See also What is a PhD in HCI? As well as Aaron Sloman's Notes on Presenting Theses.

What is Research in Computing Science?

Chris Johnson

Glasgow Interactive Systems Group (GIST), Department of Computer Science, Glasgow University, Glasgow, G12 8QQ.

Tel: +44 141 330 6053
Fax: +44 141 330 4913
EMail:johnson@dcs.gla.ac.uk

This paper argues that the expanding scope of `computing science' makes it difficult to sustain traditional scientific and engineering models of research. In particular, recent work in formal methods has abandoned the traditional empirical methods. Similarly, research in requirements engineering and human computer interaction has challenged the proponents of formal methods. These tensions stem from the fact that `Computing Science' is a misnoma. Topics that are currently considered part of the discipline of computing science are technology rather than theory driven. This creates problems if academic departments are to impose scientific criteria during the assessment of PhDs. It is, therefore, important that people ask themselves `What is Research in Computing Science' before starting on a higher degree.

This paper is intended as a high level introduction for first year research students or students on an advanced MSc course. It should be read in conjunction with Basic Research Skills in Computing Science

Keywords: research skills, computing science.

1. Introduction

Good research practice suggests that we should begin by defining our terms. The Oxford Concise dictionary defines research as:

- **research.** 1.a. the systematic investigation into and study of materials, sources, etc, in order to establish facts and reach new conclusions. b. an endeavour to discover new or collate old facts etc by the scientific study of a subject or by a course of critical investigation.

This definition is useful because it immediately focuses upon the systematic nature of research. In other words, the very meaning of the term implies a research method. These methods or systems essentially provide a model or structure for logical argument.

1.1 The Dialectic of Research

The highest level of logical argument can be seen in the structure of debate within a particular field. Each contribution to that debate falls into one of three categories:

- **thesis**
  This presents the original statement of an idea. However, very few research contributions can claim total originality. Most borrow ideas from previous work, even if that research has been

conducted in another discipline.

- **antithesis**
  This presents an argument to challenge a previous thesis. Typically, this argument may draw upon new sources of evidence and is typically of progress within a field.

- **synthesis**
  This seeks to form a new argument from existing sources. Typically, a synthesis might resolve the apparent contradiction between a thesis and an antithesis.

A good example of this form of dialetic is provided by the debate over prototyping. For example, some authors have argued that prototypes provide a useful means of generating and evaluating new designs early in the development process (thesis), (Fuchs, 1992). Others have presented evidence against this hypothesis by suggesting that clients often choose features of the prototyping environment without considering possible alternatives (antithesis) (Hayes and Jones, 1989). A third group of researchers have, therefore, developed techniques that are intended to reduce bias towards features of prototyping environments (synthesis) (Gravell and Henderson, 1996). Research in a field progresses through the application of methods to prove, refute and reassess arguments in this manner.

2. Models of Argument

A more detailed level of logical argument can be seen in the structures of discourse that are used to support individual works of thesis, antithesis or synthesis.

2.1 Proof by Demonstration?

Perhaps the most intuitively pursuasive model for research is to build something and then let that artefact stand as an example for a more general class of solutions. There are numerous examples of this approach being taken within the field of computer science. It is possible to argue that the problems of implementing multi-user operating systems were solved more through the implementation and growth of UNIX than through a more measured process of scientific enquiry.

However, there are many reasons why this approach is an unsatisfactory model for research. The main objection is that it carries high risks. For example, the artefact may fail long before we learn anything about the conclusion that we are seeking to support. Indeed, it is often the case that this approach ignores the formation of any clear hypothesis or conclusion until after the artefact is built. This may lead the artefact to become more important to the researcher than the ideas that it is intended to establish.

The lack of a clear hypothesis need not be the barrier that it might seem. The proof by demonstration approach has much in common with current engineering practice. Iterative refinement can be used to move an implementation gradually towards some desired solution. The evidence elicited during previous failed attempts can be used to better define the goal of the research as the work progresses. The key problem here is that the iterative development of an artefact, in turn, requires a method or structure. Engineers need to carefully plan ways in which the faults found in one iteration can be fed back into subsequent development. This is, typically, done through testing techniques that are based upon other models of scientific argument. This close relationship between engineering and scientific method should not be surprising:

- **engineering** n. an application of science to the design, building and use of machines, construction etc. (The Oxford Concise Dictionary).

2.2 Empiricism

The Western empirical tradition can be seen as an attempt to avoid the undirected interpretation of artefacts. It has produced the most dominant research model since the seventeenth century. It can be summarised by the following stages:

- **Hypothesis generation**
  This explicitly identifies the ideas that are to be tested by the research.

- **Method identification**
  This explicitly identifies the techniques that will be used in order to establish the hypothesis. This is critical because it must be possible for one's peers to review and criticise the appropriateness of the methods that you have chosen. The ability to *repeat* an experiment is a key feature of strong empirical research.

- **Result compilation**
  This presents and compiles the results that have been gathered from following the method. An important concept here is that of statistical significance; whether or not the observed results could be due to chance rather than an observable effect.

- **Conclusion**
  Finally, the conclusions are stated either as supporting the hypothesis or rejecting it. In the case that results do not support a hypothesis, it is important always to remember that this may be due to a weakness in the method. Conversely, successful results might be based upon incorrect assumptions. Hence, it is vital that all details of a method are made available to peer review.

This approach has been used to support many different aspects of research within Computing Science. For example, Boehm, Gray and Seewaldt (1984) used it to compare the effectiveness of specification and prototyping techniques for software engineering. Others have used it to compare the efficiency of searching and sorting algorithms. Researchers in Information Retrieval have even developed standard methods which include well known test sets to establish performance gains from new search engines.

There are many problems with the standard approach to scientific empiricism when applied to computing science. The principle objection is that many aspects of computing defy the use of probabilistic measures when analysing the results of empirical tests. For example, many statistical measures rely upon independence between each test of a hypothesis. Such techniques clearly cannot be used when attempting to measure the performance of any system that attempts to optimise its performance over time; this rules out load balancing algorithms etc. Secondly, it can be difficult to impose standard experimental conditions upon the products of computer science. For example, if a program behaves in one way under one set of operating conditions then there is no guarantee that it will behave in the same way under another set of conditions. These conditions might go down to the level of alpha particles hitting memory chips. Thirdly, it can be difficult to generalise the results of tightly controlled empirical experiments. For example, just because a user finds a system easy to use in a lab-based evaluation, there is no guarantee that another user will be able to use that product amidst the distractions of their everyday working environment. Finally, it is difficult to determine when a sufficient number of trials have been conducted to support many hypotheses. For example, any attempt to prove that a program always satisfies some property will be almost certainly doomed to failure using standard experimental techniques, The number of potential execution paths through even simple code makes it impossible to test properties against every possible execution path.

2.3 Mathematical Proof

The dissatisfaction with empirical testing techniques has led many in the computing science research community to investigate other means of structuring arguments in support of particular conclusions. In the United Kingdom, much of this work has focussed upon argumentation techniques that were originally developed to model human discourse and thought within the field of philosophy. For example, Burrows, Abadi and Needham (1990) adopted this approach to reason about the correctness of network authentication protocols. The central idea in this work is that mathematics can be used to set up a system of rules about valid and invalid inferences. These rules can then be applied to work out whether or not a conclusion is a valid inference given some initial statements about a program or some hardware.

The field of mathematical reasoning is a research area in its own right. It is, however, possible to identify two different approaches to the use of formal proof as a research technique in computing science:

- **the argument of verification.**
  This attempts to establish that some good property will hold in a given system. The classical approach is to allow a human to interactively guide a theorem proving system towards some sequence of proof steps that support the conclusion. The problem here is that if the human cannot construct a proof, this does not imply that the conclusion is invalid. Simply that they have failed to prove it. Another person might be capable of constructing the necessary mathematical argument.

- **the argument of refutation.**
  Rather than attempting to prove the correctness of an argument, this approach attempts to refute it. Typically, this is done by setting up a description of the intended system behaviour. Model checking tools then automatically explore the state space of the proposed application in an attempt to find a situation in which the desired conclusion does not hold.

The attractions of mathematical proof techniques are very strong. They provide a coherent framework for analysing research questions in computing science. They also explicitly state the criteria for valid inferences, as well as the environmental conditions, that restrict the scope and applicability of the reasoning process. There are, however, many problems that limit the utility of this approach as a general research tool.

The first is that incredible care needs to be made over the interpretation of results from mathematical proof. Formal methods are nothing more than a system of argumentation and mistakes are to be expected. Problems arise because mistakes can be very difficult to detect given the complex nature of the mathematics that are often used. Recall that a central feature of the empirical approach was that open peer review should be used to check that your method is correct.

The second problem with formal reasoning is that their scope is limited. Interactive and time critical systems pose specially challenges for the application of mathematics. These issues are being addressed but many problems remain.

The third problem relates to the cost of applying formal techniques. It takes a long time to acquire the necessary skills. Similarly, it can take several months to conduct relatively simple proofs for medium to large scale applications.

Finally, it can be argued that there is inadequate discussion about the failures of formal methods. Again, it is important to recall that a failure to prove a hypothesis was a valuable result for empirical techniques. Exaggerated claims have been made for formal reasoning, typically not by the researchers themselves, and many of these claims have been falsified. As a result errors in the application of mathematical reasoning can be seen as a source of shame rather than a learning opportunity for one's colleagues and peers.

2.4 Hermeneutics

Formal proof techniques rely upon the development of a mathematical model of the artifact that is being created. This raises important questions about the relationship between that model and the reality which it is intended to represent. For eexample, if a model omits some critical aspect of a program's environment then it may be proven to be safe but may well fail when implemented. The distance between mathemctaical models and reality is, popularly, know as the formality gap. Hermeneutics provide an alternative that addresses this problem. Hermeneutic research methods have been pioneered within the field of sociology. The term itself means:

- `adj. concerning interpretation, esp. of Scripture or literary texts'. (The Oxford Concise Dictionary).

In practice, these approaches force researchers to observe the operation and use of an artifact within its intended working environment. The basic premise is that abstract models provide no substitute for real application. Similarly, the results of controlled experiments fail to provide generic results that can be accurately used to assess performance outside of those controlled settings. In particular, the Hawthorne effect suggests that people and, indeed systems, will perform very differently when they are placed within an empirical setting. Repair and maintencnace activies are very different for equipment that is provided in a laboratory setting. Individuals react differently when they know that they are being observed. Hermeneutic research, therefore, relies upon the interpretation of signs and observations in the working context rather than on explicity asking people about the performance of their systems. Hermeneutics techniques urge researchers to enter into the workplace. Taken to an extreme, the performance of an algorithm could only be assessed in field trials with real sets of data on existing architectures with `real' levels of loading from other applications. This stress upon the analysis of a final implementation closely resembles proof by demonstration. The major difference, however, is that the researcher approaches the context of work with an open mind and without any set hypothesis to prove or disprove (Suchman, 1987). This raises problems for the conduct of directed research because users may not use programs in the manner that was intended. For example, it can be difficult to demonstrate that one search engine is faster than another if users continually abandon their requests after one or two items are returned or if they only use those search engines once or twice in their working day.

Conclusions and a Way Forward...

Computing science is an immature discipline. Vast resources have also been poured into the subject in a relatively short period of time. This has brought startling advances in both hardware and software engineering. Unfortunately the development of computing technology has not been matched by a similar development in academic research techniques. In the pursuit of technological goals, researchers have borrowed models of argument and discourse from disciplines as varied as philosophy, sociology and the natural sciences. This lack of any agreed research framework reflects the strength and vitality of computing science. An optimist might argue that we have learnt greatly from the introduction of hermenuetics into the field of requirements analysis. Similarly, we have profited from the introduction of mathematical models of argument to specify and verify of complex systems. A key aim of this paper is, however, to encourage people to think about the costs that have also been incurrend by the heterogenous nature of research in our discipline:

> \item err>

I do not argue that we must develop a single research model for Computing Science. I do, however, argue that researchers must actively think about the strengths and weaknesses of the research tradition that they adopt. Too often, MSc and PhD theses slavishly follow empirical or formal proof techniques without questioning the suitability of those approaches. For example, the hermeneutic tradition has delivered results that ignore the constraints of time and money on commerical system development. Formal methods research has produced results that abstract so far away from the problem domain that they cannot be applied or validated. The tragedy is that unless we begin to recognise these failures then we will continue to borrow flawed research methods from other disciplines.

References

- B.W. Boehm, T.E. Gray and T. Seewaldt, Prototyping Vs. Specification: A Multi-Project Experiment, IEEE - Seventh Conference On Software Engineering, 473-484, Computer Society Press, Washington, United States of America, May, 1984.
- M. Burrows, M. Abadi and R. Needham, A Logic of Authentication. ACM Transcations on Computer Systems, 8(1):18-36, 1990.
- N.E. Fuchs, Specifications Are (Preferably) Executable, Software Engineering Journal, 323-334, September 1992.
- A. M. Gravell and P. Henderson, Executing Formal Specifications need not be Harmful, Software Engineering Journal, 104-110, March 1996.
- I.J. Hayes and C.B. Jones (1989), Specifications are not (necessarily) executable, Software Engineering Journal, 1989, 4, (6), pp. 330-338
- C.W. Johnson, Literate Specification, Software Engineering Journal, 225-237, September, 1996.
- L. Suchman, Plans And Situated Actions: The Problem of Human Machine Communication, Cambridge University Press, Cambridge, United Kingdom, 1987.