

# CLASSIFICATION OF RESEARCH EFFORTS IN REQUIREMENTS ENGINEERING

Pamela Zave

AT&T Bell Laboratories

## I. Introduction

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.

Of all the areas in which computer scientists do research, requirements engineering is probably the most informal, interdisciplinary, and subjective. Although these qualities are inherent to the topic under investigation, they make scientists and mathematicians uncomfortable.

Given these circumstances, a rigorous classification of research efforts in requirements engineering—if comprehensive and intelligible—might have several benefits, including:

1. It would delineate the area and would encourage research coverage of the whole area.
2. It would provide structure that might encourage the discovery and articulation of new principles.
3. It would 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.

This article proposes and justifies a trial classification scheme. An earlier version was used to organize the papers submitted to this symposium, and the scheme has been refined somewhat in response to inadequacies discovered during the process of selecting the program. It is offered in hopes of stimulating discussion and eventual consensus.

The first issue to be tackled is the heterogeneity of the topics usually considered part of requirements engineering. They include . . . .

*Tasks that must be completed:* elicitation, 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 practice, case studies, controlled experiments.

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

A list with all these topics is intended to be comprehensive, but its heterogeneity undermines all chance of bringing order to the field.

There seems to be a need for several orthogonal dimensions of classification. While multiple dimensions will certainly help us cope with the heterogeneity of concerns, there is a danger of making the classification scheme too complex to use. I have compromised by settling on two dimensions, which are presented separately in the next two sections.

## II. The first dimension: Problems of requirements engineering

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

Basing this primary dimension on solutions to problems seems like a particularly 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! Another way of saying this is that a method is a proposed solution to a problem, and a

method dictates which tasks are performed.

Here is the first dimension of the proposed 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.

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 proposed as a solution to this problem.

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

For example, prototyping is often proposed for exploring user-friendliness.

1.3. *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.

1.4. *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.

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. Measuring previous development projects is a typical ingredient in solutions to this problem.

1.6. *Ensuring completeness*

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

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.

2.1. *Integrating multiple views and representations*

The results of investigation are likely to be diverse and to contain conflicts. Formal methods are probably needed to manage diverse notations. Negotiation is probably needed to reconcile conflicting viewpoints.

2.2. *Evaluating alternative strategies for satisfying requirements*

Work on 1.2 and 1.3 may generate many alternatives, from which the specific system behavior must be chosen.

2.3. *Choosing which optional requirements to satisfy*

Work on 1.4 may generate many possibilities, from which specific system behavior must be chosen.

2.4. *Obtaining complete, consistent, and unambiguous specifications*

This is "completeness" in the formal sense of having no missing parts.

2.5. *Checking that the specified system will satisfy the requirements*

This problem is usually known as "validation." A wide range of well-known techniques is applicable here. An example of a lesser-known technique is a readability metric, the purpose of which is to ensure that clients can read the specification. (If clients cannot understand the specification, they are unlikely to provide helpful comments on it.)

2.6. *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. A technique for generating test cases automatically from the specification would help solve this problem.

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 the 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 the artifacts of requirements engineering during evolutionary phases*

Traceability—recording the connection between the system behavior and the requirement it is satisfying—is a common solution here.

3.2. *Reusing the artifacts of requirements engineering for developing similar systems*

Conceptual modeling of an entire application domain is a commonly proposed solution to this problem.

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.

III. *The 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 practice*

This establishes a baseline from which others can work.

B. *Analysis of the cultural, political, organizational, and economic factors relevant to a problem*

This provides understanding of the nontechnical context in which technical work is performed.

C. *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. These contributions are “process-oriented solutions,” because they focus on the manual process of requirements engineering.

D. *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.

E. *Case study applying a proposed solution to a substantial example*

A case study provides important evidence, but it is necessarily anecdotal. Ideally it would be done in preparation for a more systematic and objective evaluation of the proposed solution, as in F.

F. *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). Naturally, a comparison of several solutions is more likely to be systematic and objective. A controlled experiment with quantitative results would be the ideal contribution in this category, but such methods are seldom applicable in requirements engineering.

G. *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.

IV. *Conclusion*

Using this scheme to classify research papers, I have observed that some papers fit exactly into one lowest-level category in each dimension, and others span many categories.

Some good research is broad, and some good research is narrow and deep. This difference in style accounts for most of the variation in the numbers of categories applicable to papers. There is also the unavoidable fact that every classification scheme has a bias, and this one performs worst on (fails to classify neatly) research that solves many problems simultaneously.

A more serious concern is that some categories (1.4, 1.5, 2.2, 2.3, 3.2, 3.3, B, F) describe relatively few research efforts. I do not believe that the problems enumerated under these numbers do not exist, or that the contributions listed under these letters are not valuable. The obvious conclusion is that these problems and contributions deserve more attention from researchers than they now receive.