Writing Good Software Engineering Research Papers

Minitutorial

Mary Shaw
Carnegie Mellon University
mary.shaw@cs.cmu.edu

Abstract

Software engineering researchers solve problems of several different kinds. To do so, they produce several different kinds of results, and they should develop appropriate evidence to validate these results. They often report their research in conference papers. I analyzed the abstracts of research papers submitted to ICSE 2002 in order to identify the types of research reported in the submitted and accepted papers, and I observed the program committee discussions about which papers to accept. This report presents the research paradigms of the papers, common concerns of the program committee, and statistics on success rates. This information should help researchers design better research projects and write papers that present their results to best advantage.

Keywords: research design, research paradigms, validation, software profession, technical writing

1. Introduction

In software engineering, research papers are customary vehicles for reporting results to the research community. In a research paper, the author explains to an interested reader what he or she accomplished, and how the author accomplished it, and why the reader should care. A good research paper should answer a number of questions:

- What, precisely, was your contribution?
  - What question did you answer?
  - Why should the reader care?
  - What larger question does this address?
- What is your new result?
  - What new knowledge have you contributed that the reader can use elsewhere?
  - What previous work (yours or someone else's) do you build on? What do you provide a superior alternative to?
  - How is your result different from and better than this prior work?
  - What, precisely and in detail, is your new result?
- Why should the reader believe your result?
  - What standard should be used to evaluate your claim?
  - What concrete evidence shows that your result satisfies your claim?

If you answer these questions clearly, you'll probably communicate your result well. If in addition your result represents an interesting, sound, and significant contribution to our knowledge of software engineering, you'll have a good chance of getting it accepted for publication in a conference or journal.

Other fields of science and engineering have well-established research paradigms. For example, the experimental model of physics and the double-blind studies of medicines are understood, at least in broad outline, not only by the research community but also by the public at large. In addition to providing guidance for the design of research in a discipline, these paradigms establish the scope of scientific disciplines through a social and political process of "boundary setting" [5].

Software engineering, however, has not yet developed this sort of well-understood guidance. I previously [19, 20] discussed early steps toward such understanding, including a model of the way software engineering techniques mature [17, 18] and critiques of the lack of rigor in experimental software engineering [1, 22, 23, 24, 25]. Those discussions critique software engineering research reports against the standards of classical paradigms. The discussion here differs from those in that this discussion reports on the types of papers that are accepted in practices as good research reports. Another current activity, the Impact Project [7] seeks to trace the influence of software engineering research on practice. The discussion here focuses on the paradigms rather than the content of the research

This report examines how software engineers answer the questions above, with emphasis on the design of the research project and the organization of the report. Other sources (e.g., [4]) deal with specific issues of technical writing. Very concretely, the examples here come from the papers submitted to ICSE 2002 and the program committee review of those papers. These examples report research results in software engineering. Conferences often include other kinds of papers, including experience reports, materials on software engineering education, and opinion essays.

Proceedings of the 25th International Conference on Software Engineering (ICSE'03)
0270-5257/03 $17.00 © 2003 IEEE

2. What, precisely, was your contribution?

Before reporting what you did, explain what problem you set out to solve or what question you set out to answer —and why this is important.

2.1 What kinds of questions do software engineers investigate?

Generally speaking, software engineering researchers seek better ways to develop and evaluate software. Development includes all the synthetic activities that involve creating and modifying the software, including the code, design documents, documentation, etc. Evaluation includes all the analytic activities associated with predicting, determining, and estimating properties of the software systems, including both functionality and extra-functional properties such as performance or reliability.

Software engineering research answers questions about methods of development or analysis, about details of designing or evaluating a particular instance, about generalizations over whole classes of systems or techniques, or about exploratory issues concerning existence or feasibility. Table 1 lists the types of research questions that are asked by software engineering research papers and provides specific question templates.

Table 1. Types of software engineering research questions

Type of question	Examples
Method or means of development	How can we do/create/modify/evolve (or automate doing) X? What is a better way to do/create/modify/evolve X?
Method for analysis or evaluation	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	How good is Y? What is property X of artifact/method Y? What is a (better) design, implementation, maintenance, or adaptation for application X? 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 its important characteristics? What is a good formal/empirical model for X? What are the varieties of X, how are they related?
Feasibility study or exploration	Does X even exist, and if so what is it like? Is it possible to accomplish X at all?

The first two types of research produce methods of development or of analysis that the authors investigated in one setting, but that can presumably be applied in other settings. The third type of research deals explicitly with some particular system, practice, design or other instance of a system or method; these may range from narratives about industrial practice to analytic comparisons of alternative designs. For this type of research the instance itself should have some broad appeal—an evaluation of Java is more likely to be accepted than a simple evaluation of the toy language you developed last summer. Generalizations or characterizations explicitly rise above the examples presented in the paper. Finally, papers that deal with an issue in a completely new way are sometimes treated differently from papers that improve on prior art, so "feasibility" is a separate category (though no such papers were submitted to ICSE 2002).

Newman's critical comparison of HCI and traditional engineering papers [12] found that the engineering papers were mostly incremental (improved model, improved technique), whereas many of the HCI papers broke new ground (observations preliminary to a model, brand new technique). One reasonable interpretation is that the traditional engineering disciplines are much more mature than HCI, and so the character of the research might reasonably differ [17, 18]. Also, it appears that different disciplines have different expectations about the "size" of a research result—the extent to which it builds on existing knowledge or opens new questions. In the case of ICSE, the kinds of questions that are of interest and the minimum interesting increment may differ from one area to another.

2.2 Which of these are most common?

The most common kind of ICSE paper reports an improved method or means of developing software—that is, of designing, implementing, evolving, maintaining, or otherwise operating on the software system itself. Papers addressing these questions dominate both the submitted and the accepted papers. Also fairly common are papers about methods for reasoning about software systems, principally analysis of correctness (testing and verification). Analysis papers have a modest acceptance edge in this very selective conference.

Table 2 gives the distribution of submissions to ICSE 2002, based on reading the abstracts (not the full papers—but remember that the abstract tells a reader what to expect from the paper). For each type of research question, the table gives the number of papers submitted and accepted, the percentage of the total paper set of each kind, and the acceptance ratio within each type of question. Figures 1 and 2 show these counts and distributions.

Table 2. Types of research questions represented in ICSE 2002 submissions and acceptances

Type of question	Submitted	Accepted	Ratio Acc/Sub
Method or means of development	142(48%)	18 (42%)	(13%)
Method for analysis or evaluation	95 (32%)	19 (44%)	(20%)
Design, evaluation, or analysis of a particular instance	43 (14%)	5 (12%)	(12%)
Generalization or characterization	18 (6%)	1 (2%)	(6%)
Feasibility study or exploration	0 (0%)	0 (0 %)	(0%)
TOTAL	298(100.0%)	43 (100.0%)	(14%)

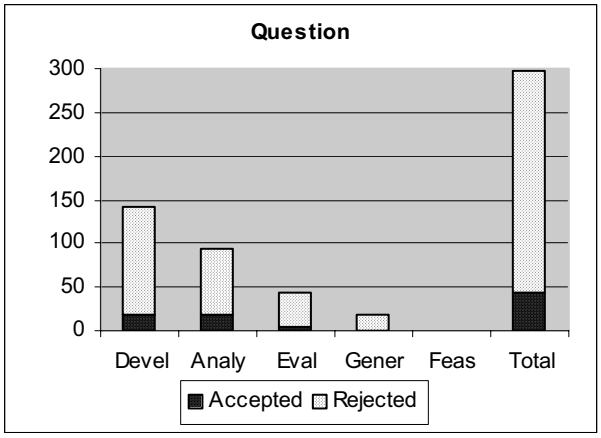

Figure 1. Counts of acceptances and rejections by type of research question

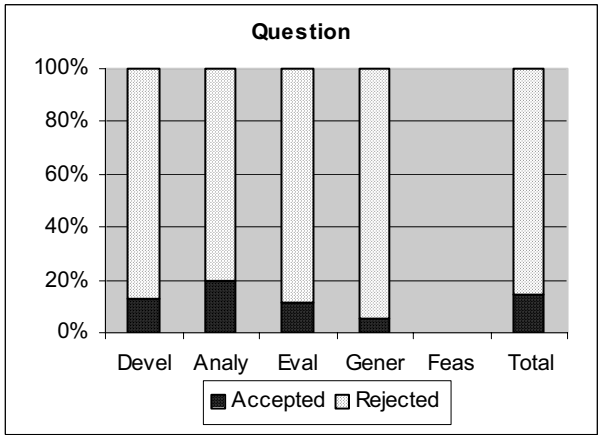

Figure 2. Distribution of acceptances and rejections by type of research question

2.3 What do program committees look for?

Acting on behalf of prospective readers, the program committee looks for a clear statement of the specific problem you solved—the question about software development you answered—and an explanation of how the answer will help solve an important software engineering problem. You'll devote most of your paper to describing your result, but you should begin by explaining what question you're answering and why the answer matters.

If the program committee has trouble figuring out whether you developed a new evaluation technique and demonstrated it on an example, or applied a technique you reported last year to a new real-world example, or evaluated the use of a well-established evaluation technique, you have not been clear.

3. What is your new result?

Explain precisely what you have contributed to the store of software engineering knowledge and how this is useful beyond your own project.

3.1 What kinds of results do software engineers produce?

The tangible contributions of software engineering research may be procedures or techniques for development or analysis; they may be models that generalize from specific examples, or they may be specific tools, solutions, or results about particular systems. Table 3 lists the types of research results that are reported in software engineering research papers and provides specific examples.

3.2 Which of these are most common?

By far the most common kind of ICSE paper reports a new procedure or technique for development or analysis. Models of various degrees of precision and formality were also common, with better success rates for quantitative than for qualitative models. Tools and notations were well represented, usually as auxiliary results in combination with a procedure or technique. Table 4 gives the distribution of submissions to ICSE 2002, based on reading the abstracts (but not the papers), followed by graphs of the counts and distributions in Figures 3 and 4.

Table 3. Types of software engineering research results	
Type of result	**Examples**
Procedure or technique	New or better way to do some task, such as design, implementation, maintenance, measurement, evaluation, selection from alternatives; includes techniques for implementation, representation, management, and analysis; a technique should be operational—not advice or guidelines, but a procedure
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, well-organized interesting observations
Empirical model	Empirical predictive model based on observed data
Analytic model	Structural model that permits formal analysis or automatic manipulation
Tool or notation	Implemented tool that embodies a technique; formal language to support a technique or model (should have a calculus, semantics, or other basis for computing or doing inference)
Specific solution, prototype, answer, or judgment	Solution to application problem that shows application of SE principles – may be design, prototype, or full implementation; careful analysis of a system or its development, result of a specific analysis, evaluation, or comparison
Report	Interesting observations, rules of thumb, but not sufficiently general or systematic to rise to the level of a descriptive model.

Table 4. Types of research results represented in ICSE 2002 submissions and acceptances			
Type of result	**Submitted**	**Accepted**	**Ratio Acc/Sub**
Procedure or technique	152(44%)	28 (51%)	18%
Qualitative or descriptive model	50 (14%)	4 (7%)	8%
Empirical model	4 (1%)	1 (2%)	25%
Analytic model	48 (14%)	7 (13%)	15%
Tool or notation	49 (14%)	10 (18%)	20%
Specific solution, prototype, answer, or judgment	34 (10%)	5 (9%)	15%
Report	11 (3%)	0 (0%)	0%
TOTAL	348(100.0%)	55 (100.0%)	16%

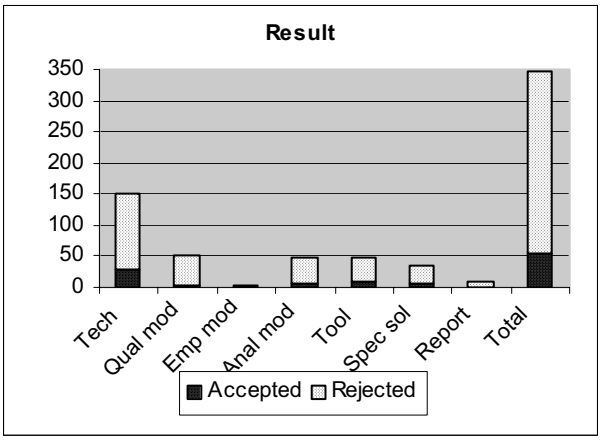

Figure 3. Counts of acceptances and rejections by type of result

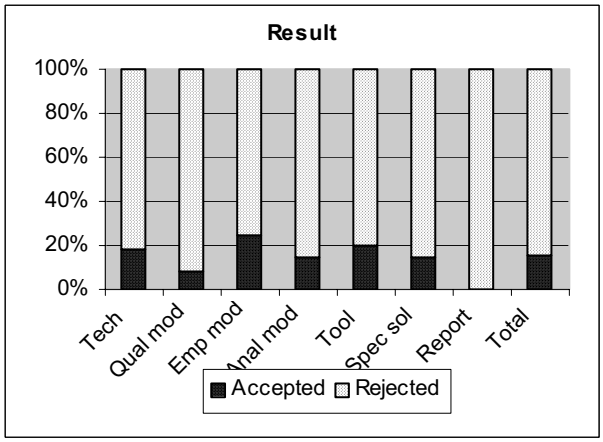

Figure 4. Distribution of acceptances and rejections by type of result

The number of results is larger than the number of papers because 50 papers included a supporting result, usually a tool or a qualitative model.

Research projects commonly produce results of several kinds. However, conferences, including ICSE, usually impose strict page limits. In most cases, this provides too little space to allow full development of more than one idea, perhaps with one or two supporting ideas. Many authors present the individual ideas in conference papers, and then synthesize them in a journal article that allows space to develop more complex relations among results.

3.3 What do program committees look for?

The program committee looks for interesting, novel, exciting results that significantly enhance our ability to develop and maintain software, to know the quality of the software we develop, to recognize general principles about software, or to analyze properties of software.

You should explain your result in such a way that someone else could use your ideas. Be sure to explain what's novel or original – is it the idea, the application of the idea, the implementation, the analysis, or what?

Define critical terms precisely. Use them consistently. The more formal or analytic the paper, the more important this is.

Here are some questions that the program committee may ask about your paper:

What, precisely, do you claim to contribute?

Does your result fully satisfy your claims? Are the definitions precise, and are terms used consistently?

Authors tend to have trouble in some specific situations. Here are some examples, with advice for staying out of trouble:

- *If your result ought to work on large systems*, explain why you believe it scales.
- *If you claim your method is "automatic",* using it should not require human intervention. If it's automatic when it's operating but requires manual assistance to configure, say so. If it's automatic except for certain cases, say so, and say how often the exceptions occur.
- *If you claim your result is "distributed",* it probably should not have a single central controller or server. If it does, explain what part of it is distributed and what part is not.
- *If you're proposing a new notation for an old problem,* explain why your notation is clearly superior to the old one.
- *If your paper is an "experience report",* relating the use of a previously-reported tool or technique in a practical software project, be sure that you explain what idea the reader can take away from the paper to use in other settings. If that idea is increased confidence in the tool or technique, show how your experience should increase the reader's confidence for applications beyond the example of the paper.

What's new here?

The program committee wants to know what is novel or exciting, and why. What, specifically, is the contribution? What is the increment over earlier work by the same authors? by other authors? Is this a sufficient increment, given the usual standards of subdiscipline?

Above all, the program committee also wants to know what you actually contributed to our store of knowledge about software engineering. Sure, you wrote this tool and tried it out. But was your contribution the technique that is embedded in the tool, or was it making a tool that's more effective than other tools that implement the technique, or was it showing that the tool you described in a previous paper actually worked on a practical large-scale problem? It's better for you as the author to explain than for the program committee to guess. Be clear about your claim …

Awful	▼	• I completely and generally solved … (unless you actually did!)
Bad	▼	• I worked on galumphing. (or studied, investigated, sought, explored)
Poor	▼	• I worked on improving galumphing. (or contributed to, participated in, helped with)
Good	▲	• I showed the feasibility of composing blitzing with flitzing. • I significantly improved the accuracy of the standard detector. (or proved, demonstrated, created, established, found, developed)
Better	▲	• I automated the production of flitz tables from specifications. • With a novel application of the blivet transform, I achieved a 10% increase in speed and a 15% improvement in coverage over the standard method.

Use verbs that show results and achievement, not just effort and activity.

"***Try*** not. ***Do,*** or ***do*** not. There is no ***try***." -- Yoda

What has been done before? How is your work different or better?

What existing technology does your research build on? What existing technology or prior research does your research provide a superior alternative to? What's new here compared to your own previous work? What alternatives have other researchers pursued, and how is your work different or better?

As in other areas of science and engineering, software engineering knowledge grows incrementally. Program committees are very interested in your interpretation of prior work in the area. They want to know how your work is related to the prior work, either by building on it or by providing an alternative. If you don't explain this, it's hard for the program committee to understand how you've added to our store of knowledge. You may also damage your credibility if the program committee can't tell whether you know about related work.

Explain the relation to other work clearly …

Awful	▼	The galumphing problem has attracted much attention [3,8,10,18,26,32,37]
Bad	▼	Smith [36] and Jones [27] worked on galumphing.
Poor	▼	Smith [36] addressed galumphing by blitzing, whereas Jones [27] took a flitzing approach.
Good	▲	Smith's blitzing approach to galumphing [36] achieved 60% coverage [39]. Jones [27] achieved 80% by flitzing, but only for pointer-free cases [16].
Better	▲	Smith's blitzing approach to galumphing [36] achieved 60% coverage [39]. Jones [27] achieved 80% by flitzing, but only for pointer-free cases [16]. We modified the blitzing approach to use the kernel representation of flitzing and achieved 90% coverage while relaxing the restriction so that only cyclic data structures are prohibited.

What, precisely, is the result?

Explain what your result is and how it works. Be concrete and specific. Use examples.

If you introduce a new model, be clear about its power. How general is it? Is it based on empirical data, on a formal semantics, on mathematical principles? How formal is it—a qualitative model that provides design guidance may be as valuable as a mathematical model of some aspect of correctness, but they will have to satisfy different standards of proof. Will the model scale up to problems of size appropriate to its domain?

If you introduce a new metric, define it precisely. Does it measure what it purports to measure and do so better than the alternatives? Why?

If you introduce a new architectural style, design pattern, or similar design element, treat it as if it were a new generalization or model. How does it differ from the alternatives? In what way is it better? What real problem does it solve? Does it scale?

If your contribution is principally the synthesis or integration of other results or components, be clear about why the synthesis is itself a contribution. What is novel, exciting, or nonobvious about the integration? Did you generalize prior results? Did you find a better representation? Did your research improve the individual results or components as well as integrating them? A paper that simply reports on using numerous elements together is not enough, even if it's well-engineered. There must be an idea or lesson or model that the reader can take from the paper and apply to some other situation.

If your paper is chiefly a report on experience applying research results to a practical problem, say what the reader can learn from the experience. Are your conclusions strong and well-supported? Do you show comparative data and/or statistics? An anecdotal report on a single project is usually not enough. Also, if your report mixes additional innovation with validation through experience, avoid confusing your discussion of the innovation with your report on experience. After all, if you changed the result before you applied it, you're evaluating the changed result. And if you changed the result while you were applying it, you may have confounded the experiences with the two versions.

If a tool plays a featured role in your paper, what is the role of the tool? Does it simply support the main contribution, or is the tool itself a principal contribution, or is some aspect of the tool's use or implementation the main point? Can a reader apply the idea without the tool? If the tool is a central part of result, what is the technical innovation embedded in the tool or its implementation?

If a system implementation plays a featured role in your paper, what is the role of the implementation? Is the system sound? Does it do what you claim it does? What ideas does the system demonstrate?

- *If the implementation illustrates an architecture or design strategy*, what does it reveal about the architecture? What was the design rationale? What were the design tradeoffs? What can the reader apply to a different implementation?
- *If the implementation demonstrates an implementation technique*, how does it help the reader use the technique in another setting?
- *If the implementation demonstrates a capability or performance improvement*, what concrete evidence does it offer to support the claim?
- *If the system is itself the result*, in what way is it a contribution to knowledge? Does it, for example, show you can do something that no one has done before (especially if people doubted that this could be done)?

4. Why should the reader believe your result?

Show evidence that your result is valid—that it actually helps to solve the problem you set out to solve.

4.1. What kinds of validation do software engineers do?

Software engineers offer several kinds of evidence in support of their research results. It is essential to select a form of validation that is appropriate for the type of research result and the method used to obtain the result. As an obvious example, a formal model should be supported by rigorous derivation and proof, not by one or two simple examples. On the other hand, a simple example derived from a practical system may play a major role in validating a new type of development method. Table 5 lists the types of research validation that are used in software engineering research papers and provides specific examples. In this table, the examples are keyed to the type of result they apply to.

Table 5. Types of software engineering research validation

Type of validation	Examples
Analysis	I have analyzed my result and find it satisfactory through rigorous analysis, e.g. … For a formal model … rigorous derivation and proof For an empirical model … data on use in controlled situation For a controlled experiment … carefully designed experiment with statistically significant results
Evaluation	Given the stated criteria, my result... For a descriptive model … adequately describes phenomena of interest … For a qualitative model … accounts for the phenomena of interest… For an empirical model … is able to predict … because …, or … generates results that fit actual data … Includes feasibility studies, pilot projects
Experience	My result has been used on real examples by someone other than me, and the evidence of its correctness/usefulness/effectiveness is … For a qualitative model … narrative For an empirical model or tool … data, usually statistical, on practice For a notation or technique … comparison of systems in actual use
Example	Here's an example of how it works on For a technique or procedure …a "slice of life" example based on a real system … For a technique or procedure …a system that I have been developing … For a technique or procedure … a toy example, perhaps motivated by reality The "slice of life" example is most likely to be convincing, especially if accompanied by an explanation of why the simplified example retains the essence of the problem being solved. Toy or textbook examples often fail to provide persuasive validation, (except for standard examples used as model problems by the field).
Persuasion	I thought hard about this, and I believe passionately that ... For a technique … if you do it the following way, then … For a system … a system constructed like this would … For a model … this example shows how my idea works Validation purely by persuasion is rarely sufficient for a research paper. Note, though, that if the original question was about feasibility, a working system, even without analysis, can suffice
Blatant assertion	No serious attempt to evaluate result. This is highly unlikely to be acceptable

4.2 Which of these are most common?

Alas, well over a quarter of the ICSE 2002 abstracts give no indication of how the paper's results are validated, if at all. Even when the abstract mentions that the result was applied to an example, it was not always clear whether the example was a textbook example, or a report on use in the field, or something in between.

The most successful kinds of validation were based on analysis and real-world experience. Well-chosen examples were also successful. Persuasion was not persuasive, and narrative evaluation was only slightly more successful. Table 6 gives the distribution of submissions to ICSE 2002, based on reading the abstracts (but not the papers), followed by graphs of the counts and distributions. Figures 5 and 6 show these counts and distributions.

Table 6. Types of research validation represented in ICSE 2002 submissions and acceptances			
Type of validation	**Submitted**	**Accepted**	**Ratio Acc/Sub**
Analysis	48 (16%)	11 (26%)	23%
Evaluation	21 (7%)	1 (2%)	5%
Experience	34 (11%)	8 (19%)	24%
Example	82 (27%)	16 (37%)	20%
Some example, can't tell whether it's toy or actual use	6 (2%)	1 (2%)	17%
Persuasion	25 (8%)	0 (0.0%)	0%
No mention of validation in abstract	84 (28%)	6 (14%)	7%
TOTAL	300(100.0%)	43 (100.0%)	14%

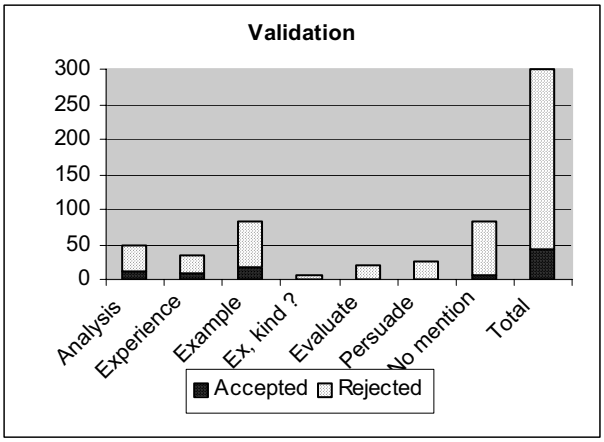

Figure 5. Counts of acceptances and rejections by type of validation

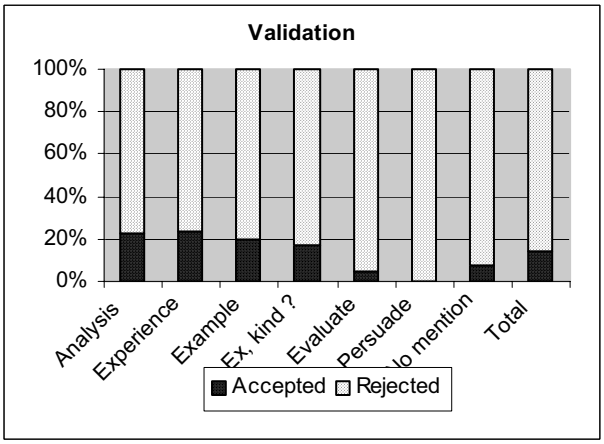

Figure 6. Distribution of acceptances and rejections by type of validation

4.3 What do program committees look for?

The program committee looks for solid evidence to support your result. It's not enough that your idea works for you, there must also be evidence that the idea or the technique will help someone else as well.

The statistics above show that analysis, actual experience in the field, and good use of realistic examples tend to be the most effective ways of showing why your result should be believed. Careful narrative, qualitative analysis can also work if the reasoning is sound.

Why should the reader believe your result?

Is the paper argued persuasively? What evidence is presented to support the claim? What kind of evidence is offered? Does it meet the usual standard of the subdiscipline?

Is the kind of evaluation you're doing described clearly and accurately? "Controlled experiment" requires more than data collection, and "case study" requires more than anecdotal discussion. Pilot studies that lay the groundwork for controlled experiments are often not publishable by themselves.

Is the validation related to the claim? If you're claiming performance improvement, validation should analyze performance, not ease of use or generality. And conversely.

Is this such an interesting, potentially powerful idea that it should get exposure despite a shortage of concrete evidence?

Authors tend to have trouble in some specific situations. Here are some examples, with advice for staying out of trouble:

- *If you claim to improve on prior art,* compare your result objectively to the prior art.
- *If you used an analysis technique,* follow the rules of that analysis technique. If the technique is not a common one in software engineering (e.g., meta-analysis, decision theory, user studies or other behavioral analyses), explain the technique and standards of proof, and be clear about your adherence to the technique.
- *If you offer practical experience as evidence for your result,* establish the effect your research has. If at all possible, compare similar situations with and without your result.

- *If you performed a controlled experiment,* explain the experimental design. What is the hypothesis? What is the treatment? What is being controlled? What data did you collect, and how did you analyze it? Are the results significant? What are the potentially confounding factors, and how are they handled? Do the conclusions follow rigorously from the experimental data?
- *If you performed an empirical study,* explain what you measured, how you analyzed it, and what you concluded. What data did you collect, and how? How is the analysis related to the goal of supporting your claim about the result? Do not confuse correlation with causality.
- *If you use a small example for explaining the result,* provide additional evidence of its practical use and scalability.

5. How do you combine the elements into a research strategy?

It is clear that not all combinations of a research question, a result, and a validation strategy lead to good research. Software engineering has not developed good general guidance on this question.

Tables 1, 3, and 5 define a 3-dimensional space. Some portions of that space are densely populated: One common paradigm is to find a better way to perform some software development or maintenance task, realize this in a concrete procedure supported by a tool, and evaluate the effectiveness of this procedure and tool by determining how its use affects some measure (e.g., error rates) of quality. Another common paradigm is to find a better way to evaluate a formalizable property of a software system, develop a formal model that supports inference, and to show that the new model allows formal analysis or proof of the properties of interest.

Clearly, the researcher does not have free choice to mix and match the techniques—validating the correctness of a formal model through field study is as inappropriate as attempting formal verification of a method based on good organization of rules of thumb.

Selecting a type of result that will answer a given question usually does not seem to present much difficulty, at least for researchers who think carefully about the choice. Blindly adopting the research paradigm someone used last year for a completely different problem is a different case, of course, and it can lead to serious misfits.

Choosing a good form of validation is much harder, and this is often a source of difficulty in completing a successful paper. Table 6 shows some common good matches. This does not, unfortunately, provide complete guidance.

When I advise PhD students on the validation section of their theses, I offer the following heuristic: Look carefully at the short statement of the result—the principal claim of the thesis. This often has two or three clauses (e.g., I found an efficient and complete method …"); if so, each presents a separate validation problem. Ask of each clause whether it is a global statement ("always", "fully"), a qualified statement ("a 25% improvement", "for noncyclic structures…"), or an existential statement {"we found an instance of"). Global statements often require analytic validation, qualified statements can often be validated by evaluation or careful examination of experience, and existential statements can sometimes be validated by a single positive example. A frequent result of this discussion is that students restate the thesis claims to reflect more precisely what the theses actually achieve. If we have this discussion early enough in the thesis process, students think about planning the research with demonstrable claims in mind.

Concretely, Table 7 shows the combinations that were represented among the accepted papers at ICSE 2002, omitting the 7 for which the abstracts were unclear about validation:

Table 7. Paradigms of ICSE2002 acceptances

Question	Result	Validation	#
Devel method	Procedure	Analysis	2
Devel method	Procedure	Experience	3
Devel method	Procedure	Example	3
Devel method	Qual model	Experience	2
Devel method	Analytic model	Experience	2
Devel method	Notation or tool	Experience	1
Analysis method	Procedure	Analysis	5
Analysis method	Procedure	Evaluation	1
Analysis method	Procedure	Experience	2
Analysis method	Procedure	Example	6
Analysis method	Analytic model	Experience	1
Analysis method	Analytic model	Example	2
Analysis method	Tool	Analysis	1
Eval of instance	Specific analysis	Analysis	3
Eval of instance	Specific analysis	Example	2

6. Does the abstract matter?

The abstracts of papers submitted to ICSE convey a sense of the kinds of research submitted to the conference. Some abstracts were easier to read and (apparently) more informative than others. Many of the clearest abstracts had a common structure:

- Two or three sentences about the current state of the art, identifying a particular problem
- One or two sentences about what this paper contributes to improving the situation

- One or two sentences about the specific result of the paper and the main idea behind it
- A sentence about how the result is demonstrated or defended

Abstracts in roughly this format often explained clearly what readers could expect in the paper.

Acceptance rates were highest for papers whose abstracts indicate that analysis or experience provides evidence in support of the work. Decisions on papers were made on the basis of the whole papers, of course, not just the abstracts—but it is reasonable to assume that the abstracts reflect what's in the papers.

Whether you like it or not, people judge papers by their abstracts and read the abstract in order to decide whether to read the whole paper. It's important for the abstract to tell the story. Don't assume, though, that simply adding a sentence about analysis or experience to your abstract is sufficient; the paper must deliver what the abstract promises

7. Questions you might ask about this report

7.1. Is this a sure-fire recipe?

No, not at all. First, it's not a recipe. Second, not all software engineers share the same views of interesting and significant research. Even if your paper is clear about what you've done and what you can conclude, members of a program committee may not agree about how to interpret your result. These are usually honest technical disagreements, and committee members will try hard to understand what you have done. You can help by explaining your work clearly; this report should help you do that.

7.2 Is ICSE different from other conferences?

ICSE recognizes several distinct types of technical papers [6]. For 2002, they were published separately in the proceedings

Several other conferences offer "how to write a paper" advice:

In 1993, several OOPSLA program committee veterans gave a panel on "How to Get a Paper Accepted at OOPSLA" [9]. This updated the 1991 advice for the same conference [14]

SIGSOFT offers two essays on getting papers accepted, though neither was actually written for a software engineering audience. They are "How to Have Your Abstract Rejected" [26] (which focuses on theoretical papers) and "Advice to Authors of Extended Abstracts", which was written for PLDI. [16].

Rather older, Levin and Reddell, the 1983 SOSP (operating systems) program co-chairs offered advice on writing a good systems paper [11]. USENIX now provides this advice to its authors. Also in the systems vein, Partridge offers advice on "How to Increase the Chances Your Paper is Accepted at ACM SIGCOMM" [15].

SIGCHI offers a "Guide to Successful Papers Submission" that includes criteria for evaluation and discussion of common types of CHI results, together with how different evaluation criteria apply for different types of results [13]. A study [8] of regional factors that affect acceptance found regional differences in problems with novelty, significance, focus, and writing quality.

In 1993, the SIGGRAPH conference program chair wrote a discussion of the selection process, "How to Get Your SIGGRAPH Paper Rejected" [10]. The 2003 SIGGRAPH call for papers [21] has a description of the review process and a frequently-asked questions section with an extensive set of questions on "Getting a Paper Accepted".

7.3. What about this report itself?

People have asked me, "what would happen if you submitted this to ICSE?" Without venturing to predict what any given ICSE program committee would do, I note that as a research result or technical paper (a "finding" in Brooks' sense [3]) it falls short in a number of ways:

- There is no attempt to show that anyone else can apply the model. That is, there is no demonstration of inter-rater reliability, or for that matter even repeatability by the same rater.
- The model is not justified by any principled analysis, though fragments, such as the types of models that can serve as results, are principled. In defense of the model, Bowker and Starr [2] show that useful classifications blend principle and pragmatic descriptive power.
- Only one conference and one program committee is reflected here.
- The use of abstracts as proxies for full papers is suspect.
- There is little discussion of related work other than the essays about writing papers for other conferences. Although discussion of related work does appear in two complementary papers [19, 20], this report does not stand alone.

On the other hand, I believe that this report does meet Brooks' standard for "rules of thumb" (generalizations, signed by the author but perhaps incompletely supported by data, judged by usefulness and freshness), and I offer it in that sense.

8. Acknowledgements

This work depended critically on access to the entire body of submitted papers for the ICSE 2002 conference,

which would not have been possible without the cooperation and encouragement of the ICSE 2002 program committee. The development of these ideas has also benefited from discussion with the ICSE 2002 program committee, with colleagues at Carnegie Mellon, and at open discussion sessions at FSE Conferences. The work has been supported by the A. J. Perlis Chair at Carnegie Mellon University.

9. References

1. Victor R. Basili. The experimental paradigm in software engineering. In *Experimental Software Engineering Issues: Critical Assessment and Future Directives*. Proc of Dagstuhl-Workshop, H. Dieter Rombach, Victor R. Basili, and Richard Selby (eds), published as *Lecture Notes in Computer Science #706*, Springer-Verlag 1993.
2. Geoffrey Bowker and Susan Leigh Star: *Sorting Things Out: Classification and Its Consequences*. MIT Press, 1999
3. Frederick P. Brooks, Jr. Grasping Reality Through Illusion—Interactive Graphics Serving Science. *Proc 1988 ACM SIGCHI Human Factors in Computer Systems Conf (CHI '88)* pp. 1-11.
4. Rebecca Burnett. *Technical Communication*. Thomson Heinle 2001.
5. Thomas F. Gieryn. *Cultural Boundaries of Science: Credibility on the line*. Univ of Chicago Press, 1999.
6. ICSE 2002 Program Committee. *Types of ICSE papers*. http://icse-conferences.org/2002/info/paperTypes.html
7. Impact Project. "Determining the impact of software engineering research upon practice. Panel summary, *Proc. 23rd International Conference on Software Engineering (ICSE 2001)*, 2001
8. Ellen Isaacs and John Tang. *Why don't more non-North-American papers get accepted to CHI?* http://acm.org/sigchi/bulletin/1996.1/isaacs.html
9. Ralph E. Johnson & panel. How to Get a Paper Accepted at OOPSLA. *Proc OOPSLA'93*, pp. 429-436, http://acm.org/sigplan/oopsla/oopsla96/how93.html
10. Jim Kajiya. How to Get Your SIGGRAPH Paper Rejected. Mirrored at http://www.cc.gatech.edu/student.services/phd/phd-advice/kajiya
11. Roy Levin and David D. Redell. How (and How Not) to Write a Good Systems Paper. *ACM SIGOPS Operating Systems Review*, Vol. 17, No. 3 (July, 1983), pages 35-40. http://ftp.digital.com/pub/DEC/SRC/other/SOSPadvice.txt
12. William Newman. A preliminary analysis of the products of HCI research, using pro forma abstracts. *Proc 1994 ACM SIGCHI Human Factors in Computer Systems Conf (CHI '94)*, pp.278-284.
13. William Newman et al. *Guide to Successful Papers Submission at CHI 2001*. http://acm.org/sigs/sigchi/chi2001/call/submissions/guide-papers.html
14. OOPSLA '91 Program Committee. How to get your paper accepted at OOPSLA. *Proc OOPSLA'91*, pp.359-363. http://acm.org/sigplan/oopsla/oopsla96/how91.html
15. Craig Partridge. How to Increase the Chances your Paper is Accepted at ACM SIGCOMM. http://www.acm.org/sigcomm/conference-misc/author-guide.html
16. William Pugh and PDLI 1991 Program Committee. *Advice to Authors of Extended Abstracts*. http://acm.org/sigsoft/conferences/pughadvice.html
17. Samuel Redwine, et al. *DoD Related Software Technology Requirements, Practices, and Prospects for the Future*. IDA Paper P-1788, June 1984.
18. S. Redwine & W. Riddle. Software technology maturation. *Proceedings of the Eighth International Conference on Software Engineering*, May 1985, pp. 189-200.
19. Mary Shaw. The coming-of-age of software architecture research. *Proc. 23rd Int'l Conf on Software Engineering (ICSE 2001)*, pp. 656-664a.
20. Mary Shaw. What makes good research in software engineering? Presented at ETAPS 02, appeared in Opinion Corner department, *Int'l Jour on Software Tools for Tech Transfer*, vol 4, DOI 10.1007/s10009-002-0083-4, June 2002.
21. SigGraph 2003 Call for Papers. http://www.siggraph.org/s2003/cfp/papers/index.html
22. W. F. Tichy, P. Lukowicz, L. Prechelt, & E. A. Heinz. "Experimental evaluation in computer science: A quantitative study." *Journal of Systems Software,* Vol. 28, No. 1, 1995, pp. 9-18.
23. Walter F. Tichy. "Should computer scientists experiment more? 16 reasons to avoid experimentation." *IEEE Computer*, Vol. 31, No. 5, May 1998
24. Marvin V. Zelkowitz and Delores Wallace. Experimental validation in software engineering. *Information and Software Technology*, Vol 39, no 11, 1997, pp. 735-744.
25. Marvin V. Zelkowitz and Delores Wallace. Experimental models for validating technology. *IEEE Computer*, Vol. 31, No. 5, 1998, pp.23-31.
26. Mary-Claire van Leunen and Richard Lipton. *How to have your abstract rejected.* http://acm.org/sigsoft/conferences/vanLeunenLipton.html