Design, The "Straw" Missing From the "Bricks" of IS Curricula

Leslie J. Waguespack
lwaguespack@bentley.edu
Computer Information Systems Department, Bentley University
Waltham, Massachusetts 02154-4705, USA

Abstract

As punishment in the biblical story of Moses the slaves were told they had to make bricks without straw. This was impossible because bricks made without straw had the appearance of strength and function but could not withstand the proof of actual use. The slaves' punishment was therefore not only to make bricks, but also to find the straw on their own with which to make them. In this day and age it would seem that many of our Information Systems curricula ask students to learn to make systems without teaching them about design. We are good at teaching students how to make software systems that do things but not so good at teaching students how one way of doing things in a system design is better than another. In this essay I consider the role of teaching systems design in preparing an IS professional and the forces that have come into play over the history of computing that have, in many cases, frozen out the study of design from the IS curricula.

Keywords: design, IS discipline, IS curricula

1. INTRODUCTION

As computing education embarks on its eighth decade of preparing the professionals who will build information systems supporting every facet of humankind's culture and commerce, the specialization of computing curricula has subdivided and compartmentalized the principles, science, and practice of computing into five general categories: computer science, computer engineering, software engineering, information technology and information systems (Shackelford, Cross, Davies, Impagliazzo, Kamali, LeBlanc, Lunt, McGettrick, Sloan & Topi, 2005). Without question the breadth of all the knowledge encompassing computing today is too large to be addressed to any significant depth in a computing student's undergraduate education. Reason and practicality dictate that the knowledge of computing be subdivided (aka. specialized) in practice-focused curricula. This essay explores the proposition that one *practice* essential to any form of computing, *design*, has been sidelined (if not virtually forgotten) in computing's curricular subdivision. This paper examines the disciplinary evolution of computing and the most recently published guidelines for computing curricula. I consider whether design education is sufficiently represented in their prescriptions and focus specifically on information systems education.

2. THE EVOLUTION OF COMPUTING FROM PROGRAMS TO SYSTEMS

In the early years of computing (1938 - 1958) computer systems (analog computers particularly) were capable of working on the solution of only a single problem at a time. This single-mindedness of function meant that computers were indivisible resources that could not be shared except through sequenced allocation (Green, 2010). Digital computing eventually revealed the opportunity to use the natural differential between the processing speeds of various computing components (i.e. I/O vs. computation usually resulted in idle time for the computation units) to multiplex tasks and recover time otherwise lost waiting for slower operations.

In that era the primary design challenge was bridging the conceptual distance between human requirements and computing functionality. Success most often depended upon the ability of designers to reshape their problems to accommodate the computer's capabilities.

The transition from running a single stream of sequential "work" through a computing resource into the coordination of multiple (seemingly) concurrent streams of "work" more closely approximated the real world of organizations and life but also introduced the challenges of workflow management (coordination, prioritization, dependency, and planning). What here-to-fore may have been challenges of resource utilization optimization for individual programs became optimization for application systems.

Although the dramatic growth of computing power and resources (e.g. virtual memory, parallel processing, multiprogramming, and multitasking: 1958-1975 (Blaauw & Brooks, 1997)) may have obviated detailed study of operating systems principles for application programmers, the same principles of problem solving remain critical because they (coordination, prioritization, dependency, and planning) had become the critical resource management issues at the service oriented application level of systems!

3. THE WIDENING BREADTH OF TECHNICAL INFORMATION IN COMPUTING CURRICULA

For the first generation of information system builders in the digital age (1956 - 1968) the patterns and recognition of software design quality in programming were learned / developed through countless repetitions of programming exercises across three or more programming languages (i.e. assembler, FORTRAN, COBOL). This included problems from the trivial (to learn syntax) to the more complex approaching application system complexity.

The paucity of pattern enforcing mechanisms in programming tools (languages, editors, compilers, debuggers, etc.) required successful developers to be vigilant as they wrote software: crafting modularity, transparency, traceability, and maintainability – the selfsame characteristics that in concert condition a holistic mindset on the design quality of systems. In particular traceability testified to the conceptual integrity of a design's pertinence as a "solution" to the problem.

Underlying structural software concepts received individual focus in coursework that isolated data structures, control structures, communications, module, and systems architecture (at various levels) more or less independent of any particular modeling or programming dialect.

Structured programming was the first overarching model to organize basic design principles of coding into a paradigm of do's and don't 's that focused on achieving qualities of clarity, reliability and transparency in code (Dijkstra, 1968).

In these first couple decades of computing removed from the research laboratories into the university classroom, the breadth of concepts and practice in computer science and computer engineering did not yet outstretch the capacity of an individual's awareness of issues and topics across the entire field.

4. THE EXPANSION OF COMPUTING'S APPLICATION SPACE FROM SCIENTIFIC TO COMMERCIAL

In the advent of digital computing (1950-1965) only a handful of organizations had access to any form of problem solving using "mechanical computation." Those organizations were resource-privileged either because of their governmental or financial power. As a result, the professionals involved in learning and employing these tools were recruited from the same ranks as those who were sought for research in mathematics, engineering and the sciences. Academia's response to the resource requirement for education of these professionals followed the same pattern as that found in mathematics, engineering and the sciences with heavy doses of foundational coursework including broad coverage of basic theory followed by extensive review of the current research in digital computation, electronic circuitry, hardware and software architecture (which usually meant reviewing the dozen or so contemporarily predominant computer designs).

As it became commercially feasible to offer computing systems within the financial means of more and more commercial customers, the demand for information systems development exploded. Professionals were needed to develop and manage computers in more far-flung application domains (business, medicine, applied engineering, etc.) in which computing's primary purpose was augmenting the existing culture of systems and problem representation in a domain. This prompted academic programs in those domains to introduce application domain-based computing education. Those programs naturally treated computing as an addendum to their "core" disciplinary foci. The subset of computing knowledge that was incorporated narrowed down to a treatment of application development. In most cases these applications were seen as generally isolated solutions to individual and separate applications of problem solving.

5. THE GROWTH OF FACADE-BASED APPLICATION DEVELOPMENT ENVIRONMENTS

Marked by a steady increase in connectivity and the coming of the Internet over the last three decades, the breadth of applying computing to more and more commercial opportunities for problem solving has swelled. Tools for application development have evolved to insulate developers more and more from the details and intricacies of the computing platforms and environment. At the same time application development has expanded to an ever-broadening population of "developers" less and less versed in the core fundamentals of computing theory and practice. Indeed business computing as confined to the collection, organization and reporting of data has evolved into more of a clerical activity as opposed to one of problem solving. Quite reasonably, as a proportion of ongoing business computing activities, "data processing" predominates.

Because of this dominance, technical education in computing activities has migrated from departments of mathematics and engineering to departments applying computing to their domain-based interests. And to the extent the academic programs focus on teaching best practice using applications of known solutions to domain-based problems, they serve their students well. But, the ever-increasing interconnectedness of information and processes has levied a new layer of complexity upon collaboration and adaptability. More than ever computing capabilities are changing “the existing culture of systems and problem representation in a domain.” The challenges arise at the frontier of known solutions where either the reshaping of the domain-based problem or the creation of innovative applications of computing require more than the mastery of off-the-shelf solutions – they require creative design. They require systems that integrate the people, policies, information, hardware, software, networks, and quality management in the design of complete, holistic solutions. They require systems that accomplish a conceptual integrity and enlightened design (Brooks, 2010).

6. THE CONFINING PEDAGOGICAL RESOURCE - CURRICULAR TURF

When we consider domain-based education (business, medicine, applied engineering, etc.) combined with the fundamentals of computing and systems, the inventory of prospective, relevant coursework quickly exceeds the course credit hour “budget” of any undergraduate curriculum. Under this pressure the balance of emphasis and the share of the curricular coursework naturally tilts in the favor of the domain-based disciplines and away from the depth of fundamental computing theory and practice needed to fuel innovation and enlightened design. This has clearly been the case in computing programs contained in schools of business naturally preoccupied with certifying their “business” credentials [AACSB 2010, EQUIS 2010]. The footprint of coursework assigned to a business computing major is seldom more than 24 course credit hours dedicated to computing.

7. WHAT DESIGN IS ABOUT

The New Oxford American dictionary defines *design* (noun) as a plan representing the form and function of something before it is built or made. Design engenders the purpose, planning or intention that exists or is thought to exist behind an action, fact or material object.

Over the last 50 years Fred Brooks has been one of the most ardent and influential advocates of design as essential to the pursuit of information system quality.

“Whereas the difference between poor conceptual designs and good ones may lie in the soundness of design-method, the difference between good designs and great ones surely does not. Great designs come from great designers. Software construction is a creative process. Sound methodology can empower and liberate the creative mind; it cannot inflame or inspire the drudge.

The differences are not minor – they are rather like the differences between Salieri and Mozart. Study after study shows that the very best designers produce structures that are faster, smaller, simpler, cleaner and produced with less effort. [...] The differences between the great and the average approach an order of magnitude." (Brooks, 1995)

In his most recent reflection on the professional practice of creating information systems that support organizational goals he comments on the central role of design in this way.

"The essentials of [design] are *plan*, *in the mind*, and *later execution*. Thus a design (noun) is a created object, preliminary to and related to the thing being designed, but distinct from it."

"A book, in this conception, or a computer, or a program, comes into existence first as an ideal construct, built outside time and space, but complete in essence in the mind of the author. It is implemented in time and space, by pen, ink, and paper; or by silicon and metal. The creation is complete when someone reads the book, uses the computer, or runs the program, thereby interacting with the mind of the maker." (Brooks, 2010)

Brooks clearly distinguishes the act of system *design* from the *implementation*. The cycle of system creation differentiates *design*, *implementation* and *use*, but it does not segregate them! Indeed their interdependency is core to understanding each aspect as declared in the agile development concept. (Beck, 2010) Although distinct, these elements of system creation fuse as they conceive, develop and judge the design qualities that mark the degree of satisfaction (success) the stakeholders experience during a system's lifetime.

This distinction between design and implementation has faded from the structure of computing education. To ignore the conceptual distinction between the *design* and an *implementation* is tantamount to accepting any "solution" without even considering whether (as Brooks declares compared to the "average") there is a solution out there that is an order of magnitude "faster, smaller, simpler, cleaner and produced with less effort."

8. CURRICULUM GUIDELINES – IN SEARCH OF DESIGN

Finding the latest focus on design in computing curricula starts with The Overview Volume on Undergraduate Degree Programs in Computing. The CC2005 report is the de facto definition of subdivisions of computing education (see Figure 1 in the appendix).

As the report declares "We have created this report to explain the character of the various undergraduate degree programs in computing and to help you determine which of the programs are most suited to particular goals and circumstances." (Shackelford et. al., 2005)

The CC2005 report explains the general evolution of computing curricula depicted in Figure 2 (see the appendix).

Among the 39 Knowledge Areas of computing identified in CC2005 only 7 reference design as a specific professional competency in any form. Among those the area definitions in the glossary do not distinguish between *design* and *implement*. To some extent this is not surprising since the CC2005 effort was primarily conceived to contrast the foci of the 5 computing subdivisions rather than explain them in detail. To get detail we must explore each of the five subdivision curriculum guideline documents: CE, CS, SE, IT and IS. (Soldan, Hughes, Impagliazzo, McGettrick, Nelson, Srimani & Theys 2004; Cassel, Clements, Davies, Guzdial, McCauley, McGettrick, Sloan, Snyder, Tymann & Weide, 2008; Diaz-Herrara & Hilburn, 2004; Lunt, Ekstrom, Gorka, Hislop, Kamali, Lawson, LeBlanc, Miller & Reichgelt, 2008; Topi, Valacich, Wright, Kaiser, Nunamaker, Sipior & de Vreede, 2010)

All 5 curriculum guideline documents liberally refer to *design* in various applications of technology to systems development. However, only the software engineering curriculum guidelines address specific aspects of design quality or design principles in its knowledge area content (Diaz-Herrara et al, 2004). Indeed only the software engineering guidelines imply to any degree that design is a separate conceptual or practical activity distinct from implementation. There are no learning unit designations in the IS 2010 curriculum guidelines addressing aspects of design distinct from a technology.

This is the case because current practice in IS curricula has assumed that teaching any form of *implementation* suffices for teaching *design*. When *implementation* was taught across several courses and languages in earlier days of computing curricula, extensive *implementation* may indeed have sufficed for *design*-focused pedagogy. In an IS curriculum today, when it is almost impossible to find room for more than two or three courses in any systems development technology or more than a single course in any particular technology, teaching *implementation* cannot suffice for teaching *design*. If these current challenges weren't severe enough, IS 2010 no longer lists *implementation* (application development) as a core requirement. With that "juridical" justification removed IS and CIS programs may find it even harder to maintain any semblance of practical system life cycle pedagogy.

9. CONSEQUENCES OF TEACHING "BRICKS WITHOUT STRAW"

De-emphasizing *design* in IS curricula results in the narrowing of the learning experience toward talking about systems rather than forming systems. Here the term "forming systems" is not limited to "writing program code," but includes developing requirements, modeling information, processes and transactions, as well as building application software. *Design* permeates the forming of systems – even if only to describe them (Waguespack 2010). *Design* is the fundamental problem-solving aspect of systems. *Design* is the foundation and justification of systems and is essential to understanding them.

The most prominent consequence of de-emphasizing *design* in IS curricula is the effect it has on IS graduates' employment opportunities. Graduates of an "about-IS" focused academic program are increasingly challenged to justify to themselves and to employers their value over graduates in the business domain without an IS degree. It is increasingly difficult for an employer to distinguish the hiring advantage of a business student with an IS major over that of an IS minor or general business graduate. Where IS programs share a college with accountancy, marketing, management, finance, etc., these programs have successfully co-opted interest in IS to their programs by offering courses focused exclusively on the use of discipline-based, extant application systems - avoiding systems development completely. As a result, unable to clearly promote the career advantages of an IS degree over "general business," IS programs find it increasingly difficult to recruit IS majors.

10. WHAT THE FUTURE MAY HOLD

Whether Information Systems is or is not a discipline has long been the subject of debate in the field of computing. This can be evidenced by the search for labels in the field: DP, IS, MIS, CIS, and IT. Clearly IS first emerged at the intersection of computer science, business, management and (many would say) engineering. Over the past two or three decades many IS programs have devolved by de-emphasizing the construction aspects of their curricula; effectively jettisoning merged content from computer science and engineering in the process.

This essay contends that the primary loss in this devolution has not been "coding skill" in some particular programming language. The loss is the aspect of design as a holistic mindset and the tools it provides in shaping IS problem representation and problem solving – applying computing in the information and organizational contexts (Denning 2004) and reinforcing "systems think" (Waguespack 2010). This loss negatively impacts the students' ability to understand requirements and formulate models of software, models of business, and models of business process. In IS, design is the act of fusing technological opportunity with business opportunity often reshaping or reinventing both. Absent design, computing assumes the status of a contraption that one might take off the shelf as-is, surrendering the solution quality to the purposes of others – basically surrendering innovation to the appliance manufacturers. If the trajectory of this evolution continues I believe the debate will be over and IS as a discipline will indeed be no more.

The challenge is no simple one. If Information Systems is to maintain its valid role as the bridge between computing and the effective / efficient application of technology to information and process problems, IS curriculum architects must find a way to re-energize the teaching of *design* in their programs. In many institutions business programs are limited to prerequisite chains no longer than two courses. That makes it unlikely that renewed emphasis can be gained simply by adding courses to existing program structures. Some renewed energy may be gained through creative pedagogy by introducing systems building activities into more theoretical IS study (e.g. computer organization, networking, project management or policy). Such a creative reorganization of learning activities will surely require extensive investment in textbook and laboratory redirection. In some cases this will require the reversal of the IS-diffusion among business departments. In other cases it may require the inventive re-structuring of curricula that bridge departments of IS and computer science to take broader advantage of arts and sciences elective opportunities across the university.

In any case, the time is relatively short for reversing the decline of IS's relevance as an academic discipline. Remarkably, as few as the number of graduates from most IS programs there are, they are highly sought-after, and the employment market for them has weathered major storms of off-shoring and economic downturn. These are indications that society (particularly business) still needs practically educated professionals who understand both the application domain and computing, and combine that knowledge and skills to deliver tomorrow's quality, innovative information systems. How will IS programs and higher education respond?

11. ACKNOWLEDGEMENTS

Thanks to helpful referees. Special thanks are due my colleagues in Computer Information Systems at Bentley University for their insightful discussions and comments on these ideas.

12. REFERENCES

AACSB (2010). *Eligibility Procedures and Accreditation Standard for Business Accreditation*. Retrieved July 16, 2010 from http://www.aacsb.edu/accreditation/AAACSB-STANDARDS-2010.pdf

Beck K., Beedle M., van Bennekum A., Cockburn A., Cunningham W., Fowler M., Grenning J., Highsmith J., Hunt A., Jeffries R., Kern J., Marick B., Martin R.C., Mellor S., Schwaber K., Sutherland J., & Thomas D. (2010). *Manifesto for Agile Software Development*. Retrieved July 12, 2010 from agilemanifesto.org

Blaauw, G.A., & Brooks, F.P. (1997). Computer Architecture: Concepts and Evolution. Addison-Wesley, Reading, Massachusetts.

Brooks, Frederick P. (1995). The Mythical Man-Month: Essays on Software Engineering (2ed). Addison-Wesley, Boston, MA.

Brooks, Frederick P. (2010). The Design of Design: Essays from as Computer Scientist. Addison-Wesley, Pearson Education, Inc., Boston, MA.

Cassel L., Clements A., Davies G., Guzdial M., McCauley R., McGettrick A., Sloan B., Snyder L, Tymann P., & Weide B.W., (2008). Computer Science Curriculum 2008 An Interim Revision of CS2001. Association of Computing Machinery (ACM), & IEEE Computing Society (IEEE-CS).

Dijkstra, E. (1968). GOTO Statement Considered Harmful. *Communications of the ACM*, 11(3), 147-148.

Diaz-Herrara, J.L., & Hilburn, Thomas B. (eds.) (2004). Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering, IEEE Computing Society (IEEE-CS), Association of Computing Machinery (ACM).

Denning, P. J. (2004). The Great Principles of Computing, *Ubiquity*, 4(48), 4–10.

EQUIS (2010). *EQUIS Standards and Criteria*. Retrieved July 16, 2010 from http://www.efmd.org/attachments/tmpl_1_art_041027xvpa_att_080404qois.pdf

Green, T. (2010). Bright Boys. A.K. Peters, Ltd., Natick, Massachusetts.

Lunt, B.M., Ekstrom, J.J., Gorka, S., Hislop, G., Kamali, R., Lawson, E., LeBlanc, R., Miller, J., & Reichgelt, H. (eds.) (2008). Information Technology 2008: Curriculum Guidelines for Undergraduate Degree Programs in Information Technology, Association of Computing Machinery (ACM), IEEE Computing Society (IEEE-CS).

Shackelford, R., Cross, J.H., Davies, G., Impagliazzo, J., Kamali, R., LeBlanc, R., Lunt, B., McGettrick, A., Sloan, R., & Topi, H., (2005). Computing Curricula 2005: The Overview Report, Association for Computing Machiner (ACM), The Association of Information Systems (AIS), The Computer Society (IEEE-CS).

Soldan, D., Hughes, J.L.A., Impagliazzo, J., McGettrick, A., Nelson, V.P., Srimani, K., & Theys, M.D. (eds.) (2004). Computer Engineering 2004: Curriculum Guidelines for Undergraduate Degree programs in Computer Engineering, IEEE Computer Society (IEEE-CS), Association for Computing Machinery (ACM).

Topi, H., Valacich, J.S., Wright, R.T., Kaiser, K.M., Nunamaker, J.F. Jr., Sipior, J.C., & de Vreede, G.J. (eds.) (2010). IS2010: Curriculum Guidelines for Undergraduate Degree Programs in Information Systems, Association for Computing Machinery (ACM), Association for Information Systems (AIS).

Waguespack, L. J. (2010). Thriving Systems Theory and Metaphor-Driven Modeling. Springer, London, U.K.

Editor's Note:

This paper was selected for inclusion in the journal as an ISECON 2010 Meritorious Paper. The acceptance rate is typically 15% for this category of paper based on blind reviews from six or more peers including three or more former best papers authors who did not submit a paper in 2010.

Appendix

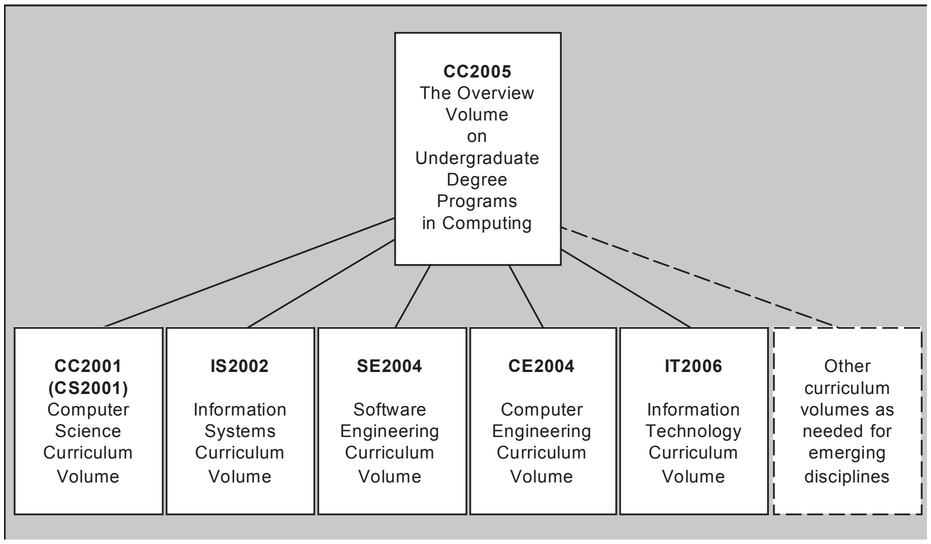
CC2005
The Overview Volume on Undergraduate Degree Programs in Computing
CC2001 (CS2001) Computer Science Curriculum Volume
IS2002 Information Systems Curriculum Volume
SE2004 Software Engineering Curriculum Volume
CE2004 Computer Engineering Curriculum Volume
IT2006 Information Technology Curriculum Volume
Other curriculum volumes as needed for emerging disciplines

Figure 1 - Computing Curricula Guidelines

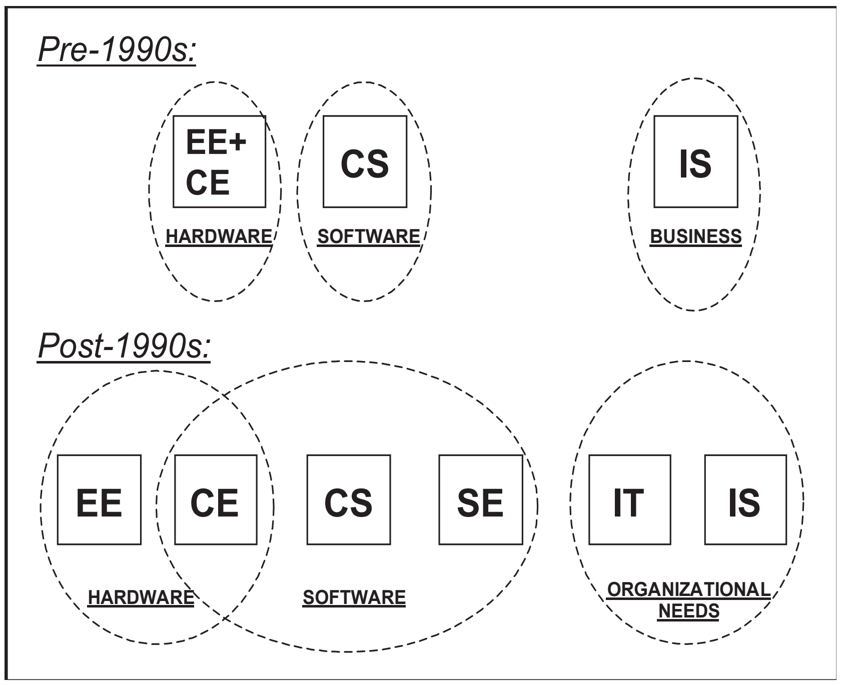

Figure 2 - The Outward Appearance of Computing Curricula Evolution