What is Experimental Computer Science?

Peter J. Denning

The Feldman Report has stimulated extensive discussion about experimental computer science.[1] This report suggested that, without a large-scale rejuvenation of experimental computer science at universities, much basic research in computer science will cease. It suggested further that declining experimental research has produced declining innovation in computing.

This discussion is even now causing action. In government, we find sympathy for allocating a greater share of research funds to computer science. We find new NSF programs such as the Industry/University Cooperative program, the experimental research center program, and the new researcher support program. We find growing interagency cooperation such as between NSF and ARPA. In industry, we find efforts to cooperate with universities and foster more experimental research and curricula; these efforts include modest cash grants and equipment discounts for selected departments.

Despite positive talk and positive action, experts differ in their definitions of "experimental computer science." Reviewers of software engineering proposals submitted to NSF often differ widely in their assessments of the quality of the same proposal. (This discourages authors, who don't know how to revise their proposals.)

Some have argued that computer science is in flux—making a transition from theoretical to experimental science—hence, no precise definition of "experimental computer science" is yet available.

I reject this argument. Resources are being diverted to computer science that could have been allocated to other sciences. If, after a reasonable trial period, computer science has not produced good experimental work when judged by *traditional* standards, these resources will be allocated elsewhere. We should judge ourselves by the same criteria the rest of the world will judge us.

Hypotheses, Apparatus, and Tests

Science classifies knowledge. Experimental science classifies knowledge derived from observations. The experimenters construct apparatus to measure a given phenomenon; they then attempt to collect data sufficient to support their hypotheses about that phenomenon. Hypothesis Testing, a branch of Statistics, is an important tool. The experimental apparatus is not usually a subject of research.

The experimental apparatus may be a real system or subsystem—for example, the program implementing a hashing algorithm, an interactive computer system, or a paging algorithm. But the apparatus can also be a model—for example, a simulator of VM/370 or a queueing network.

The hypothesis may concern a law of nature—for example, one can test whether a hashing algorithm's average search time is a small constant independent of the table size by measuring a large number of retrievals. The hypothesis may concern characteristics of people—for example, one can test whether interactive computing improves programmer productivity by comparing the ability of control groups to solve problems with and without interactive terminals. The hypothesis may deal with design principles of computers—for example, one can determine which paging algorithm is best by controlled experiments with different algorithms managing the same workload. The hypothesis may concern the quality of models—for example, one can systematically measure the errors between response-time estimates calculated by a queueing network model and the real response times measured in a computer system.

The key ideas here are an apparatus to be measured, a hypothesis to be tested, and systematic analysis of

[1] See: "Rejuvenating experimental computer science," *Comm. ACM*, September 1979; the ACM Executive Committee position in the same issue; and my letter in the ACM Forum in *Comm. ACM* of January 1980.

the data (to see whether it supports the hypothesis). There is considerable flexibility in the types of hypotheses and apparatuses that may be used.

Some of the most renowned experimental work has been the construction of systems—for example, M.I.T.'s Multics or ARPA's communication network. The proposers of these projects did not, however, propose merely to build the system and "see what happens."[2] They stated a hypothesis that required an ambitious project to test. The basic hypothesis of Multics was that an interactive system with sharable files and a high degree of security, a "computer utility," would significantly increase productivity and creativity of programmers. The basic hypothesis of the ARPAnet is that long distance message and file transfer services would significantly increase scientific productivity by permitting critical masses of researchers to form across long distances. These hypotheses have been confirmed by observing these systems after they were completed.

No scientific discipline can be productive in the long term if its experimenters merely build components. Building a complex apparatus in the lab is a technological effort that may require great skill. But unless the apparatus is used to obtain significant new knowledge, the research is judged not to be substantive and is soon forgotten. This is why scientists from other disciplines regard machine construction as engineering, not science.

Repetition, Tinkering, and Resources

I have encountered three misconceptions of "experimental computer science."

Misconception 1: *It is not novel to repeat an experiment.*

Many proposals are rejected because a reviewer said: "That's already been done." Many others have never been submitted because the proposer feared such a response. How untraditional! It is the custom in Physics, Chemistry, Biology, and Medicine for different groups to repeat an important experiment under slightly different conditions or with slightly different methods—to see if it can be independently corroborated. Results are not accepted by the community unless they have been independently verified. The reviewers of such proposals must evaluate whether the ultimate value of the result justifies the cost of repeating the experiment.

Misconception 2: *Mathematics is the antithesis of experiment.*

This misconception is manifest in common phrases like "theory *versus* practice" and "mathematicians *versus* practitioners." It is manifest in statements like, "Once a theorem is proved, there's no point in reproving it," and, "Once a thing is built, there's no point in theorizing about it."

The whole point of science is to discover which *ideas* are important. Experiments are essential: to understand *ideas* and convince others of their value. Once an *idea* is assimilated by the community, the experiments behind it may be forgotten. This is true even of mathematics: Results are reproved to improve understanding of the underlying principles, the best theorems have many proofs, and social processes with empirical overtones help identify and simplify the best concepts.[3]

History shows clearly that science and mathematics are complementary. People *like* to theorize about important ideas!

Misconception 3: *Tinkering is experimental science.*

(We use the word "hacking," rather than "tinkering," in our field.) Unless it seeks to support a hypothesis, tinkering is not science. It is not *science* to assemble parts to "see what happens." Undirected work wanders aimlessly, finding interesting results only by accident; it produces "researchers" with spotty and erratic records. Directed work, systematic testing, and dogged scientific perseverance have traditionally characterized the most productive researchers. "Hacking" is not experimental computer science: It may improve the personal knowledge of the hacker, but it does not contribute to our sum of knowledge.

I do not mean to suggest that tinkering has no role. Indeed, many interesting results have been discovered serendipitously. But many more have been discovered by systematic, persistent workers. Tinkering is the exception, not the rule, in productive research.

On account of this misconception there is a serious risk that funds being allocated for experimental research will be used merely for hacking. Combined with the previous misconception, there is a further risk of discouraging conceptual work. Tinkering is no substitute for thinking.

In our quest for better experimental science we must not forget the small inventor. Many good ideas have been contributed by lone researchers working in modest settings.

* * * *

Let me close by quoting from a letter by Lewis Branscomb, Vice-President and Chief Scientist at IBM, in his comments to NSF on the Feldman Report:[4]

"One further comment on my use of the words 'experimental' and 'theoretical'. Much of the exciting work that needs to be done is conceptual in character and is often referred to as architecture in the field. To make this work real and useful, architectural ideas need to be reduced to practice and measurements made. The establishment of a quantitative base for making technical judgments in the field of computer science deserves the highest possible priority in my personal opinion. My endorsement of the emphasis given in the [Feldman Report] is an endorsement of conceptualization and documented principles of architecture [which are] design tested in a quantitative and functionally useful way. I do not endorse the temptation of some (in the weaker departments) to purely empirical experimentation with microprocessors and other systems from which familiarity with the technology is gained but cumulative knowledge is not."

Let us employ traditional measures when assessing experimental computer science. Let us always have a clear plan for testing a clear hypothesis. Let us not call "hacking" science. These are the criteria by which the rest of the world will evaluate our field's experimental work. If we do not live up to the traditional standards of science, there will come a time when no one takes us seriously.

[2] In some disciplines, most notably recombinant DNA, scientists avoid combining ingredients without carefully evaluating probable outcomes. Mistakes can be costly.

[3] See R. DeMillo, R. Lipton, and A. Perlis, "Social processes and proofs of theorems and programs," *Comm. ACM* (May 1979), 271–280.

[4] Source: NSF Computer Science Program Report, Vol. 4, No. 2, April 1980, p. 24.