# Should Computer Scientists Experiment More?

16 Excuses to Avoid Experimentation

Walter F. Tichy
University of Karlsruhe, Germany

Nov. 1997

## Abstract

Computer scientists and practitioners defend the lack of experimentation with a wide range of arguments. Some arguments suggest that experimentation may be inappropriate, too difficult, useless, and even harmful. This article discusses several such arguments to illustrate the importance of experimentation for computer science.

This is a preprint of an article with the same title that appeared in IEEE Computer, 31(5), May 1998, 32–40.

Keywords: Empiricism, experiments, laboratory, scientific method.

## 1 Is computer science an experimental science?

Do computer scientists need to experiment at all? Only if we answer "yes" does it make sense to ask whether there is enough of it.

In his Allen Newell Award Lecture, Fred Brooks suggests that computer science is "not a science, but a synthetic, an engineering discipline"[2]. In an engineering field, testing theories by experiments would be misplaced. Brooks and others seem troubled by the fact that the phenomena studied by computer scientists appear manufactured — computers and programs are human creations, so we could conclude that computer science is not a natural science in the traditional sense.

I think that the engineering view of computer science is too narrow, too computer-myopic. First of all, the primary subjects of inquiry in computer science are not merely computers, but information and information processes[13]. Computers play a dominant role because they make information processes easier to model and observe. However, by no means are computers the only place where information processes occur. In fact, computer models compare poorly with information processes found in nature, say in nervous systems, in immune systems, in genetic processes, or, if you will, in the brains of programmers and computer users. The phenomena studied in computer science are much broader than those arising around computers.

Regarding "syntheticness", I prefer to think about computers and programs as models. Modeling is in the best tradition of science, because it helps us study phenomena closely. For example, for studying lasing, one needs to build a laser. Regardless of whether lasers occur in nature, building a laser does not make the phenomenon of massive stimulated emission artificial. Superheavy elements must be synthesized in the lab for study, because they are unstable and do not occur naturally, yet nobody assumes that particle physics is synthetic. Similarly, computers and software don't occur naturally, but they help us model and study information processes closely. Using these devices does not render information processes artificial.

A major difference to traditional sciences is that information is neither energy nor matter. Could this difference be the reason we see little experimentation in computer science? To answer this questions, let's look at the purpose of experiments.

# 2 Why should we experiment?

When I discuss the purpose of experiments with mathematicians, they often exclaim that experiments don't prove a thing. It is true that no amount of experimentation provides proof with absolute certainty. What then are experiments good for? We use experiments for theory testing and for exploration.

Experimentalists test theoretical predictions against reality. A community gradually accepts a theory if all known facts within its domain can be deduced from the theory, if it has withstood numerous experimental tests and if it correctly predicts new phenomena.

Nevertheless, there is always an element of suspense: To paraphrase Dijkstra, an experiment can only show the presence of bugs in a theory, not their absence. Scientists are keenly aware of this uncertainty and are therefore ready to shoot down a theory if contradicting evidence comes to light.

A good example of theory falsification in computer science is the famous Knight-and-Leveson experiment[8]. The experiment was concerned with the failure probabilities of multi-version programs. Conventional theory predicted that the failure probability of a multi-version program was the product of the failure probabilities of the individual versions. However, Knight and Leveson observed in an experiment that the failure probabilities of real multi-version programs were significantly higher. In essence, the experiment falsified the basic assumption of conventional theory, namely that faults in program versions are statistically independent.

Experiments are also used for exploring areas where theory and deductive analysis do not reach. Experiments probe the influence of assumptions, eliminate alternative explanations of phenomena, and unearth new phenomena in need of explanation. In this mode, experiments help with induction: deriving theories from observation.

Artificial neural networks are a good example of this process. After having been discarded on theoretical grounds, experiments demonstrated properties better than predicted. Researchers have now developed better theories to account for these properties.

## 2.1 Traditional scientific method isn't applicable

The fact that — in the field of computer science — the subject of inquiry is information rather than matter or energy makes no no difference to the applicability of the traditional scientific method. In order to understand the nature of information processes, computer scientists must observe phenomena, formulate explanations and theories, and test them.

There are plenty of computer science theories that haven't been tested. For instance, functional programming, object-oriented programming, and formal methods are all thought to improve programmer productivity, program quality, or both. It is surprising that none of these obviously important claims have ever been tested in a systematic way, even though they are all 30 years old and a lot of effort has been invested in developing programming languages and formal techniques.

Traditional sciences use theory test and exploration iteratively because observations help formulate new theories that can be tested later. An important requirement for any experiment, however, is repeatability. Repeatability makes sure that results can be checked independently and thus raises confidence in the results and helps eliminate errors, hoaxes, and frauds.

## 2.2 The current level of experimentation is good enough

Suggesting that the current level of experimentation doesn't need to change is based on the assumption that computer scientists, as a group, know what they are doing. This argument maintains that if we need more experiments, we'll simply do them.

But this argument is tenuous; let's look at the data. In [15], 400 papers were classified. Only those papers were considered further whose claims required empirical evaluation. For example, papers that proved theorems were excluded, because mathematical theory needs no experiment. In a random sample of all papers ACM published in 1993, the study found that of the papers with claims that would need empirical backup, 40% had none at all. In journals related to software, this fraction was 50%. The same study also analyzed a non-CS journal, Optical Engineering, and found that in this journal, the fraction of papers lacking quantitative evaluation was merely 15%.

The study by Zelkowitz and Wallace[17] found similar results. When applying consistent classification schemes, both studies report between 40% and 50% unvalidated papers in software engineering. Zelkowitz and Wallace also surveyed journals in physics, psychology, and anthropology and again found much smaller percentages of unvalidated papers there than in computer science.

Relative to other sciences, the data shows that computer scientists validate a smaller percentage of their claims. One could argue that computer science at age 50 is still young and hence a comparison with other sciences is of limited value. I disagree, because 50 years seems plenty of time for two to three generations of scientists to establish solid principles. But even on an absolute scale, I think that it is scary when half of the non-mathematical papers make unvalidated claims. Assume that each idea published without validation would have to be followed up by at least two validation studies (that's a very mild requirement). It follows trivially that no more than one third of papers published could contain unvalidated claims. The data suggests that computer scientists publish a lot of untested ideas or the ideas published are not worth testing.

I'm not advocating replacing theory and engineering by experiment, but I am advocating a better balance. I advocate balance not because it would be desirable for computer science to appear more scientific, but because of the following principal benefits:

- Experiment can help build up a reliable base of knowledge and thus reduce uncertainty about which theories, methods, and tools are adequate.
- Observation and experiment can lead to new, useful, and unexpected insights and open up whole new areas of investigation. Experimentation can push into unknown areas where engineering alone progresses only slowly, if at all.
- Experimentation can accelerate progress by quickly eliminating fruitless approaches, erroneous assumptions, and fads. It also helps orient engineering and theory into promising directions.

Conversely, when we ignore experimentation and avoid contact with reality, we hamper progress.

## 2.3 Experiments cost too much

The first line of defense against experimentation goes typically like the following: "Doing an experiment would be incredibly expensive" or "For doing this right, I would need hundreds of subjects, I would be busy for years without being able to publish, and the cost would be enormous."

To this, a hard-nosed scientist might say: "So what?" Instead of being paralyzed by cost considerations, he or she would first probe the importance of the research question. When convinced that a fundamental problem is being addressed, an experienced experimentalist would then go about planning an appropriate research program, actively look for affordable experimental techniques, and suggest intermediate steps with partial results along the way.

For a scientist, funding potential should not be the only or primary criterion for deciding what questions to ask. In the traditional sciences, there is a complex social process at work in which important questions crystallize. These become the foci of research, the breakthrough goals that open up new areas, and scientists actively search for economic ways to conduct the necessary experiments. For instance, the first experimental validation of General Relativity was tremendously expensive and barely showed the effect. The experiment was performed by Sir Issac Eddington in 1919. Eddington used a total solar eclipse to check Einstein's theory that gravity bends light when it passes near a massive star. At the time, this was a truly expensive experiment since it involved an expedition to Principe Island (West Africa) and the technology of photographic emulsions had to be pushed to its limits. However, it was important to test whether Einstein was correct or not.

Not many investigations are of a scope comparable to General Relativity, but there are many smaller, but still important questions to answer. How can such work be done economically? Since cost seems to be uppermost in everybody's mind, I will spend more space on this issue than on the others. My goal is to help the cost-conscious scientist or engineer overcome the cost barrier.

Experiments can indeed be expensive. But are all of them prohibitively expensive? I think not. There are meaningful experiments that fit the budget of small laboratories. There are also expensive experiments that are worth much more than their cost. And there is a wide spectrum in between.

**Benchmarking.** Though often criticized, benchmarks are an effective and affordable way of conducting experiments. Essentially, a benchmark is a sample of a task domain; this sample is executed by a computer or by human and computer. During execution, well-defined performance measurements are taken. Benchmarks have been used successfully in widely differing areas, including speech understanding, information retrieval, pattern recognition, software reuse, computer architecture, performance evaluation, applied numerical analysis, algorithms, data compression, logic synthesis, and robotics. A benchmark provides a level playing field for competing ideas, and assuming the benchmark is sufficiently representative, it allows repeatable and objective comparisons. At the very least, a benchmark can quickly eliminate unpromising approaches and exaggerated claims.

Constructing a benchmark is usually intensive work, but the burden can be shared among several laboratories. Once a benchmark is defined, it can be executed repeatedly at moderate cost. In practice, it is necessary to evolve benchmarks to prevent over-fitting.

Regarding benchmark tests in speech recognition, Raj Reddy writes: "Using common databases, competing models are evaluated within operational systems. The successful ideas then seem to appear magically in other systems within a few months, leading to a validation or refutation of specific mechanisms for modeling speech."[14]

In many of the examples cited above, benchmarks have caused a sudden blossoming of the area, because they made it easy to identify promising approaches and discard poor ones. I agree with Reddy that "all of experimental computer science could benefit from such disciplined experiments."

**Costly experiments.** When human subjects are involved in an experiment, the cost often goes up dramatically, while significance goes down. When are expensive experiments justified? When the implications of the insights gained outweigh the cost. Let us take an example. A significant segment of the software industry has converted from C to C++ at a substantial cost in retraining. One might well ask how solidly grounded the decision to switch to C++ was. Other than case studies (which are questionable because they don't generalize easily and may be under pressure to demonstrate desired outcomes), I'm not aware of any solid evidence showing that C++ is superior to C with respect to programmer productivity or software quality. Nor am I aware of any independent confirmation of such evidence. However, while training students in improving their personal software processes, my research group has recently observed that C++ programmers may make many more mistakes and take much longer than C programmers of comparable training – both during initial development *and* maintenance. Suppose this observation is not a fluke.[1] Then running experiments to test the fundamental tenets of object-oriented programming would be truly valuable. These experiments might save resources far in excess of their cost. The experiments might also have a lasting and positive effect on the direction of programming language research. They may not only save industry money, but also save research effort.

It is useful to check what scientists in other disciplines spend on experimentation. Everyone realizes that drug testing in medicine is extremely expensive, but only desperate patients accept poorly tested drugs and therapies. In aeronautics, we demand that airfoils be tested; expensive wind tunnels have been built for just this purpose. Numerical simulation has reduced the number of such tests, but not eliminated them. In many sciences, simulation has become an important form of experimentation, and computer science might also benefit from good simulation techniques. In biology, Wilson names the Forest Fragmentation Project in Brasilia as the most expensive biological experiment ever[16]. While clearing a large tract of the Amazon jungle, isolated patches of various sizes (1 to 1000 hectares) were left standing. The purpose was to test hypotheses regarding the relationship between habitat size and number of species remaining. And the list of experiments continues – in physics, chemistry, ecology, geology, climatology, and on and on. Any reader of Scientific American can find experiments in every issue. Computer scientists need not be afraid or ashamed of conducting large experiments when exploring important questions.

[1] Just as this article went to press, we learned that a paper by Les Hatton, "Does OO Really Match the Way We Think?" will appear in the May issue of *IEEE Software*, reporting strong evidence of the negative effects of C++.

### 2.4 Demonstrations will suffice

In his 1994 Turing Award lecture, Juris Hartmanis argues that computer science differs sufficiently from other sciences to permit different standards in experimentation, and that demonstrations can take the place of experiments[5]. I couldn't disagree more. Demos can provide proof-of-concepts in the engineering sense, or provide incentives to study a question further. Too often, however, these demos merely illustrate a potential. Demonstrations depend critically on the observers' imagination and their willingness to extrapolate; they do not normally produce solid evidence. To obtain such evidence, a careful analysis is necessary, involving experiments, data, and replication.

What would be interesting questions amenable to experimentation in the traditional sense? Here are a few examples. The programming process is poorly understood; computer scientists could therefore introduce different theories of how requirements are refined into programs and test them experimentally. Similarly, a deeper understanding of intelligence might be discovered and tested. The same applies to research in perception, questions about the quality of man-machine interfaces, or human-computer interaction in general. Also, the behavior of algorithms on typical problems or on computers with storage hierarchies cannot be predicted accurately. Better algorithm theories are needed and should be tested in the laboratory. Research in parallel systems is currently generating a number of machine models; their relative merits can only be explored experimentally. This list is certainly not exhaustive, but the examples all involve experiments in the tradition of science: They require a clear question, an experimental apparatus to test the question, data collection, interpretation, and sharing of the results.

### 2.5 There is too much noise in the way

The second line of defense against experimentation goes like this: "There are too many variables to control, and the results would be meaningless, because the effects I'm looking for are swamped by noise."

True, experimentation is difficult – for researchers in all disciplines, not just computer science. I think researchers who are invoking this excuse are looking for an easy way out.

An effective simplification for repeated experiments is benchmarking. Fortunately, benchmarking can be used for many questions in computer science. The subjective and therefore weakest part in a benchmark test is the composition of the benchmark; everything else, if properly documented, can be checked by the skeptic. Hence, the composition of the benchmark is always hotly debated (is it representative enough?), and benchmarks must evolve over time to get them closer to what one wants to test.

Experiments with human subjects involve many additional challenges. Several fields have found techniques for dealing with human variability, notably medicine and psychology. We've all heard about control groups, random assignments, placebos, pre- and post-testing, balancing, blocking, blind and double-blind studies, and the battery of statistical tests. The fact that a drug influences different people in different ways doesn't stop medical researchers from testing. And when control is impossible, then case studies, observational studies and an assortment of other investigative techniques are used. Indeed, medicine offers many important lessons on experimental design, on how to control variables and how to minimize errors. Eschewing experimentation because of difficulties is not acceptable.

## 2.6 Progress will slow

The argument here is that if everything must be backed up by experiment before publication, then the number of ideas that can be generated and discussed in the scientific community will be throttled and progress will slow.

This is not an argument to be taken lightly. In a fast-paced field such as computer science, the number of ideas being discussed is obviously important. However, experimentation need not have an adverse effect; quite the contrary.

First, increasing the ratio of papers with meaningful validation has a good chance of actually accelerating progress: Questionable ideas will be weeded out more quickly and scientists will concentrate their energies on more promising approaches.

Second, I'm confident that good conceptual papers and papers formulating new hypotheses will continue to be valued by readers and will therefore get published. It should be understood that experimental testing of these hypotheses will come later.

So it is a matter of balance once more. Presently, non-theory research rarely moves beyond the assertive state, a state characterized by such weak justification as "it seems intuitively obvious", or "it looks like a good idea", or "I tried it on a small example and it worked." Reaching a ground firmer than assertion is desirable.

## 2.7 Technology changes too fast

This concern comes up frequently in computer architecture. Trevor Mudge summarizes it: "...the rate of change in computing is so great that by the time results are confirmed they may no longer be of any relevance."[9] The same can be said about software. What good is an experiment when the duration of the experiment exceeds the useful life of the experimental subject, i.e., of a software product or tool?

If a question becomes irrelevant quickly, it is perhaps too narrow and not worth spending a lot of effort on it. But behind many questions with a short lifetime lurks a fundamental problem with a long lifetime. My first advice to scientists here is to probe the fundamental and not the ephemeral, and to learn to tell the difference. My second advice hinges on the observation that technological change often shifts or eliminates assumptions that were taken for granted. Therefore, scientists should anticipate changes in assumptions and proactively employ experiments to explore the consequences of such changes. Note that this type of work is much more demanding, and can have much higher long-term value, than merely comparing software products.

## 2.8 You'll never get it published

This is actually partly true. Some established computer science journals have difficulty finding editors and reviewers capable of evaluating empirical work. Promotion committees may be dominated by theoreticians. The experimenter is often confronted with reviewers who expect perfection and absolute certainty. However, experiments are conducted in the real world and are therefore always flawed somehow. Reviewers may also build up impossibly high barriers. I've seen demands for experiments to be conducted with hundreds of subjects over a span of many years involving several industrial projects before publication. That smaller steps are still worth publishing because they improve our understanding and raise new questions is a thinking that some are not familiar with.

However, this situation is changing. In my experience, publication of experimental results is not a problem of one chooses the right outlet. I'm on the editorial board of three journals; I review for quite a number of additional journals and have served on numerous conference committees. All non-theory journals and conferences that I've seen would greatly welcome papers reporting on solid experiments. The occasional rejection of high-quality papers not withstanding, I'm convinced that the low number of good experimental papers is a supply problem.

The funding situation for experimentation is more difficult, especially in industry/academia collaborations. However, it helps to note that experimentation may give industry a three to five year lead over the competition. For example, suppose an experiment discovered an effective way to reduce maintenance costs by using software design patterns. The industrial partner of such an experiment could exploit this result immediately, especially since the experiment prepared the groundwork for adopting the technology. Given a two-year publication time lag and various other delays (such as the results being noticed by others, let alone adopted), the industrial partner in such an experiment can exploit at least a three-year lead. Lucent Technologies estimates that it is presently benefiting from a five-year lead in software inspection methods based on a series of in-house experiments,[2] apparently despite (or because of) vigorous publication of the results.

On the negative side I fear that the "systems researcher" of old will face difficulties. Just building systems is not enough unless the system demonstrates some kind of a "first," a breakthrough. Computer science continues to be favored with such breakthroughs and we should continue to strive for them. The majority of systems researchers, however, works on incremental improvements of existing ideas. These researchers should try to become respectable experimentalists. They must articulate how their systems contributes to our knowledge. Systems come and go; insights about the concepts and phenomena underlying systems are what is needed. I have great expectations for systems researchers who use their skills in setting up interesting experiments.

[2]Larry Votta, private communication, Lucent Technolgies.

# 3 Why substitutes won't work

Can we get by with forms of validation that are weaker than experiments? It depends on what question we're asking, but here are some excuses that I find less than satisfactory.

## 3.1 Feature comparison is good enough

A frequently found model of a scientific paper is the following. The work describes a new idea, prototyped perhaps in a small system. The claim to "scientificness" is then made by feature comparison. The reports sets out a list of features and qualitatively compares older approaches with the new one, feature by feature.

I find this method satisfactory when a radically new idea or a significant breakthrough is presented, such as the first compiler for a block-structured language, the first timesharing system, the first object-oriented language, the first web browser. Unfortunately, the majority of papers published take much smaller steps forward. As computer science becomes a harder science, mere discussions of advantages and disadvantages or long feature comparisons will no longer be sufficient. Any PC magazine can provide those. A science, on the other hand, cannot live off such weak phenomenological inferences in the long run. Instead, scientists should create models, formulate hypotheses, and test them using experiments.

### 3.2 Trust your intuition

In his March 1996 column, Al Davis, the editor of IEEE Software suggests that gut feeling is enough when adopting new software technology; experimentation and data are superfluous[3]. He even suggests ignoring evidence that contradicts one's intuition.

However, instinct and personal experience sometimes lead down the wrong path and computer science is no exception. Here are some examples. For about twenty years, it was thought that meetings were essential for software reviews. However, recently Porter and Johnson found that reviews without meetings are neither substantially more nor less effective than those with meetings[11]. Meetingless reviews also cost less and cause fewer delays, which can lead to a more effective inspection process overall. Another example where observation contradicts conventional wisdom is that small software components are proportionally *less* reliable than larger ones. This observation was first reported by Basili [1] and has been confirmed by a number of disparate sources; see Hatton [6] for summaries and an explanatory theory. As mentioned, the failure probabilities of multi-version programs were incorrectly believed to be the product of the failure probabilities of the component versions. Another example is type checking in programming languages. Type checking is thought to reveal programming errors, but there are contexts when it does not help [12]. Pfleeger et al. [10] provides further discussion of the pitfalls of intuition.

What we can learn from these examples is that intuition may provide a starting point, but must be backed up by empirical evidence. Without grounding, intuition is highly questionable. What one thinks obvious may turn out to be dead wrong sometimes.

### 3.3 Trust the experts

During a recent talk at a top US university, I was about to present my data, when a colleague interrupted and suggested that I skip that part and go on to the conclusions. "We trust you." was the explanation. Flattering as that was, it shows a disturbing misunderstanding of the scientific process (or someone in a hurry). Any scientific claim is initially suspect and must be examined closely. Imagine what would have happened if physicists hadn't been skeptical about the claims by Ponds and Fleischman regarding cold fusion.

Frankly, I'm continually surprised how much the computer industry and sometimes even university teaching relies on so-called "experts" of all kinds, who fail to back up their assertions with evidence. Science, on the other hand, is built on healthy skepticism. It is a good system to carefully check results and to accept them only provisionally until they have been confirmed independently.

## 4 Problems do exist

Here are some excuses that are influenced by the quality of experiments in computer science.

### 4.1 Flawed experiments

"Experiments make unrealistic assumptions", or "The data was manipulated", or "It is impossible to quantify the variable of interest," are some of the criticisms. There are many more potential flaws: Experimenters may pick irrelevant questions, may neglect to provide enough detail for repeating experiments, may be nonchalant about control, may not validate observations, forget to bound errors, use inappropriate measurements, over-interpret their results, produce results that do not generalize, etc.

Good examples of solid experimentation in computer science are rare. And there will always be questionable, even bad experiments. However, the conclusion from this observation is not to discard the concept of experimentation. We should keep in mind that other scientific fields have been faced with bad experiments, even frauds. But the scientific process on the whole has been self-correcting. Bad ideas, errors, and downright hoaxes have been weeded out, sometimes promptly (see cold fusion) sometimes belatedly (see the Piltdown man).[3]

We can be sure of one thing, though: If scientists overlook experimentation or neglect reexamining others' claims, an important source of self-correction will be cut off and the field may drift into the wrong direction.

## 4.2 Competing theories

A science is most exciting when there are two or more strong, competing theories. When a new, major theory replaces an older one, one speaks of a paradigm shift, while the stable periods in between are called "normal science". Physics provides interesting examples of paradigm shifts.

There are a few competing theories in computer science, none of them earth-shaking. The physical symbol system theory vs. the knowledge processing theory in AI is one of them. These two theories attempt to explain intelligence. The weak reasoning methods of the first theory have gradually given way, or have been coupled with, knowledge bases [4]. Other examples include symbolic vs. subsymbolic processing, RISC vs. CISC, the various models for predicting the performance of (parallel) computers, and the competition among programming language families (logic, functional, imperative, object-oriented, rule-based, constraint-based). Another important example is algorithms theory. The present theory has many drawbacks; in particular, it does not account for the behavior of algorithms on "typical" problems[7]. A more accurate theory that applies to modern computers would be valuable.

A prerequisite for competition among theories is falsifiability. Unfortunately, computer science theorists rarely produce falsifiable theories; they tend to pursue mathematical theories that are disconnected from the real world.[4] Thus, it has largely fallen to experimentalists and engineers to formulate falsifiable theories.

While computer science is perhaps too young to have brought forth grand theories, my greatest fear is that the lack of such theories might be caused by a lack of experimentation. If scientists neglect experiment and observation, they'll have difficulties discovering new and interesting phenomena worthy of better theories.

[3] Piltdown man are fossil remains found in England in 1912. The fossils were thought to be a species of prehistoric man and generated scholarly controversy that lasted about 40 years. In 1954, intense re-examination showed the remains to be fraudulent. The fossils consisted of skillfully disguised fragments of a quite modern human cranium (50,000 years old), the jaw and teeth of an orangutan, and the tooth of a chimpanzee.

[4] In Ch. 9 of *The Quark and the Jaguar*, W.H. Freeman (1994), Gell-Mann provides a lucid discussion of the relationship between mathematics and science. If science is concerned with describing nature and its laws, then mathematics is not a science, because it is not concerned with nature; it is concerned with the logical consequences of certain assumptions. On the other hand, mathematics can also be viewed as the rigorous study of what might have been, i.e., the study of hypothetical worlds, including the real one. In that case, mathematics is the most fundamental science of all.

### 4.3 Soft science

"Soft science" means that experimental results cannot be reproduced. Experiments with human subjects are not necessarily soft. There are stacks of books on how to conduct experiments with humans. Experimental computer scientists can learn the relevant techniques or ask for help. The side-bar provides some starting points.

### 4.4 Misuse

The argument goes along the following lines: "Give the managers or funding agencies a single figure of merit and they will use it blindly to promote or eliminate the wrong research."

I think this is a red herring. Good managers, good scientists, and good engineers all know better than to rely on a single figure of merit. Second, there is a much greater danger in relying on intuition and expert assertion alone. Keeping decision makers in the dark has an overwhelmingly higher damage potential than informing them to the best of ones abilities.

## 5 Conclusion

Experimentation is central to the scientific process. Only experiments test theories. Only experiments can explore critical factors and bring new phenomena to light so theories can be formulated in the first place. Without experiments in the tradition of science, computer science is in danger of drying up and becoming an auxiliary discipline. The current pressure to concentrate on applications is the writing on the wall.

I have no doubt that computer science is a fundamental science of great intellectual depth and importance. Much has already been achieved. Computer technology has changed society, and computer science is in the process of deeply affecting the weltanschauung of the general public. There is also much evidence suggesting that the scientific method applies. As computer science leaves adolescence behind, I hope to see the experimental branch of this discipline flourish.

*Acknowledgments* This essay has benefited tremendously from numerous discussions with colleagues. I'm especially grateful for thought-provoking comments by Les Hatton, Ernst Heinz, James Hunt, Paul Lukowicz, Anneliese v. Mayrhauser, David Notkin, Shari Lawrence Pfleeger, Adam Porter, Lutz Prechelt, and Larry Votta.

## References

[1] Victor R. Basili and B.T. Perricone. Software errors and complexity: An empirical investigation. *Communications of the ACM*, 27(1):42–52, January 1984.

[2] Frederick P. Brooks. Toolsmith II. *Communications of the ACM*, 39(3):61–68, March 1996.

[3] Al Davis. From the editor. *IEEE Software*, 13(2):4–7, March 1996.

[4] Edward A. Feigenbaum. How the What becomes the How. *Communications of the ACM*, 39(5):97–104, May 1996.

[5] Juris Hartmanis. Turing award lecture: On computational complexity and the nature of computer science. *Communications of the ACM*, 37(10):37–43, October 1994.

[6] Les Hatton. Reexamining the fault density–component size connection. *IEEE Software*, 14(2):89–97, 1997.

[7] John N. Hooker. Needed: An empirical science of algorithms. *Operations Research*, 42(2):201–212, March 1994.

[8] John C. Knight and Nancy G. Leveson. An experimental evaluation of the assumption of independence in multiversion programming. *IEEE Transactions on Software Engineering*, SE-12(1):96–109, January 1986.

[9] Trevor Mudge. Report on the panel: How can computer architecture researchers avoid becoming the society for irreproducible results? *Computer Architecture News*, 24(1):1–5, March 1996.

[10] Shari Lawrence Pfleeger, Victor Basili, Lionel Briand, and Khaled El-Emam. Rebuttal to March 96 editorial. *IEEE Software*, 13(4), July 1996.

[11] Adam A. Porter and P.M. Johnson. Assessing software review meetings: Results of a comparative analysis of two experimental studies. *IEEE Transactions on Software Engineering*, 23(3):129–145, March 1997.

[12] Lutz Prechelt and Walter F. Tichy. An experiment to assess the benefits of inter-module type checking. In *Proc. Third Intl. Software Metrics Symposium*, pages 112–119, Berlin, March 1996. IEEE Computer Society Press.

[13] Anthony Ralston and Edwin D. Reilly. *Encyclopedia of Computer Science, Third Edition*. Van Nostrand Reinhold, 1993.

[14] Raj Reddy. To dream the possible dream. *Communications of the ACM*, 39(5):105–112, May 1996.

[15] Walter F. Tichy, Paul Lukowicz, Lutz Prechelt, and Ernst A. Heinz. Experimental evaluation in computer science: A quantitative study. *The Journal of Systems and Software*, 28(1):1–18, January 1995.

[16] Edward O. Wilson. *The Diversity of Life*. Harvard University Press, 1992.

[17] Marvin V. Zelkowitz and Dolores Wallace. Experimental models for validating computer technology. *IEEE Computer*, 31(5), May 1998.