What does the future hold for theoretical computer science?

Donald Sannella*
Laboratory for Foundations of Computer Science
Edinburgh University

Abstract Prospects for research in theoretical computer science are discussed. The maintenance of a genuine link between theory and practice is seen as key to the future health of both.

1 Introduction

Worries about the future of research in theoretical computer science are commonplace nowadays. Funding agencies seem less inclined than hitherto to fund theoretical work; attendance at theoretical conferences is down; jobs for theorists are scarce; and practitioners seem to take little notice of the results of theoretical research. See e.g. [AJK+96] for a US-oriented analysis of the situation. TAPSOFT was founded at or near the height of enthusiasm for formal methods in software development [EM95]. It is now generally accepted that many of the claims made in those days were overly optimistic, although great strides have been made and formal methods work is having a significant and increasing impact on practice.

In marked contrast to this air of gloom and doom is the continuing and accelerating boom in computing practice. Computer science and information technology are of ever-increasing importance to society. Advances in computing are seen as key to future developments in all areas, including virtually all sectors of industry, see e.g. [DTI96]. Very many problems require solution; some of these problems are "merely" technological ones while others are conceptual ones that may well yield to the insights offered by theoretical work.

If theoretical computer science is truly relevant to computing, then there is reason to believe that difficulties with funding etc. are a temporary phenomenon and that the value of this work is evident in the longer term. Of course, this does not mean that there is no need to justify why the work needs to be done, but at least the struggle is winnable and is worth winning. If on the other hand it is not truly relevant then it needs to be justified on different grounds.

I would like to make three main points. First, I believe that the link between theory and practice is important for the health of both. When there is no genuine link it is dishonest and ultimately counter-productive to pretend that there is one. Second, I suggest that relevance is not the same as direct applicability: the way in

*dts@dcs.ed.ac.uk; supported by an EPSRC Advanced Fellowship.

which a deep understanding of some computing phenomenon translates to practice is more subtle than that. Third, I draw an analogy between natural science and computer science and conclude that benefits for both theory and practice could be derived from experiments in the application of theory to the design and analysis of non-trivial systems.

Here at TAPSOFT we are concerned with a particular aspect of computing, namely software science. TAPSOFT will be succeeded in 1998 and subsequent years by ETAPS, the European Joint Conferences on Theory and Practice of Software. One goal of ETAPS is to strengthen the link between theory and practice in software science while giving space to both.

2 Theory and practice

I have argued above that the relevance of theory to computing practice is a key issue. This is not to say that relevance is the only or even the main yardstick to measure the value of a piece of theoretical work. Neither do I mean that theory that is not so relevant has no value. My point is merely that theory that is clearly about the practice of computing derives its value partly by reference to the importance of that practice, and this gives such work a certain moral claim for support.

Theory that is not about the practice of computing needs to be motivated without reference to that practice. This point may seem obvious, but researchers writing grant proposals sometimes lose sight of it! False claims of relevance are dangerous and give all of theoretical computer science a bad name. That said, please note that the word "relevant" means different things to different people, and I argue in the next section for a rather generous interpretation.

I will not waste space arguing to this audience that research on theory that is relevant to computing practice is often of benefit to that practice. The benefit is not always as immediate as practitioners seem to expect it to be, but there are plenty of examples that demonstrate a genuine payoff.

In conducting theoretical research, it is necessary to keep an eye on what the theory claims to be a theory *of*. This will typically be some sub-domain of computing practice which will tend to move with the times in a way that is relatively independent of work on its theoretical underpinnings. If theory advances entirely without regard for the practice that it is attempting to underpin, there is always a danger that it may become a theory of *nothing*. This may happen for at least two reasons:

1. Theory develops its own agenda. The most interesting theoretical problems may turn out to arise only in a special case of a development that itself has only minor importance in practice rather than being central to the practical problem that prompted the original investigation.

2. Theory tends to lag behind advances in practice. The problems that are attacked are the problems that were of importance at some point in the past. These problems may or may not be of current importance.

When a theory degenerates to the point where it is a theory of nothing, it is ultimately doomed in spite of the fact that it may take a long time to wither away. Examples from history (e.g. *angelology*, the study of different categories of angels,

including the calculation of how many angels can dance on the head of a pin, taking wingspan and other factors into account) invite ridicule: how could anybody have wasted time doing that! But no doubt at the time scholars who had devoted their careers to such subjects viewed them with the same seriousness as people who work on X nowadays. (I would not dare to say what X is!)

The academic reward structure tends to reinforce the inertia that is inherent in the system while giving little credit to developments that are genuinely useful. Difficult results are admired, particularly when the problem has withstood attack for some considerable time, while applicable theory does not receive much attention. "Seminal" work which opens up a new avenue of investigation is greatly respected while the opposite (is there even a word for it?) which closes off an avenue, "harvests" the results, and consolidates what remains, is rare. Work that shows how a theoretical result can be applied to give some practical benefit is not highly regarded.

Such things do not change by decree. Theoretical computer science is surely not the first subject in which this problem has arisen so perhaps a dramatic change is unnecessary. Nonetheless, the problem deserves notice and for the health of the subject it is necessary to ensure that work that is "only" useful continues to be done.

3 Relevance and applicability

Sometimes the link between theory and practice is quite direct: for example, some parts of formal language theory are directly relevant to the construction of parsers for programming languages. But this is not always the case, and to expect otherwise seems naive. Colleagues have complained that this over-simplified model is the one that is adopted by certain funding agencies, where (perhaps this is a caricature) work on the theory of multimedia is promoted but work on models of type theory is not.

In Edinburgh we have recently been thinking in terms of a four-level model known locally as the "Fourman hierarchy". The four levels are:

1. *Products and services*, e.g. an air traffic control system.
2. *Generic issues*, e.g. security.
3. *Research themes*, e.g. concurrent systems.
4. *Research achievements*, e.g. a proof that $P = NP$.

Computing practice is on level 1, and our day-to-day research work is on level 4. There is a many-to-many relation between each level and the next. For example: research on concurrent systems has a bearing on a number of generic issues, for example security and performance; and security requires input from research on concurrent systems and programming languages and from other areas as well. This relation is not fixed: in particular, new links can arise in unpredictable ways as a result of new developments at each level. The specific items that appear on each level also change with time, with old items disappearing (perhaps because a problem has been solved once and for all) and new items appearing (perhaps because changes in technology have led to new possibilities that in turn raise new problems).

This model only tells part of the story. For example, it does not really reflect the difference between research that aims to understand fundamental concepts and research that aims to solve problems or provide methods, or more generally the way

that research in certain areas underpins research in other areas. But it might help in understanding and justifying the way that theory can have an indirect bearing on practice, and suggest ways in which the link (if there is one) can be made more explicit.

The fact that a piece of research is not relevant to current practice does not mean that it will not be relevant to future practice. Still, to make a convincing case for relevance to future practice it is worth speculating about future issues that might be addressed by the research, and about the future products and services that solutions to these issues might someday enable. Such things are hard and sometimes impossible to predict reliably, but this does not mean that it is pointless to try.

4 The importance of experiment

We are all familiar with the *scientific method* as used in physics and other natural sciences. One develops a theory that explains some aspect of reality, and then conducts experiments in order to provide evidence that the theory is right or demonstrate that it is wrong. If an experiment shows that a theory is wrong then often it is possible to modify the theory to make it fit the experimental evidence rather than simply discarding it. In this case the outcome of the experiment which revealed the problem might supply hints for ways of modifying the theory.

The experimental method can be applied to theories in computer science too. (A difference is that the "reality" that is being studied is often man-made, so when an experiment shows that something is wrong it is sometimes possible to change reality to fit the theory!) In particular, when the theory is about design or analysis of systems, as is much of the theory that is seen at TAPSOFT, conducting an experiment involves attempting to build or analyze some non-trivial system using the ideas in the theory. Such an experiment might supply evidence that the theory "fits" practice and/or that it makes correct and interesting predictions about the behaviour of systems. Or it might bring to light some deficiency in the theory, for instance that the simplifying assumptions made are so strong that the theory cannot be applied to any system of interest or that the methods provided are so cumbersome that they cannot be used. What exactly goes wrong provides strong hints about ways in which the theory can be revised to make the next experiment more successful.

In traditional sciences, a theory that has not been subjected to experiment is regarded with considerable skepticism. Although the people who do the experiments are often different from the people who invent the theories, neither activity makes much sense without the other. Vast sums of money are invested in experimental apparatus (CERN is just one example) and scientists win Nobel prizes both for inventing theories and for performing experiments.

In contrast, most theory in computer science is never subjected to experiment. Why should unvalidated theories be taken more seriously in computer science — as they clearly are — than in physics? The only reason I can see is that we have somewhat more intuition about the man-made systems that our theories are about than we do about the behaviour of sub-atomic particles or black holes. Nevertheless, it seems clear that a theory that has been validated is worth much more than a theory that has nothing but intuition in its favour. Moreover, the feedback that is obtained from experimental application of a theory can be a tremendously valuable stimulus

to further theoretical development. By trying out a theory, it becomes clear whether or not the concepts being studied are the important ones as well as whether or not what the theory says about these is valid.

These ideas are explained more eloquently and at greater length by Robin Milner in [Mil86]. They constitute an important part of the basis upon which the Laboratory for Foundations of Computer Science (LFCS), which Milner inaugurated and which I currently direct, was founded.

5 TAPSOFT and ETAPS

Discussions aiming at a consolidation of the European conference situation in the area of software science took place in public and private around the time of TAPSOFT'95. On the basis of a broad consensus, it was decided to establish a single annual federated spring conference in the slot currently occupied by TAPSOFT and CAAP/ESOP/CC, comprising a number of existing and new conferences and covering a spectrum from theory to practice. The first instance of the European Joint Conferences on Theory and Practice of Software (ETAPS) will take place next year in Lisbon and will comprise five conferences: FoSSaCS, FASE, ESOP, CC and TACAS.

ETAPS is a natural development from TAPSOFT. One difference is a change of focus on the theory end of the spectrum, with FoSSaCS replacing CAAP. Another difference is that events covering a broader spread of practice are included. Finally, its format is open-ended, allowing it to grow and evolve as time goes by. I hope that ETAPS will provide a forum within which new links between work on theory and practice of software science can be forged and existing links can be strengthened.

Acknowledgements I am grateful to: Robin Milner for the ideas in Section 4; Samson Abramsky, Alan Bundy, Mike Fourman and others for their contributions to discussions from which some of the ideas in Sections 2 and 3 are derived; and, all those involved in the genesis of ETAPS.

References

[AJK+96] A. Aho, D. Johnson, R. Karp, S.R. Kosaraju, C. McGeoch, C. Papadimitriou and P. Pevzner. Emerging opportunities for theoretical computer science. `ftp://ftp.cs.washington.edu/tr/1996/03/UW-CSE-96-03-03.PS.Z` (1996).

[DTI96] Department of Trade and Industry. Software for IT and telecoms. Technology Foresight summary, Office of Science and Technology. `http://www.open.gov.uk/ost/foresigh/wn2.htm` (1996).

[EM95] H. Ehrig and B. Mahr. A decade of TAPSOFT: aspects of progress and prospects in theory and practice of software development. *Proc. TAPSOFT'95*, Aarhus. Springer LNCS 915, 3–24 (1995).

[Mil86] R. Milner. Is computing an experimental science? LFCS report ECS-LFCS-86-1 (1986). Reprinted in *Journal of Information Technology* 2:58–66 (1987).