

CSci 780
Advanced Software Engineering

Effective Papers

“Writing Good Software Engineering Research Papers” – Mary Shaw

**“An Evaluation of the Ninth SOSP Submissions”
– Roy Levin and David D. Redell**

8/27/2004

1

Outline

- Types of papers
- Structure
 - Sample Abstracts
 - Sample Related Work
- Hints

8/27/2004

2

Types of Papers—Forums (1/4)

- Position papers (2-3 pages)
 - State a point of view: challenge the readers
 - Often solicited from experienced researchers
 - Research results not necessary
- Workshop papers (3-6 pages)
 - Position papers, or unevaluated ideas
 - Used to spur discussion, or decide who can attend

Neither of these gets you “credit”

8/27/2004

3

Types of Papers—Forums (2/4)

- Technical papers at top conferences (10-12 pages)
 - Present one good idea with recent supporting results
 - Reviewed by 3 people from the program committee
 - One chance to get in, one chance to revise
 - Publication time is usually about a year
 - Best paper awards

*Top conferences give you “credit”
(<20% acceptance rate)*

8/27/2004

4

Types of Papers—Forums (3/4)

- Journal papers (10-50 pages)
 - Present multiple related ideas with well-developed supporting results
 - Carefully reviewed by editorial board
 - Author usually has multiple chances to revise (if you get “accept with revisions”)
 - Publication time is usually about 2-5 years

Usually full “credit”

8/27/2004

5

Types of Papers—Forums (4/4)

- Technical Reports (10-50 pages)
 - No “credit”
 - Useful for user’s manuals, long appendix-like material
 - No review
 - Publication time is nil
- PhD Theses (200-500 pages)
 - 4-8 papers work of ideas
 - Brings together a body of related work
 - Reviewed by PhD committee
 - Published like a technical report

8/27/2004

6

Types of Papers—Content [Shaw] (1/2)

- Method or means of development
 - “Automated Support for Classifying Software Failure Reports”
 - 13% accepted, 42% of accepted papers
- Method for analysis or evaluation
 - “Cost Estimation for Web Applications”
 - 20% accepted, 44% of accepted papers
- Design, evaluation, or analysis of instance
 - “Software Technology in an Automotive Company—Major Challenges”
 - 12% accepted, 12% of accepted papers

8/27/2004

7

Types of Papers—Content [Shaw] (2/2)

- Generalization or characterization
 - “Toward an Understanding of the Motivation of Open Source Developers”
 - 6% accepted, 2% of accepted papers
- Feasibility study or exploration *
 - “Multiple Mass Market Applications As Components”
 - 0-100% accepted, 0% of accepted papers

8/27/2004

8

Structure

8/27/2004

9

“The Scientific Method”

- Problem
- Background
- Hypothesis
- Materials
- Procedure
- Data
- Data analysis
- Conclusion

8/27/2004

10

Conceptual Paper Structure

- Problem Statement
- Proposed Solution
- Evaluation Methodology
- Data
- Evaluation
- Related Work
- Conclusion

8/27/2004

11

Physical Paper Structure

- Abstract 150-200 words
- 1: Introduction 1-1½ pages
- 2: Background * 1-2 pages
- 3: Description 4-6 pages
- 4: Experiment & Data 2-3 pages
- 5: Evaluation/Discussion ½-2 pages
- 6: Related Work * ½-1 page
- 7: Conclusion 2 paragraphs
- Acknowledgements 2 sentences
- References ½-1 page

8/27/2004

12

The Abstract [Shaw]

- 2-3 sentences about current state of the art
 - The “problem statement”
- 1-2 sentences about the how you plan to fix it
 - The “approach” or “contribution”
- 1-2 sentences about the specific details
 - The “experiment”
- 1 sentence about how the result is defended
 - The “results”
 - Sometimes omitted for space

8/27/2004

13

Example #1

Software maintenance and evolution are the dominant activities in the software lifecycle. Modularization can separate design decisions and allow them to be independently evolved, but modularization often breaks down and complicated global changes are required. Tool support can reduce the costs of these unfortunate changes, but current tools are limited in their ability to manage information for large-scale software evolution. In this paper we argue that the map metaphor can serve as an organizing principle for the design of effective tools for performing global software changes. We describe the design of Aspect Browser, developed around the map metaphor, and discuss a case study of removing a feature from a 500,000 line program written in Fortran and C.

8/27/2004

14

Example #2

Modeling languages and the software tools which support them are essential to engineering. However, as these languages become more sophisticated, it becomes difficult to assure both the validity of their semantic specifications and the dependability of their program implementations. To ameliorate this problem we propose to develop shared semantic domains and corresponding implementations for otherwise unrelated modeling languages. The idea is to amortize investments at the intermediate level across multiple language definitions and implementations. To assess the practicality of this approach for modeling languages, we applied it to two languages for reliability modeling and analysis. In earlier work, we developed the intermediate semantic domain of *failure automata* (FA), which we used to formalize the semantics of *dynamic fault trees* (DFTs). In this paper, we show that a variant of the original FA can serve as a common semantic domain for both DFTs and reliability block diagrams (RBDs). Our experiences suggest that the use of a common semantic domain and a shared analyzer for expressions at this level can ease the task of formalizing and implementing modeling languages, reducing development costs and improving their dependability.

8/27/2004

15

Example #3

This paper describes a methodology for designing Open Implementations—software modules that can adapt or change their internals to accommodate the needs of different clients. Analysis techniques are used for capturing domain knowledge, user requirements, and domain properties that influence the module's eventual implementation. Design techniques are used for determining and refining the interfaces by which clients control the modules implementation strategies. The methodology has evolved over the past two years in several pilot projects.

8/27/2004

16

Example #4

This paper proposes a technique for identifying program properties that indicate errors. The technique generates machine learning models of program properties known to result from errors, and applies these models to program properties of user-written code to classify and rank properties that may lead the user to errors. Given a set of properties produced by the program analysis, the technique selects a subset of properties that are most likely to reveal an error.

An implementation, the Fault Invariant Classifier, demonstrates the efficacy of the technique. The implementation uses dynamic invariant detection to generate program properties. It uses support vector machine and decision tree learning tools to classify those properties. In our experimental evaluation, the technique increases the relevance (the concentration of fault-revealing properties) by a factor of 50 on average for the C programs, and 4.8 for the Java programs. Preliminary experience suggests that most of the fault-revealing properties do lead a programmer to an error.

8/27/2004

17

Introduction

- A long form of the abstract
- Discuss background if necessary
- Convince the reader that your problem is important, and your solution worth reading about
- Make your contributions and claims clear
- End with a “roadmap” for the rest of the paper

8/27/2004

18

Background/Related Work

- Background:
 - If the reader needs a lot to understand the paper
- Related work:
 - Discuss (carefully!) the point of the work
 - Relate it to what you are doing—How is your work better or different?
 - Gives you credibility

8/27/2004

19

Example #1

Our experience using Z/Eves is similar to that of Knight et al. [20], who used the PVS theorem prover on a modest-sized nuclear power plant specification. They too found it hard to formulate theorems and proof strategies, and were hindered by poor tool usability.

8/27/2004

20

Example #2

Another prototyping environment has been developed for the formal specification language TROLL [Jungclaus91]. It is called Tbench [Kusch95] and uses an editing and execution system for the specifications. In contrast to our environment, Tbench does not automatically generate a first software model, as a first step in the development process of the final system. There is another CASE tool for LCM [Wieringa94] called TCM (Toolkit for Conceptual Modeling), but in this case, the generation of prototypes is not allowed, to the best of our knowledge.

8/27/2004

21

Description

- Usually multiple sections
- Usually the easiest part to write
- Usually the longest part of the paper
 - Unless the whole paper is an empirical study
- A good example goes a long way
- Beware of strong statements: "never", "always"
- Support every claim

8/27/2004

22

Experiment & Data

- Convince the reader the experiment is sound
- Use good statistics
 - Error bars/confidence intervals
 - Beware scalar values
 - \$16 billion in foreign aid = 0.135% of US GDP
 - Beware percent values
 - 1 penny + 1 penny = 100% increase

Unfortunately, collecting hard data in SE is hard

8/27/2004

23

Evaluation/Discussion

- Discuss the weaknesses of your approach
- Discuss the weaknesses of your experiment
- Explain what you learned
- Explain how your hypothesis is supported

A negative result is still a result

8/27/2004

24

Conclusion

- Summarize in 1 paragraph
- Wax philosophical:
 - Broader research/impact
 - Future work
- Some people merge discussion and conclusion

8/27/2004

25

Acknowledgements and References

- Acknowledge
 - Grants: So you can show funding agencies what they're paying you for
 - Helpful people: implementors, industrial collaborators, discussion people
 - Anonymous reviewers
- References
 - Simple: use BibTeX

8/27/2004

26

Anti-Hints

8/27/2004

27

Anti-Hints [Levin & Redell] (1/2)

- Fail to identify a problem
- Fail to present a new idea
- Be unable to describe your approach during an elevator ride
- Ignorance of related work is bliss
- Deceive the reader
 - Don't tell them you haven't implemented it
 - Don't tell them when your approach doesn't work

8/27/2004

28

Anti-Hints [Levin & Redell] (2/2)

- Let them figure out your assumptions
- Make sure there's nothing the reader can learn or use from your paper
- Confuse them with useless formalism
- Keep telling them the read more for the secret
- Make them read a future paper to get the rest
- Any monkey can write a top conference paper

8/27/2004

29

Hints

8/27/2004

30

Hints: Use Keywords (1/6)

- Problem: “unfortunate”, “difficult”, “tedious”, “poor”
- Approach: “In this paper we present”, “We address this problem by”
- Experiment: “To evaluate”, “To assess”, “case study”, “implemented a system”
- Results: “Our experiences suggest”, “our data supports”
- Protection: “To the best of our knowledge”, “As far as we are aware”

8/27/2004

31

Hints: Remember That It's *Science* (2/6)

- The cool system is not the point—it's the idea that the system supports
- Don't say “prove” unless you mean it
- Describe your experiment as such
- What is your null hypothesis?
- Your experiment is one data point
 - “Our results demonstrate that the approach works”
 - “Our results indicate that the approach can work”

8/27/2004

32

Hints: Your Own Work (3/6)

- Cite your own previous work
- Relate the paper to your previous work
- You can plagiarize yourself!
- Minimally publishable unit? (MPU)

8/27/2004

33

Hints: Spin (4/6)

- Write the abstract first to clarify the paper
- “Spin” can make a mediocre paper a great one
 - More than fluff—putting the research in context
- Maybe write half the paper before starting work!
- SOSP theme: 5 person-years of effort, no research contribution!
 - Do a little revisionist history: find out what the right problem/context would have been
 - Dangerous: need to work in “fruitful” areas

8/27/2004

34

Hints: Writing (5/6)

- Use simple writing—the ideas are complicated enough
- Make sure the paper tells a “research story”
- Avoid third-level subsections
- Make sure figure text is readable
- Be honest and up-front, but not meek or defensive
- Learn the art of compression
- Always spell check and grammar check

8/27/2004

35

Hints: Miscellaneous (6/6)

- Get as many people to read it as possible
 - Finish it 1 month before the deadline
 - Reread it yourself after a couple weeks
- Learn your document system
 - Figure positioning, captions, cross-references
 - BibTeX in LaTeX
 - Styles in Word
- Use the conference or journal template
- *Always* get co-author sign-off before submission

8/27/2004

36