Int J STTT (2002) 4: 1–7 / Digital Object Identifier (DOI) 10.1007/s10009-002-0083-4

The opinion corner

What makes good research in software engineering?

Mary Shaw

School of Computer Science, Carnegie Mellon University, Pittsburgh PA 15213, USA; E-mail: mary.shaw@cs.cmu.edu

Published online: 13 June 2002 – © Springer-Verlag 2002

Abstract. Physics, biology, and medicine have well-refined public explanations of their research processes. Even in simplified form, these provide guidance about what counts as "good research" both inside and outside the field. Software engineering has not yet explicitly identified and explained either our research processes or the ways we recognize excellent work. Science and engineering research fields can be characterized in terms of the kinds of questions they find worth investigating, the research methods they adopt, and the criteria by which they evaluate their results. I will present such a characterization for software engineering, showing the diversity of research strategies and the way they shift as ideas mature. Understanding these strategies should help software engineers design research plans and report the results clearly; it should also help explain the character of software engineering research to computer science at large and to other scientists.

1 Introduction

Many sciences have good explanations of their research strategies. These explanations include not only detailed guidance for researchers but also simplified views for the public and other observers. Acceptance of their results relies on the process of obtaining the results as well as analysis of the results themselves. Schoolchildren learn the experimental model of physics: hypothesis, controlled experiment, analysis, and possible refutation. The public understands large-scale double-blind medical studies well enough to discuss the risks of experimental treatment, the ethics of withholding promising treatment from the control group, and the conflicts of interest that are addressed by the blinding process.

Software engineering does not have this sort of well-understood guidance. Software engineering researchers rarely write explicitly about their paradigms of research and their standards for judging quality of results. A number of attempts to characterize software engineering research have contributed elements of the answer, but they do not yet paint a comprehensive picture. In 1980, I [7] examined the relation of engineering disciplines to their underlying craft and technology and laid out expectations for an engineering discipline for software. In 1984–1985, Redwine, Riddle, and others [5, 6] proposed a model for the way software engineering technology evolves from research ideas to widespread practice. More recently, software engineering researchers have criticized common practice in the field for failing to collect, analyze, and report experimental measurements in research reports [9–12]. In 2001 I [8] presented preliminary sketches of some of the successful paradigms for software engineering research, drawing heavily on examples from software architecture.

Scientific and engineering research fields can be characterized by identifying what they value:

- What kinds of questions are "interesting"?
- What kinds of results help to answer these questions, and what research methods can produce these results?
- What kinds of evidence can demonstrate the validity of a result, and how are good results distinguished from bad ones?

In this paper I attempt to make generally accepted research strategies in software engineering explicit by examining research in the area to identify what is widely accepted in practice.

This paper was presented as an invited lecture at ETAPS 2002, the Joint European Conferences on Theory and Practice of Software, in April 2002 in Grenoble.

1.1 Software technology maturation

Redwine and Riddle [5, 6] reviewed a number of software technologies to see how they develop and propagate. They found that it typically takes 15–20 years for a technology to evolve from concept formulation to the point where it's ready for popularization. They identify six typical phases:

- *Basic research.* Investigate basic ideas and concepts, put initial structure on the problem, frame critical research questions.
- *Concept formulation.* Circulate ideas informally, develop a research community, converge on a compatible set of ideas, publish solutions to specific subproblems.
- *Development and extension.* Make preliminary use of the technology, clarify underlying ideas, generalize the approach.
- *Internal enhancement and exploration.* Extend approach to another domain, use technology for real problems, stabilize technology, develop training materials, show value in results.
- *External enhancement and exploration.* Similar to internal, but involving a broader community of people who weren't developers, show substantial evidence of value and applicability.
- *Popularization.* Develop production-quality, supported versions of the technology, commercialize and market technology, expand user community.

Redwine and Riddle presented timelines for several software technologies as they progressed through these phases up until the mid-1980s. I presented a similar analysis for the maturation of software architecture in the 1990s [8].

Our interest here is in the first three phases, i.e., the research phases. Software engineering research is intended to help improve the practice of software development, so research planning should make provisions for the transition. The Redwine–Riddle data suggests that around 10 of the 15–20 years of evolution are spent in concept formation and in development and extension (still more time is spent in basic research, but it is very difficult to identify the beginning of this phase). As a result, full understanding of research strategy must account for the accumulation of evidence over time as well as for the form and content of individual projects and papers.

The IMPACT project [3] is tracing the path from research into practice. The objectives of the project include identifying the kinds of contributions that have substantial impact and the types of research that are successful. Preliminary results are now being discussed at conferences.

1.2 Prior reflections on software engineering and related research

Software engineering research includes, but is not limited to, experimental research. Further, it resembles in some respects research in human-computer interaction.

1.2.1 Critiques of experimental software engineering

In 1993, Basili laid out experimental research paradigms appropriate for software engineering [1]. Later, Tichy [9, 10] and colleagues criticized the lack of quantitative experimental validation reported in conference papers:

> "Computer scientists publish relatively few papers with experimentally validated results ... The low ratio of validated results appears to be a serious weakness in CS research. This weakness should be rectified." [9]

They classified 246 papers in computer science and, for comparison, 147 papers in two other disciplines, according to the type of contribution in the article. The majority of the papers (259 of 403) produced design and modeling results. They then assessed each paper's evaluation of its results on the basis of the fraction of the article's text devoted to evaluation. They found, for example, that hypothesis testing was rare in all samples, that a large fraction (43%) of computer science design and modeling papers lacked any experimental evaluation, and that software engineering samples were worse than computer science in general.

Zelkowitz and Wallace [11, 12] built on Basili's description of experimental paradigms and evaluated over 600 computer science papers and over 100 papers from other disciplines published over a 10-year period. Again, they found that too many papers had no experimental validation or only informal validation, though they did notice some progress over the 10-year period covered by their study.

These critiques start from the premise that software engineering research should follow a classical experimental paradigm. Here I explore a different question: what are the characteristics of the software engineering research that the field recognizes as being of high quality?

1.2.2 Analyzing research approaches with pro forma abstracts

Newman compared research in human-computer interaction (HCI) to research in engineering [4]. He characterized engineering practice, identified three main types of research contributions, and performed a preliminary survey of publications in five engineering fields. He found that over 90% of the contributions were of three kinds:

EM *Enhanced analytical modeling techniques*, based on relevant theory, that can be used to tell whether the design is practicable or to make performance predictions.

ES *Enhanced solutions* that overcome otherwise insoluble aspects of problems, or that are easier to analyze with existing modeling techniques.

ET *Enhanced tools and methods* for applying analytical models and for building functional models or prototypes [4].

Newman created pro forma abstracts – templates for stylized abstracts what would capture the essence of the papers – for each of these types of contributions. For example, the pro forma abstract for enhanced modeling techniques is

> "Existing <model-type> models are deficient in dealing with <properties> of <solution strategy>. An enhanced <model-type> is described, capable of providing more accurate analyses/predictions of <properties> in <solution strategy> designs. The model has been tested by comparing analyses/predictions with empirically measured values of <properties>." [4]

He found that in order to account for a comparable fraction of the HCI literature, he needed two more templates, for "radical solutions" and for "experience and/or heuristics".

Newman reported that in addition to helping to identify the kind of research reported in a paper, the pro forma abstracts also helped him focus his attention while reading the paper. It seems reasonable to assume that if authors were more consciously aware of typical paper types, they would find it easier to write papers that presented their results and supporting evidence clearly. The approach of characterizing papers through pro forma abstracts is also useful for software engineering, though a more expressive descriptive model as described below provides better matches with the papers.

1.2.3 Broad view of research

Brooks reflected on the tension in human-computer interaction research between:

- "narrow truths proved convincingly by statistically sound experiments" and
- "broad 'truths', generally applicable, but supported only by possibly unrepresentative observations" [2].

The former satisfy the gold standard of science, but are few and narrow compared to the decisions designers make daily. The latter provide pragmatic guidance, but at risk of over-generalization.

Brooks proposes to relieve the tension through a *certainty-shell* structure – to recognize three nested classes of results,

- *Findings:* well-established scientific truths, judged by truthfulness and rigor;
- *Observations:* reports on actual phenomena, judged by interestingness;
- *Rules of thumb:* generalizations, signed by their author but perhaps incompletely supported by data, judged by usefulness

with freshness as a criterion for all three.

This tension is as real in software engineering as in human-computer interaction. Observations and rules of thumb provide valuable guidance for practice when findings are not available. They also help to understand an area and lay the groundwork for the research that will, in time, yield findings.

2 Questions, results, and validation in software engineering

Generally speaking, software engineering researchers seek better ways to develop and evaluate software. They are motivated by practical problems, and key objectives of the research are often quality, cost, and timeliness of software products.

This section presents a model that explains software engineering research papers by classifying the types of research questions they ask, the types of results they produce, and the character of the validation they provide. This model has evolved over several years. It refines the version I presented at ICSE 2001 [8] based on discussion of the model in a graduate class and review of abstracts submitted to ICSE 2002. Its status is, in Brooks' sense, a set of observations, perhaps becoming a generalization.

2.1 Types of research questions

Research questions may be about methods for developing software, about methods for analyzing software, about the design, evaluation, or implementation of specific systems, about generalizations over whole classes of systems, or about the sheer feasibility of a task. Table 1 shows the types of research questions software engineers ask, together with some examples of specific typical questions.

Among ICSE submissions, the most common kind of paper reports an improved method or means of developing software. Papers about methods for analysis are also fairly common, principally analysis of correctness (testing and verification).

Looking back over the history of software engineering, there is some indication that the types of questions have changed as the field matures. For example, generalizations, especially in the form of more formal models, are becoming more common, and feasibility papers seem to be becoming less common as the field matures.

2.2 Types of research results

Research yields new knowledge. This knowledge is expressed in the form of a particular result. The result may be a specific procedure or technique for software development or for analysis. It may be more general, capturing a number of specific results in a model; such models are of many degrees of precision and formality. Sometimes, the result is the solution to a specific problem or the outcome of a specific analysis. Finally, as Brooks observed, observations and rules of thumb

Table 1. Research questions in software engineering

Type of question	Examples
Method or means of development	How can we do/create (or automate doing) X? What is a better way to do/create X?
Method for analysis	How can I evaluate the quality/correctness of X? How do I choose between X and Y?
Design, evaluation, or analysis of a particular instance	What is a (better) design or implementation for application X? What is property X of artifact/method Y? How does X compare to Y? What is the current state of X/practice of Y?
Generalization or characterization	Given X, what will Y (necessarily) be? What, exactly, do we mean by X? What are the important characteristics of X? What is a good formal/empirical model for X? What are the varieties of X, how are they related?
Feasibility	Does X even exist, and if so what is it like? Is it possible to accomplish X at all?

may be good preliminary results. Table 2 lists these types, together with some examples of specific typical questions.

By far, the most common kind of ICSE paper reports a new procedure or technique for software development or analysis. It may be described narratively, or it may be embodied in a tool. Analytic models and descriptive models are also common; analytic models support predictive analysis, whereas descriptive models explain the structure of a problem area or expose important design decisions. Empirical models backed up by good statistics are uncommon.

Table 2. Research results in software engineering

Type of result	Examples
Procedure or technique	New or better way to do some task, such as design, implementation, measurement, evaluation, selection from alternatives Includes operational techniques for implementation, representation, management, and analysis, but not advice or guidelines
Qualitative or descriptive model	Structure or taxonomy for a problem area; architectural style, framework, or design pattern; non-formal domain analysis Well-grounded checklists, well-argued informal generalizations, guidance for integrating other results,
Empirical model	Empirical predictive model based on observed data
Analytic model	Structural model precise enough to support formal analysis or automatic manipulation
Notation or tool	Formal language to support technique or model (should have a calculus, semantics, or other basis for computing or inference) Implemented tool that embodies a technique
Specific solution	Solution to application problem that shows use of software engineering principles – may be design, rather than implementation Careful analysis of a system or its development Running system that embodies a result; it may be the carrier of the result, or its implementation may illustrate a principle that can be applied elsewhere
Answer or judgment	Result of a specific analysis, evaluation, or comparison
Report	Interesting observations, rules of thumb

Table 3. Validation techniques in software engineering

Type of validation	Examples	
Analysis	I have analyzed my result and find it satisfactory through ...	
	(formal analysis)	. . . rigorous derivation and proof
	(empirical model)	. . . data on controlled use
	(controlled experiment)	. . . carefully designed statistical experiment
Experience	My result has been used on real examples by someone other than me, and the evidence of its correctness/usefulness/effectiveness is . . .	
	(qualitative model)	. . . narrative
	(empirical model,	. . . data, usually statistical, on practice
	(notation, tool) technique)	. . . comparison of this with similar results in actual use
Example	Here's an example of how it works on	
	(toy example)	. . . a toy example, perhaps motivated by reality
	(slice of life)	. . . a system that I have been developing
Evaluation	Given the stated criteria, my result...	
	(descriptive model)	. . . adequately describes the phenomena of interest . . .
	(qualitative model)	. . . accounts for the phenomena of interest . . .
	(empirical model)	. . . is able to predict . . . because . . . ,
	or	. . . gives results that fit real data . . .
	Includes feasibility studies, pilot projects	
Persuasion	I thought hard about this, and I believe . . .	
	(technique)	. . . if you do it the following way, . . .
	(system)	. . . a system constructed like this would . . .
	(model)	. . . this model seems reasonable
	Note that if the original question was about feasibility, a working system, even without analysis, can be persuasive	
Blatant assertion	No serious attempt to evaluate result	

2.3 Types of research validations

Good research requires not only a result, but also clear and convincing evidence that the result is sound. This evidence should be based on experience or systematic analysis, not simply persuasive argument or textbook examples. Table 3 shows some common types of validation, indicating that validation in practice is not always clear and convincing.

The most common kinds of validation among ICSE papers are experience in actual use and systematic analysis. A significant number of ICSE submissions depend only on blatant assertion to demonstrate the validity of their results, or offer no evidence at all; such submissions are only rarely accepted.

3 Research strategies

Section 2 identifies the three important aspects of an individual research result as reported in a typical conference or journal paper. It is clear that the spectrum of good research strategies includes experimental computer science in the sense of [9–12]; it is also clear that the spectrum is much broader than just experimental research. Of course, not all the combinations of question, result, and validation make sense.

Inspection of Tables 1–3 suggests combinations that make sense. Continuing to report on what strategies are accepted, rather than setting a prescriptive standard, in this section I observe and comment on some of the patterns that appear in the literature.

3.1 Creating research strategies

The most common research strategy in software engineering solves some aspect of a software development problem by producing a new procedure or technique and validating it by analysis or by discussing an example of its use; examples of use in actual practice are more compelling than examples of use on idealized problems.

Another common research strategy provides a way to analyze some aspect of software development by developing an analytic, often formal, model and validating it through formal analysis or experience with its use.

Table 4. Research strategies for ICSE 2002, based on submitted abstracts

Question	Result	Validation	Count
Development method	Procedure	Analysis	3
Development method	Procedure	Experience	4
Development method	Procedure	Example	7
Development method	Qualitative model	Experience	2
Development method	Qualitative model	Persuasion	1
Development method	Analytic model	Experience	3
Development method	Notation or tool	Analysis	1
Development method	Notation or tool	Experience	1
Development method	Notation or tool	Example	2
Analysis method	Procedure	Analysis	1
Analysis method	Procedure	Experience	3
Analysis method	Procedure	Example	2
Analysis method	Analytic model	Analysis	1
Analysis method	Analytic model	Experience	1
Analysis method	Analytic model	Example	2
Analysis method	Tool	Example	1
Evaluation of instance	Specific analysis	Analysis	3
Evaluation of instance	Specific analysis	Example	1
Evaluation of instance	Answer	Analysis	1

Table 4 shows the strategies for 40 research papers accepted for ICSE 2002, based on the submitted abstracts for those papers. Some papers are not included because the strategy was not clear from the abstract.

These descriptions are consistent with Newman's pro forma abstracts. Those templates identify sets of compatible questions, results, and validations. For example, the "enhanced model" quoted in Sect. 1.2.2 corresponds to a generalization or characterization question answered by an analytic or empirical (or precise descriptive) model, validated by empirical analysis or controlled experiment. By packaging several choices together and naming the set, Newman identifies the selected strategies clearly. However, attempts to apply them to the software engineering literature revealed shortcomings in coverage.

3.2 Building good results from good papers

This discussion has focused on individual results as reported in conference and journal papers. Major results, however, gain credibility over time as successive papers provide incremental improvement of the result and progressively stronger credibility. Assessing the significance of software engineering results should be done in this larger context.

As increments of progress appear, they offer assurance that continued investment in research will pay off. Thus initial reports in an area may be informal and qualitative but present a persuasive case for exploratory research, while later reports present empirical and later formal models that justify larger investment. This pattern of growth is consistent with the Redwine–Riddle model of technology maturation.

The model presented here does not address this cumulative confidence building.

4 Summary

Software engineering will benefit from a better understanding of the research strategies that have been most successful. The model presented here reflects the character of the discipline: it identifies the types of questions software engineers find interesting, the types of results we produce in answering those questions, and the types of evidence that we use to evaluate the results.

Research questions are of different kinds, and research strategies vary in response. The strategy of a research project should select a result, an approach to obtaining the result, and a validation strategy appropriate to the research question. More explicit awareness of these choices may help software engineers design research projects and report their results; it may also help readers read and evaluate the literature.

The questions of interest change as the field matures. One indication that ideas are maturing is a shift from qualitative and empirical understanding to precise and quantitative models.

This analysis has considered individual research reports, but major results that influence practice rely on accumulation of evidence from many projects. Each individual paper thus provides incremental knowledge, and collections of related research projects and reports provide both confirming and cumulative evidence.

Acknowledgements. The development of these ideas has benefited from discussion with colleagues at Carnegie Mellon, at open discussion sessions at the FSE Conference, and with the program committee of ICSE 2002. The work has been supported by the A.J. Perlis Chair at Carnegie Mellon.

References

1. Basili, VR: The experimental paradigm in software engineering. In: Dieter Rombach, H, Basili, VR, Selby, R (eds), Experimental software engineering issues: critical assessment and future directives. Proc Dagstuhl-Workshop, Lecture Notes in Computer Science, vol 706. Springer, Berlin Heidelberg New York, 1993
2. Brooks, FP, Jr: Grasping reality through illusion– interactive graphics serving science. Proc 1988 ACM SIGCHI Human Factors in Computer Systems Conference (CHI '88), pp 1–11
3. Impact Project: Determining the impact of software engineering research upon practice. Panel summary, Proc 23rd International Conference on Software Engineering (ICSE 2001), 2001
4. Newman, W: A preliminary analysis of the products of HCI research, using pro forma abstracts. Proc 1994 ACM SIGCHI Human Factors in Computer Systems Conference (CHI '94), pp 278–284
5. Redwine, S, et al.: DoD Related software technology requirements, practices, and prospects for the future. IDA Paper P-1788, June 1984
6. Redwine, S, Riddle, T: Software technology maturation. Proc 8th International Conference on Software Engineering, pp 189–200, May 1985
7. Shaw, M: Prospects for an engineering discipline of software. IEEE Software, pp 15–24, November 1990
8. Shaw, M: The coming-of-age of software architecture research. Proc 23rd International Conference on Software Engineering (ICSE 2001), pp 656–664a
9. Tichy, WF, Lukowicz, P, Prechelt, L, Heinz, EA: Experimental evaluation in computer science: a quantitative study. J Syst Software 28(1):9–18, 1995
10. Tichy, WF: Should computer scientists experiment more? 16 reasons to avoid experimentation. IEEE Comp 31(5): 32–40, 1998
11. Zelkowitz, MV, Wallace, D: Experimental validation in software engineering. Inf Software Technol 39(11):735–744, 1997
12. Zelkowitz, MV, Wallace, D: Experimental models for validating technology. IEEE Comput 31(5):23–31, 1998