

On ICSE's "Most Influential" Papers

David Lorge Parnas

30.1 Background

The International Conference on Software Engineering has established a tradition of looking back ten conferences and selecting papers that have stood the test of time. The remarks below were prepared in connection with an acceptance speech at ICSE 17 where two colleagues and I received the award for the best paper of ICSE 7.

30.2 What Are the Best Papers of Our Most Important Software Engineering Conference?

The following seven papers have won a "best paper" retrospective award from the International Conference of Software Engineering, the major conference devoted to Software Engineering. When these awards were first given, they were called "Most Influential Paper" awards and this is what has stimulated the remarks that follow the list.

ICSE 1—September 1975

Marc J. Rochkind, "The Source Code Control System"

ICSE2—October 1976

William A. Wulf, Ralph L. London, Mary Shaw, "An Introduction to the Construction and Verification of Alphard Programs"

ICSE 3—May 1978

David Lorge Parnas, "Designing Software for Ease of Extension and Contraction"

ICSE 4—September 1979

Walter F. Tichy, "Software Development Based on Module Interconnection"

ICSE 5—March 1981

Mark Weiser, "Program Slicing"

ICSE 6—September 1982

S.J. Greenspan, J. Mylopoulos, A. Borgida, "Capturing More World Knowledge in the Requirements Specification"

ICSE 7—March 1984

D.L. Parnas, P.C. Clements, D.M. Weiss, "The Modular Structure of Complex Systems"

My observation, when studying this list (which includes two papers that I helped to write), is that *at most* one of these papers could be considered influential. There is now a vast army of people who write programs as part or all of their professional activities. Most of those professional programmers have never read any of these papers and do not use the ideas explained in these papers. The one exception, the first paper, describes a system that practicing programmers do use. In that case, it is probably true that the system was influential, but the paper itself didn't matter.

The sad fact is that most Engineers actually writing code do not come to these conferences. They also do not read *IEEE Transactions on Software Engineering*, the main journal in our field. Computer professionals do not read our literature because it does not help them to do their job better. These papers may be very good papers, they may have influenced other researchers, but they have not significantly changed the way that programs are written. There are exceptions (of course) and they "prove" the rule by being exceptions.

These are strong statements, not backed by data, but I say them with confidence. My contacts with the many people who actually produce software professionally leave me no doubt. The people who come to ICSE are exceptional; they are not typical of those who produce software. Even the practitioners who do attend tell me that they do not find much that is useful in these conferences. The people I meet in the nuclear industry, the medical equipment industry, and the telephone industry are more typical of practicing professionals. They have learned to program by learning a language, an operating system, and perhaps some application software. They believe that this is all the information that they need. They have rarely, very rarely, been exposed to the ideas that we call "Software Engineering." Few read papers such as those above or do the kinds of things suggested in ICSE papers. Those who are Engineers often have some other area of engineering as their professional identity. They think of themselves as Mechanical Engineers, Electrical Engineers or perhaps Nuclear Engineers, but not as Software Engineers. They go to conferences in those fields, not to ours. The nonengineers in the field identify themselves as "programmers" and read manuals, books on specific systems, or glossy trade magazines, but not our papers.

30.3 We Must Be Doing Something(s) Wrong!

It is easy for us to say that the practitioners are doing something wrong—they are ignoring us, "the learned researchers." However, if we want to influence the way that software is written, we have to admit that *we* are doing something wrong. Otherwise it is beyond our capability to fix the problem. I have a list of things that we, the SE research clique, should reconsider.

30.3.1 We Are Talking to Ourselves

Most of our papers are written in the tradition of science, not engineering. We write up the results of our research for readers who are also researchers, not for practitioners. We base our papers on previous work done by other researchers, and often ignore current practical problems. In many other engineering fields, the results of research are reported in "how to" papers. Our papers are "this is what I did" papers. We describe *our* work, how *our* work differs from other people's work, what we will do next, etc. This is quite appropriate for papers directed to other researchers, but it is not what is needed in papers that are intended to influence practitioners. Practitioners are far more interested in being told the basic assumptions behind the work, than in knowing how this work differs from the work by the author's cohorts in the research game.

30.3.2 We Are Ignoring Engineering Organizations

If we take the "Engineering" in "Software Engineering" seriously, we must also take the organizations operating in the rest of Engineering seriously. They should be our natural allies. They have faced similar problems in the past and have goals that are similar to ours. Just as we are trying to raise the level of professionalism in software development, the Professional Engineering societies were formed to assure a high level of professionalism in traditional Engineering fields.

What the Professional Engineering societies have done is to identify the basic information and skills that should be known by every Engineer in each of the Engineering specialties. Having identified this information, they make sure that each accredited institution presents that information to its students and that every Professional Engineer has learned those ideas. In our field, no such minimum education has been identified and there is no meaningful accreditation process. At this conference we have heard an eloquent talk by a graduate of one of our leading institutions, who reported as new, observations that were reported at least 20 years ago.

Professional Engineering societies are supported by legislation in most jurisdictions and they have well thought-out accreditation and examination procedures, etc. These mechanisms assure them the attention of their members, students, and university teachers. We need those facilities to do our job and it makes no sense to try to develop similar mechanisms on our own.

In the last few years, it has become clear to me that there is a distressing lack of communication between those in Software Engineering and those in traditional Engineering fields. Traditional Engineers do not understand our work and most of us do not understand what they do.

The Engineering societies need to understand that professional programmers require more than simple knowledge of the language, support software,

and operating system. They need to increase the level of understanding of software expected of the Professional Engineer. They also, and this is quite distinct, need to recognize the need for Professional Engineers specialized in computer systems, just as they recognized the needs for other new specialties in the past. Engineers have always underestimated the difficulty of writing software; unreliable software is the result.

However, we need to understand that a Software Engineer has to understand a lot more than the things that we talk about. Just as a Chemical Engineer must know more than Chemistry, there are many areas of traditional Engineering that are relevant to the work of Software Engineers.

Above all, we need to stop the competition between societies of Computing Professionals and Societies of Professional Engineers for the right to identify people as Software Engineers. They need us and we need them.

30.3.3 We Are Ignoring History

There has been a great deal of progress in Engineering in the last few centuries. Early bridges were built by a trial-and-error (learn-from-experience) process not unlike our present programming styles. Early power system developers did their work without the mathematical models developed by people like Steinmetz and regarded such work as irrelevant theory. Doesn't this sound familiar? However, new methods in Engineering were not adopted as a result of the kinds of things we are trying in Software Engineering. Engineers use methods if they work. They did not expect "controlled experiments" (nearly impossible in studying design methods) before using Kirchoff's laws. University teachers taught Nyquist's methods because they obviously made the Engineer's job easier. It wasn't necessary for Gauss or Kirchoff to form "User Groups" or write "Ten Commandment" style articles to sell their ideas. The ideas sold themselves. Like the proverbial "better mousetraps," good methods spread. We are putting too much effort into selling methods that are not yet ready. If they were ready, we would not have to sell them. We should be putting our efforts into testing and improving our technology, not into high pressure sales.

30.3.4 We Are Becoming "Snake Foot" Salespeople

In Chinese there are sayings about "putting feet on a snake." Snakes have evolved to function quite well without feet and adding feet would not be doing them a favor. I fear that some of the things that we are asking practitioners to do when developing software fall into that category; they don't help and may interfere with the ability of designers to work under tight constraints. What I see is a great many people who have become very talented entertainers and after-dinner speakers, but they are hucksters. They write "papers" that are sales brochures, often inaccurate and sloppy. I remember

sitting in shock as one such "expert," a Professor at a European university, sold his "do it naturally" philosophy by such statements as

- Ants invented "just-in-time" manufacturing because they have no warehouses.
- If you go to a party and drink, you can then go to your office and write a program of 5000 lines overnight and it will be correct without testing.
- The important thing is to have "team spirit" (generated by standing in a circle and holding hands); disciplined development methods don't matter.

On the other side, I have watched advocates of new methods make the claim that mathematical methods can allow you to write programs that are right the first time (without testing) while displaying slides with programs that would not pass even a simple test.

30.4 We Need to Change Something

If those are some of the things that we are doing wrong, what can we do about it? Here are a few ideas:

- Recognize that Software Engineering is a branch of Engineering. Software Engineering is treated as part of Computer Science, but that is no more true than Chemical Engineering is a branch of Chemistry. Chemical Engineering is treated as a profession quite different from that of chemistry. If you look at the curricula in that field, you will see why.
- Distinguish conferences for researchers from conferences for practitioners. Researchers need to talk to each other, but to achieve our goals we really need to address practitioners properly. Conferences for practitioners might well be modeled on the Pacific Northwest Software Quality Conference where the program committee is dominated by practitioners who want to listen, not by researchers who want to talk. ICSE would do well to follow that model. We also need to remember that the distinction is between researchers and practitioners, not between people who work for universities and people who work for industry. Lots of researchers work in industry.
- Start a dialogue with the Professional Engineering societies. Perhaps the head of the VDI (Association of German Engineers) could address the next ICSE in Berlin.
- Stop misusing the word "Engineer." When I attend ICSE conferences I am constantly reminded of a housekeeper, who used to tell me that she was going to "engineer a good dinner tonight." At ICSE 17, I stayed in a hotel that sent an "Engineer" to change a light bulb in my room. He failed on his first try! Engineering is a profession; entry into the profession requires extensive training and is carefully controlled. We cannot

make up new fields such as “Requirements Engineering” or “Reliability Engineering” whenever it suits our purposes. In fact, it is my observation that Professional Engineers rarely use “engineer” as a verb. When we understand the meaning of the term “Engineering,” we won’t offer one-term courses titled Software Engineering.

30.5 Conclusions

It would not be very gracious to “accept” an award by implying that it is not worth very much. That is not the message of my remarks. I am quite proud of having won this award twice and display the awards prominently. My point is not that the papers are not good papers, but simply that they have not been influential. I hope that others will begin to think about why we are not influencing the way that programming is done and make some further suggestions.