The Importance of Theory in Computing Curricula *

ROSEN C C H, ALDRIDGE S
(School of Computing, University of Derby, Derby, UK)

CLC number: TP31 Document code: A Article ID: 0529-6579 (2007) S2-0024-05

In 1995 Ince[1] wrote an article for the Times Higher Educational Supplement noting that there was little theoretical understanding of software engineering and that therefore computer studies did not constitute an academic discipline. Computing departments should be balkanised. Computing staff should be dispersed to other departments such as maths, psychology and business studies.

This controversy has recently been reinvigorated by McBride's article[2] in the same paper and on the British Computer Society web site[3] arguing that in face of the massive decline in computing student intake onto university undergraduate courses that the future for traditional courses is bleak. The reaction generated[3—4] reflects the anxiety within computing departments. The concern is evident.

The decline in student admissions onto computer related degrees over the last three or four years is well documented. In the UK between 2000 and 2006 there has been a drop of 42% in applications to IT related courses[5—6]. In the US similar declines have been reported[7—8]. The question of how to halt this decline is top of the agenda for computing academics. The Council of Professors and Heads of Computing (CPHC) recommend that government provide incentive schemes to study, teach and retain ICT skills[5]. Others focus on making computing courses more attractive and changing prospective computing graduates' perception of the subject through marketing and image projection[9]. They fail to recognise that the perception computing has amongst the general public is inherently volatile due to the lack of definition of what "computing" is. The ephemeral and ambiguous definition of computing has served the subject well in the past by making computing appear mystical and somehow wondrous. But now, when every teenager has a computer in their bedroom, that mystical quality has evaporated. A situation that has not been helped by the false dawn of the dot com boom, the non— event of Y2K chaos and the widespread reporting of computer systems failures. Now computing, as a discipline has to become more mature to compete on equal terms with all other subjects. A competition that currently it is evidently losing.

We believe the reasons for this are more fundamental than client perception of the subject. We suggest that to survive as an academic discipline, curricula must move from skills development towards a recognition that such skills application must derive from theoretical comprehension of the subject. In order to achieve this, academics must develop theoretical models of the processes in which computing professionals engage. We have turned to Abbott[10] to support this argument. Before progressing further however, it is necessary to define the term "computing".

The concept of "computing" encompasses such a broad range of activities that it introduces ambiguity into the debate which can result in endless and ultimately futile argument. Computing is used to encompass both the use of and development of computer hardware and software. It includes information systems (IS), information technology (IT), information computing technology (ICT), software engineering (SE), software and / or systems development (SD), and the systems development process (SDP). In this article we will refer to "software systems" as the core artefact with which we are concerned, and to "software systems development" as the main activity that computing departments should be concerned with. This excludes some activities that many computing departments currently perform, but as will emerge from this paper, these are the subject of jurisdictional conflict not core elements of the discipline.

*收稿日期: 2007—10—28
作者简介: ROSEN C C H, male, senior recturer, E-mail: C.Rosen@Derby.ac.uk

Abbot(op. cit.) proposed a model of the professional jurisdiction, by which he meant how professions gain and maintain authority over work in a profession in the eyes of government, the public and other professions. His model suggests that professions achieve this status through a process of "Diagnosis", "Inference" and "Treatment". Diagnosis describes how problem domains are categorised and colligated(structured/described). Inference is the theoretical underpinning of the discipline, or rather the epistemology(accepted ways of thinking) about problems. It connects diagnosis to treatment for complex problems faced by the domain. Inference confers the right to theorise because the theoriser must have knowledge of the theoretical base.

"Any profession has rules dictating when a professional must resort to complex inference"(op. cit. P51)

The theoretical base must be the unique domain of the "professional① practitioner.

Treatment concerns the ways to solve the problem.② In this article we will argue that computing has focussed on treatment to the detriment of diagnosis and inference. In the course of this paper, we will suggest a top level diagnosis of the software systems problem and consider some potential inferences, but the main purpose of this paper is to demonstrate that without a theoretical framework for the discipline which we contend does not exist③ at present, computing as a distinct academic subject will cease to exist.

We will address "inference" in the context of software systems development first as we consider this is central to the malaise currently facing computing departments. We will return to the concept of diagnosis later. Inference implies having a distinct body of knowledge that is unique to the subject domain. It is important to defend the subject from jurisdictional claims by other professions and to prevent erosion of professional status created by alternative approaches to the problem solution. If limited or no theory exists, there is nothing to exclude untrained people from entering the profession. Abbot argues that professions gain jurisdiction over such work by establishing when judgement is required to solve complex problems. Jamous and Pebille[11] describe this as the "indetermination/technicality ratio". If solutions can be determined simply by the application of techniques and little judgement is required, the indeterminacy ratio is low. When expert knowledge is required to determine an action, the ratio is high. This occurs in "non-routine" occupations[12].

If theory exists, but is not unique to the discipline, it makes it possible for other academic domains to say "We already do that. Your discipline offers nothing new." Examples of this happening can be seen in business schools offering degrees in ecommerce and maths schools providing degrees in operational design. Ince's article(op. cit.) was premised on this being the case. Computing represents a recent challenger for jurisdictional claim over socio-technical solutions. This claim will most likely be challenged by other previous authorities such as electrical engineers, creative designers, mathematicians and business consultants. Without articulating a unique theoretical position that can be rigorously intellectually defended, computing will most probably loose that claim.

The question therefore is "does computing have a jurisdictional claim?" We would argue that it does, but that computing departments have concentrated too much on treatments to the detriment of inference. If we look at typical software systems curricula[13-17] we can ask of each subject area "does this subject address the "what" or "how" of the situation, or does it examine the "why" of the situation?" Take for example computer systems architecture. Do we teach what architectures exist and how do they work, or why do we choose one architecture rather than another? If we do the latter, is the answer simple or complex? Does it require deep knowledge of our domain, or could it be answered by, say, a business economist? Another example might be software engineering. There is some theory to suggest that complexity in programs should be avoided. But this theory derives from cognitive psychology, not from computing, and it is rarely or only superficially addressed in computing courses. Computing courses teach how to program using modularity or object orientation, in other words, how to solve a problem, not the theoretical com-

① Professional in this context means more than simply being paid. Refer to 10. Abbot A., The System of Professions. 1988, Chicago: University of Chicago Press. for arguments relating to professionalism.

② Abbot contends that the relationship between inference and treatment is, in practice, not necessarily as close as professionals would like to believe(or make believe). This does not detract from the necessity to maintain the inferential base.

③ Whether or not computing does, or does not, have a theoretical base is in itself contentious. We would argue that it is very/too limited at present, or at least poorly articulated, and is insufficiently unique for the purpose of defining computing as a distinctive academic domain. There is insufficient space here to debate this point in detail. We consider only how it might be conducted, but we believe that such a debate is fertile ground for improving the articulation of a theory/theories of computing.

ponents of problem constructs. It would be possible to work our way through the IT curricula conducting the same analysis. We contend that if we were to do so, we would discover that the draw labelled "unique computing theory" would be pretty bear, certainly compared to other academic disciplines. What we get is a series of treatments. The principles underlying inferential thinking are limited.

To identify the elements of a justifiable jurisdictional claim we must consider what theory already exists in the domain and explore whether this constitutes a sufficiently distinctive position. We believe two seminal papers provide a foundation. Brooks' paper "No Silver Bullet"[18] identifies four characteristics of software that result in it being improbable that software development is likely to make the same productivity improvements that hardware achieves. These characteristics are intangibility, malleability (how easy it is to change software), complexity and "conformability" (the pressure to adapt to the surrounding environment). None of these characteristics are unique to software. However Dijkstra[19], when considering the needs of students learning to programme, identified a fifth property that when combined with Brooks' intangibility criterion, does make software systems unique. This is that software is "discrete". In spite of the fact that software cannot be sensed as a real world entity, it has a fixed physical presence within a virtual environment. This makes it the only known intangible artefact. This means that how it behaves, and therefore how it must be produced, is unique and must be understood in its own unique context. The inferences that might be drawn from this context will therefore be unique. How and when expert inference is required can be considered further, but we also need to discuss diagnosis.

Software is intangible. Therefore it is not possible to create physical models of software systems before they are produced as is possible with physical artefacts. The models of software systems such as data flow diagrams, class diagrams or even the code itself are all abstract representations of the system that are open to interpretation and judgement. People in both the application domain (users) and the development domain (developers) need to develop conceptual models of the system without the aid of physical models. The intangible nature of software means that it is not possible to create physical models of the product that can be used to reduce the discontinuities in conceptual constructs. The challenge for software systems developers is to arrive at sufficient consensus and maintain a sufficiently coherent understanding of the system throughout the development process, to produce a discrete artefact with which application domain members are satisfied. The challenge for computing departments is to develop general theory which describes this process sufficiently to validate the treatments that are proposed. Offering treatments that cannot be substantiated by underlying theory, whether these be object orientated approaches or software process improvement, have little credibility however fashionable they become, because it is not possible to predict if they will or will not be effective in the future.

Theory from other academic domains may offer potential perspectives that are helpful, but they need to be considered with reference to how the properties of software affect production. Software systems development therefore has a unique intellectual jurisdictional claim. If the inferential framework is not applied, catastrophic failure[20,21] can occur. Calculating the strength of this claim requires consideration of the inferences that derive from the distinctiveness of software systems.

It is well beyond the scope of this paper to present such theories, but briefly, by way of exemplars, we would suggest the following possibilities. Understanding the relationship between reliability, security and usability in this context of a complex, intangible, discrete artefact might provide interesting theoretical constructs. Another possibility might be the reconsideration of the SDLC as a state model, rather than a stage model of development. This would change the emphasis of investigation from progression from stage to stage, to considering the interrelationships between states. A third line of enquiry might be modelling the interrelationship between the social—technical and the socio—dynamic aspects of the software systems development process. Some theory already exists, but it is not sufficient coherently articulated to provide computing professionals jurisdiction over work in computing. Without jurisdiction we have no cogent argument to convince prospective students to come to university to study computing. If computing is to have a future, computing departments need to refocus their attention from developping treatments to developping theory. Prospective students, clients of software systems developers and the developers themselves would then have a better understanding of what they are trying to achieve and why it is essential that they study the problem domain before proposing specific treatments.

This is not the first call for the development of theory in computing. A call emerged from the International Conference on Software Engineering 2000[22]. Sommerville[23] and Phoha[24] make similar pleas. McBride[25] also suggests the need for such theory. This is however the first time of which we are aware that the future

prospects for computing departments and the development of theory have been so explicitly linked. We believe that the challenge to develop and articulate theory that we have outlined must be broadly accepted if computing as an academic domain is to survive as anything more than obscure offshoots of maths, business and psychology. Computing academics should therefore withdraw from pursuing new treatments and teaching solutions, and concentrate on developing theoretical understanding. If we fail to make this transition, we will fail to satisfy industrial needs, the long term consequences of which could be unconscionable.

References:

[1] Ince D. Clockwork factories[C]. Times Higher Education Supplement, 10th March 1995.

[2] McBride N. Erase old programme and launch new version[C]. Times Higher Educational Supplement, February 2007.

[3] Mander K. Demise of computer science exaggerated, BCS Website Oct 2007: http://www.bcs.org/server.php?show=ConWebDoc.10138

[4] Wilkes Y, et al. Computing is the new Queen of the Sciences[C]. in Times Higher Educational Supplement. 2007.

[5] (CHHC) Council of Professors and Heads of Computing, UK ICT Industry Skills and Jobs Crisis (Briefing Note). 2007.

[6] (UCAS) Universities and Colleges Admissions Service. UCAS Statistics 2001—2006 [cited 2007; Available from: http://www.ucas.com/figures/index.html.]

[7] (CRA) Computing Research Association, 2005—2006 Taulbee Survey. Computing Research News, 2007.

[8] Kessler M. Fewer Students Major in Computer. USA Today 2007: http://www.usatoday.com/tech/news/2005—05—22—computer—science—usatx.htm.

[9] Remington W. How do CS/IS Departments Respond to Decline in Interest in the Computing Professions: Panel Discussion[J]. Computing Sciences in Colleges 2006, 21(4): 49—51.

[10] Abbot A. The System of Professions[M]. Chicargo University of Chicago Press, 1988.

[11] Jamous H, Peloille B. Professions or self—perpetuating systems: changes in the French university—hospital system, in Professions and Professionalization[M]. J A Jackson, Editor. Cambridge at the University Press: Cambridge. 1970, 111—152.

[12] Perrow C. A Framework for Comparative Analysis of Organizations. American Sociological Review, 1967, 32(2): 194—208.

[13] Davis G B, et al. IS97 Curriculum Report[C]. ACM, 1991: 1—94.

[14] ORD G. The SEI Undergraduate Curriculum in Software Engineering[C]. in 22nd SIGCSE Tech. Symp. on Comp. Sci. Educ. (SIGCSE'91). 1991: ACM SIGCSE Bulletin.

[15] Joint Task Force on Computing Curricula, Computing Curricula 2001 Computer Science[C]. IEEE Computer Society & ACM, 2001.

[16] (LACS) Liberal Arts Computer Science Consortium, A 2007 Model for a Liberal Arts Degree in Computer Science[C]. ACM Educational Resources in Computing, 2007, 7(2): 1—33.

[17] (QAA) Quality Assurance Agency for (Higher Education). Honours Degree Benchmark Statement (for Computing). 2007 [cited; Available from: http://www.qaa.ac.uk/academicinfrastructure/benchmark/statements/computing07.pdf.]

[18] Brooks Jr F. No Silver Bullet: Essence and Accidents of Software Engineering[C]. Computer, 1987: 10—19.

[19] Dijkstra E W. On the Cruelty of Really Teaching Computer Science[J]. Communications of the ACM, 1989, 32(12): 1398—1404.

[20] Leveson N. An Investigation of Therac—25 Accidents[J]. IEEE Computing, 1993, 26(7): 18—41.

[21] Lions J L. ARIANE 5 Flight 501 Failure 1996, European Space Agency: Paris.

[22] Easterbrook S, et al. Beg Borrow and Steal Workshop[C]. in Int. Conf. on Software Engineering 2000. Limerick.

[23] Sommerville J. Key Note in Empirical Assessment of Software Engineering 2003. Keele, Saffs, UK.: Keele University.

[24] Phoha V V. Theory and Practice in Software Engineering[J]. Communications of the ACM, 1997. 40(3): 28.

[25] McBride N. Letter to the Professors. [Web page] 2007 [cited 2007 20/05/2007].