study. In *Proceedings of the Second IEEE International Symposium on Requirements Engineering,* IEEE Computer Society, ISBN 0-8186-7017-7, 89–95.

REUBENSTEIN, H. B. AND WATERS, R. C. 1991. The requirements apprentice: Automated assistance for requirements acquisition. *IEEE Trans. Softw. Eng. XVII,* 3 (Mar.) 226–240.

RYAN, K. AND MATHEWS, B. 1992. Matching conceptual graphs as an aid to requirements reuse. In *Proceedings of the IEEE International Symposium on Requirements Engineering,* IEEE Computer Society, ISBN 0-8186-3120-1, 112–120.

SOMMERVILLE, I., RODDEN, T., SAWYER, P., BENTLEY, R., AND TWIDALE, M. 1992. Integrating ethnography into the requirements engineering process. In *Proceedings of the IEEE International Symposium on Requirements Engineering,* IEEE Computer Society, ISBN 0-8186-3120-1, 165–173.

VAN SCHOUWEN, A. J., PARNAS, D. L., AND MADEY, J. 1992. Documentation of requirements for computer systems. In *Proceedings of the IEEE International Symposium on Requirements Engineering,* IEEE Computer Society, ISBN 0-8186-3120-1, 198–207.

VAN LAMSWEERDE, A., DARIMONT, R., AND MASSONET, P. 1995. Goal-directed elaboration of requirements for a meeting scheduler: Problems and lessons learnt. In *Proceedings of the Second IEEE International Symposium on Requirements Engineering,* IEEE Computer Society, ISBN 0-8186-7017-7, 194–203.

WEYUKER, E., GORADIA, T., AND SINGH, A. 1994. Automatically generating test data from a boolean specification. *IEEE Trans. Softw. Eng. XX,* 5 (May) 353–363.

YEN, J. AND TIAO, W. A. 1997. A systematic tradeoff analysis for conflicting imprecise requirements. In *Proceedings of the Third IEEE International Symposium on Requirements Engineering,* IEEE Computer Society, ISBN 0-8186-7740-6, 87–96.

ZAVE, P. 1991. An insider's evaluation of PAISLey. *IEEE Trans. Softw. Eng. XVII,* 3 (Mar.) 212–225.

ZAVE, P. AND JACKSON, M. 1993. Conjunction as composition. *ACM Trans. Softw. Eng. Method. II,* 4 (Oct.) 379–411.

ZAVE, P. AND SCHELL, W. M. 1986. Salient features of an executable specification language and its environment. *IEEE Trans. Softw. Eng. XII,* 2 (Feb.) 312–325.

Received December 1995; revised March 1997; accepted October 1997