DOCUMENT RESUME

ED 396 691 IR 017 867

AUTHOR Cordes, David; And Others
TITLE Breadth-Oriented Outcomes Assessment in Computer Science.
PUB DATE 94
NOTE 9p.; In: Recreating the Revolution. Proceedings of the Annual National Educational Computing Conference (15th, Boston, Massachusetts, June 13-15, 1994); see IR 017 841.
PUB TYPE Reports - Research/Technical (143) -- Speeches/Conference Papers (150)

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS College Graduates; *College Seniors; *Computer Science; *Educational Assessment; *Evaluation Methods; Higher Education; Introductory Courses; *Knowledge Level; Pilot Projects; *Program Evaluation; Undergraduate Students
IDENTIFIERS University of Alabama

ABSTRACT

Little work has been done regarding the overall assessment of quality of computer science graduates at the undergraduate level. This paper reports on a pilot study at the University of Alabama of a prototype computer science outcomes assessment designed to evaluate the breadth of knowledge of computer science seniors. The instrument evaluated two areas: technical knowledge and knowledge of computing history and culture. The exam, which was presented to students unannounced during a regularly scheduled class meeting, consisted of 100 questions that covered the basic areas of the discipline. Results indicated: (1) graduating seniors have a degree of breadth knowledge roughly consistent with what one would expect; (2) the introductory breadth course does not provide breadth knowledge equivalent to what seniors obtain after taking several advanced depth courses, although a substantial amount of breadth material is covered in that course; and (3) students have little knowledge of computing history and culture at all levels of the curriculum. (Contains 10 references.) (Author/AEF)

Paper (W3-201A)

Breadth-Oriented Outcomes Assessment in Computer Science

David Cordes
Department of Computer Science
The University of Alabama
Tuscaloosa, AL 35487
cordes@cs.ua.edu

Allen Parrish
Department of Computer Science
The University of Alabama
Tuscaloosa, AL 35487
parrish@cs.ua.edu

Susan Vrbsky
Department of Computer Science
The University of Alabama
Tuscaloosa, AL 35487
vrbsky@cs.ua.edu

Key words: computer science, computing literacy, outcomes assessment

Abstract

Given the immaturity of computer science as a discipline, curriculum planning and organization remains a subject of much debate. Moreover, little work has been done regarding the overall assessment of quality of computer science graduates at the baccalaureate level. In this paper, we report on an initial prototype of an assessment instrument designed to evaluate the breadth of knowledge of computer science seniors. The instrument seeks to evaluate two areas: technical knowledge and knowledge of computing history and culture. Our results indicate that our curriculum does appear to contribute to students' technical knowledge; however, students are emerging with very little knowledge of the history and culture of computing. As a result of this apparent lack of knowledge, we are currently adding a new capstone course to our curriculum. This paper discusses the design and rationale for this course.

Introduction

To date, little work exists regarding the overall assessment of the quality of undergraduate computer science students. Such quality assessment (often called *outcomes assessment*) is becoming a topic of increasing interest in other disciplines (Light, 1992).

A number of techniques exist for performing outcomes assessment of computer science majors. Given that much of computer science is oriented toward design and problem solving, an examination devoted to such activities is intuitively appealing. However, one can excel at design and problem solving activities and yet fail to have a basic knowledge regarding many fundamental computer science concepts. We believe that a successful computer science program should not only instill design and problem solving skills, but should also provide the student with the knowledge of basic, fundamental concepts from across the breadth of the discipline. Moreover, the student should possess the ability to recall, understand and utilize such concepts. At the very least, a successful student should be able to converse on any of these concepts, even if the student has not "mentally referenced" the concept for a long period of time. In this paper, we refer to the basic knowledge of a wide variety of fundamental concepts about a discipline as ***breadth-oriented*** knowledge.

Based on these concerns, we have developed a prototype computer science outcomes assessment that tests for breadth-oriented knowledge in computer science. In a recent pilot study, we administered this assessment to several distinct groups of computer science students at The University of Alabama. This paper reports on the results of this pilot study. In this study, we attempted to address three general areas:

1. Are our students graduating with breadth-oriented knowledge that spans the computer science discipline? Do our students have an appropriate foundation of computer science concepts (in addition to whatever problem solving skills they might have)?

2. Our computer science curriculum is designed to be *breadth-first*, in the sense that it contains a breadth-oriented introductory course (Cordes, 1992; Denning *et. al.*, 1989). Breadth-first curriculum design in computer science has been a subject of considerable discussion and controversy (Baldwin, 1990; Locklair, 1991; Motil, 1991; Pratt, 1990). What is the value of such a course in building a breadth of knowledge about the discipline? How do students who have just finished a course covering the breadth of the discipline compare (in terms of breadth knowledge) to advanced students who have primarily been taking specialized courses for the past two years?

3. Are our students graduating with an appreciation of the history and culture of computing? As an example, are students able to recognize the contributions of major figures in the computing field?

While (3) may be secondary in importance to technical knowledge, we consider it ironic that students that are not computer science majors may actually have a better exposure to these issues than our majors. Issues related to (3) are often covered in computer literacy courses for non-majors. However, such issues often receive little coverage in majors courses where there is often too little time to properly cover technical topics.

In the remainder of this paper, we report on the results of our pilot study. In Section 2, we address the general design of our study, as well as the design of our assessment instrument. In Section 3, we address the above three sets of questions. Section 4 summarizes our conclusions and recommends several areas for future research.

Design of the Study

The pilot study took place during the Spring 1993 semester at the University of Alabama. The students involved in this study ranged from second-semester freshman to graduating seniors. The test was given during a regular class meeting. None of the students involved in the study knew of the exam prior to that day, and thus had no preparation time. Specifics of the organization and administration of the exam are provided in the following sections.

Exam Organization and Structure

The basic exam consisted of 100 questions. These questions covered the basic areas of the discipline (algorithms, architecture, data structures, operating systems, programming languages, software engineering, history and cultural issues, and basic computer literacy). The questions were divided into four basic categories, as shown below:

- Basic definitions using multiple choice answers (25 questions)
- Deeper questions, ones that required some analysis beyond simply knowing the basic definitions, also incorporating multiple choice answers (25 questions)

- A second set of 'deep' questions, similar to the previous section only having True or False answers (20 questions)
- More definitions, heavy on the history and culture of the discipline, using matching answers (30 questions)

The questions in these four sections attempted to provide a broad, uniform covering of the discipline of computing. The first section of the exam consisted of basic definitions and terms that students within the discipline should know. The majority of questions in this section were *'literacy-based,'* that is, the questions in it were general literacy questions. Sample questions included items such as:

1. *ASCII*
   a. American System for Coding Idioms and Icons
   b. Applied Structured Coding with Intelligent Interfaces
   c. American Standard Code for Information Interchange
   d. Automated System Certification version II
2. *Semiconductor*
   a. a compound with a limited ability to conduct an electrical charge
   b. pure silicon, a tightly crystalline structure
   c. the term used to refer collectively to a collector, base and emitter
   d. a group of transistors connected by aluminum strips

The next two sections consisted of 'deeper' questions that required some actual thought on the part of the student in order to solve the problem. These questions were both multiple choice and true/false questions. Sample questions of each category are shown below:

1. *A depth-first search operates as follows:*
   a. start at a node, visit its neighbors, then visit its neighbor's neighbors, *etc.*,
   b. start at a node, visit one neighbor, then visit one of the neighbor's neighbors, *etc.*, stopping when you reach the depth of the graph
   c. search a graph from one end to the other (graph depth) as quickly as possible
   d. start at a node, visit one neighbor, then visit one of the neighbor's neighbors, *etc.*, backtracking and trying other unvisited neighbors until all nodes are visited
2. *The maximum number of items stored in a binary tree of height 4 is:*
   (a) 15 (b) 16 (c) 31 (d) 32

T/F A stack is simply a list where insertion and deletion are ensured to take place in FIFO (first-in-first-out) order.

T/F Most compilers for high-level languages can detect infinite loops at compile time.

T/F For an ordered tree, the results of printing the tree in prefix and postfix order are identical.

The final section of questions consisted of additional basic definitions and terms, similar to the first section but using matching. This section also placed a heavy emphasis on the historical and cultural issues of our discipline. A subset of questions from the matching section is shown below:

1. big-endian	A. designs sums-of-products from truth tables
2. bandwidth	B. promoted structured programming constructs
3. karnaugh map	C. the range of frequencies on a carrier signal

4. LL parsers	D. developed language SMALLTALK
5. VonNeumann	E. orders bytes from left-to-right
6. Dijkstra	F. top-down, recursive descent or table driven
7. Kay	G. developed language LISP
8. McCarthy	H. developed framework for modern-day computers

The exam, as it was originally designed, was intended to assess the knowledge of our students (at various stages within their careers) regarding the technical information within the discipline. Thus, the majority of the examination emphasized the technical issues associated with computing. However, a significant number (approximately 15% of the exam) focused on the history and cultural issues associated with computer science. As this study was an initial pilot project, it is anticipated that the exam administered next Spring will focus more strongly on these cultural and historical issues.

Exam Administration

As mentioned previously, the exam was presented to the students (unannounced) during a regularly scheduled class meeting towards the end of the Spring semester. The exam was administered to three different classes, including a second-semester freshman-level course, a sophomore/junior-level course, and a course consisting primarily of graduating seniors. The three courses involved were:

- CS 124: Introduction to Computer Science: This course is a second-semester freshman course. It assumes as its pre-requisite CS 114 (Introduction to Computer Programming). Thus, all of the students within the course are familiar with the concepts of programming and software development. It then develops a breadth-first introduction to the discipline of computing as a whole. As the students are familiar with programming issues, it is possible for them to implement prototypes and examples of the various issues the students are exploring (operating systems, compilers, complexity, *etc.*). Course content includes algorithms, data structures, basic architecture, fundamentals of complexity and computability, and the foundations of operating systems and programming languages.

- CS 325: Software Development and Systems: This course is part of the 'middle tier' of the major at the University of Alabama. After students complete the first year of the discipline, they must complete four additional CS courses (data structures, assembler, discrete math, and this course) prior to moving on to the upper-level courses in the discipline. Most of the students in this course have been in the major for approximately one and a half years, and are ready to start taking senior-level courses the following semester. This course is designed as a scaling course (Cordes & Parrish, 1993a; Cordes & Parrish, 1993b), migrating the student from a small-scale, single-person development environment to a larger-scale software development environment.

- CS 426: Introduction to Operating Systems: This is one of four required senior-level courses within the major. All students must take this course prior to graduation, and the majority of the students in this course were in their last semester at the University of Alabama when this exam was administered.

Each instructor involved in the study announced the test at the start of the period. After this, the remaining students were given the exam and allowed 75 minutes in which to complete the exam. The exams were then accumulated and graded. An analysis of the results (and their implications) is presented in the following section.

Analysis of Results

In this section, we examine the results of our pilot study by looking at the three questions outlined in the introduction:

1. Do our graduating seniors (*i.e.*, CS 426 students) possess adequate technical knowledge spanning the breadth of the discipline?

2. Does the introductory breadth course (CS 124) contribute to breadth knowledge?

3. Do our students have adequate exposure to historical and cultural issues in computing?

These questions are addressed separately in the subsections below.

Issue (1): Breadth of Technical Knowledge

To address this question, we wish to consider the performance of our seniors on technical questions. The results for all three groups (based on the 88 technical questions) are given in Table 1 below.

Group	Mean	Std. Dev.	N
CS 124	40.8780 (45%)	9.3011	41
CS 325	50.2683 (57%)	10.2250	41
CS 426	60.0345 (68%)	8.4282	29

Table 1

Of course, the question of whether 68% is "good enough" is a subjective one. However, the mean GPA for the seniors taking the exam is 2.88, indicating an overall C average for the group. Based on the typical procedure used here for exam evaluation, a 68 is a borderline C. So the technical score is roughly consistent with the quality of students taking the exam.

As an additional metric, we examined the total number of questions "mastered" by the three groups. We say that a question is *'mastered at level N'* if at least *N*% of the students answered the question correctly. Table 2 below indicates the number of technical questions mastered at the 66% (two-thirds) and 80% (four-fifths) levels by the three groups:

Group	66%	80%
CS 124	22	10
CS 325	32	19
CS 426	55	38

Table 2

Thus, at least two-thirds of the senior students mastered 55 questions (62% of total). While this number is not as high as we would like, it does seem to represent mastery of a substantial percentage of the material, and provides a baseline against which to compare in future studies.

Issue (2): Value of the Introductory Breadth Course

As discussed earlier, the inclusion of breadth-oriented introductory courses in computer science curricula has been the source of some controversy. Much of the controversy has been centered on the contribution of the course in terms of providing a foundation with respect to the overall discipline. The idea of having a breadth-oriented introductory course was formalized in the well-known Curriculum '89 report (Denning, *et. al.*, 1989). However, others have suggested that computer science curricula should adhere to a model where the foundation of the program should be based on depth-oriented courses in design and problem solving, rather than exposure to material covering the entire breadth of the discipline (Baldwin, 1990; Motil, 1991; Pratt, 1990).

One way to assess the contribution of breadth-oriented courses in providing a foundation for the remainder of the curriculum, is to separately consider two questions:

1. Does a breadth-oriented introductory course really contribute to breadth knowledge?
2. Does breadth knowledge provide a foundation for the remainder of the curriculum?

By administering a breadth-oriented exam to students completing CS 124, this study begins to address question (1). In particular, consider Table 2 in Section 3.1 above. Two observations (one negative, one positive) are evident from this table:

- Students completing CS 124 have only mastered (at the two-thirds level) about 25 of the material covered on this exam. Thus, three-fourths of the questions were inaccessible to most of these students. Consequently, there is a definite limitation to the degree of breadth students are able to obtain from a one-semester course.

- Students completing CS 124 mastered approximately 40 of the material mastered by the graduating seniors. This is significant when considering that these students have only completed 8 hours in the computer science curriculum (versus 45 hours for the seniors).

Thus, while students taking an introductory course have not mastered general questions spanning the entire breadth of the discipline, they have mastered a substantial percentage of what they likely will ultimately acquire.

Thus, while our introductory breadth course does not result in complete command of the breadth of the discipline, it clearly contributes substantially to the breadth of knowledge that students ultimately obtain. Further research is needed to determine whether an initial breadth course provides a good foundation for the rest of the curriculum.

Issue (3): History and Cultural Knowledge

As observed earlier, there were 12 questions on the exam devoted to primarily "historical and cultural" issues. All but one of these questions required the identification of major contributors to computing and related disciplines. One additional question required students to identify the meaning of the acronym "ACM" (*i.e.*, Association for Computing Machinery). The list of major figures that students were required to identify is as follows:

Dijkstra	Roussel
Chomsky	Kay
Knuth	DeMorgan
Babbage	VonNeumann
McCarthy	Hoare
Backus	

All of the questions related to this area are found in the matching section of the exam in the Appendix (Questions 71-100).

As Tables 4 and 5 illustrate, student performance on this area of the exam was universally poor. Table 4 shows student mean subscores on these 12 questions, while Table 5 shows the percentage of students from each class getting each question correct.

Group	Mean	Std. Dev.	N
CS 124	2.0488 (17%)	1.8021	41
CS 325	2.6098 (22%)	1.3206	41
CS 426	3.8966 (32%)	1.9151	29

Table 4

Question	CS 124%	CS 325%	CS 426%
Dijkstra	4.9	14.6	17.2
Roussel	14.6	2.4	27.6
Chomsky	2.4	7.3	20.7
Kay	7.3	0.0	3.4
Knuth	12.2	17.1	27.6
DeMorgan	7.3	26.8	34.5
Babbage	39.0	48.8	65.5

VonNeumann	17.1	34.1	44.8
McCarthy	17.1	2.4	17.2
Hoare	9.8	0.0	17.2
Backus	12.2	12.2	13.8
ACM	61.0	95.0	100.0

Table 5

Although this is a small sample of questions, most students were unable to identify *any* of the major contributors to the computing field. This is particularly problematic, given that computer literacy courses for *non-majors* routinely cover many of these contributors to the computing discipline. We suspect that although many of these individuals are mentioned from time to time, students are never placed in a position of having to assimilate and organize information about all of them for long-term recall. Consequently, our response to these results has been to develop a new capstone course whose primary purpose is to present a retrospective over a variety of non-technical issues in computing. This required course, entitled "Ethical and Societal Issues in Computing," covers a variety of legal, ethical, historical and cultural issues. Its outline appears as follows.

CS 440: Ethical and Societal Issues in Computing	
Week	Topics
1-2	History and Culture of Computing Defining the discipline of computer science Major contributors to computing ACM, IEEE, Turing Award
3-4	Ethics in Computing Principles of Ethical Behavior/Case Studies ACM Code of Professional Conduct
5-6	Computer Crime Computer hackers, viruses and worms Case studies of criminal activity
7-8	Computing Risks Causes of computing failures/case studies Safety-control techniques (software/hardware)
9-11	Legal Issues Product liability Software patents and copyrights Intellectual Property Rights
12-15	Student Paper Presentations and Debates

This three (semester) hour course will be offered for the first time during the academic year 1994-95. Our objective in offering this course is to ensure that students have a background in social and cultural issues that is (at the very least) comparable to the backgrounds of students emerging from non-major computer literacy courses. We also wanted to provide a course that gives students an appropriate background in ethical and legal issues, given the current interest in this subject (Weiss, 1990). This course has the potential to combine both of these areas in a cost-effective fashion, and has the potential to provide an opportunity for students to place many of these issues into perspective at the close of their undergraduate experience.

Conclusion

In this paper, we have discussed the results of a pilot study to conduct an outcomes assessment in a medium-scale state university computer science program. Our assessment instrument was breadth-oriented, meaning that we were attempting to assess student knowledge of facts spanning the breadth of the computer science discipline. Our results are threefold:

- Graduating seniors have a degree of breadth knowledge roughly consistent with what one would expect, given the quality of students surveyed.
- The introductory breadth course does not provide breadth knowledge equivalent to what seniors obtain after taking several advanced depth courses; however, a substantial amount of breadth material is covered in that course.
- Students have little knowledge of computing history and culture at all levels of the curriculum.

Much future work in this area is needed. In particular, dimensions of quality need to be identified and new assessment instruments need to be developed to measure quality along these different dimensions. We believe that one of these dimensions is knowledge of computing history and culture; a more detailed instrument should be developed measuring this dimension. More studies need to be done to measure the value of breadth-first versus depth-first introductory courses and sequences. Finally, at our institution, we will continue this type of evaluation, and will give particular emphasis to evaluating the success of our senior capstone course in computing history and culture.

References

Baldwin, D. (1990). Teaching introductory computer science as the science of algorithms. *Proceedings of the Twenty-First SIGCSE Technical Symposium on Computer Science Education* (pp. 58-62).

Cordes, D. (1992). Introducing computer science to undergraduates. *Proceedings of the National Educational Computing Conference* (pp. 280-283).

Cordes, D. and Parrish, A. (1993a). An incremental approach to software engineering in a science-based computing curriculum. *Proceedings of the 21st Annual Computer Science Conference* (pp. 182-188).

Cordes, D. and Parrish, A. (1993b). Ada as part of an incremental approach to software engineering. *Proceedings of the Seventh Annual ASEET Symposium* (pp. 139-146).

Denning P., Comer, D., Gries, D., Mulder, M., Tucker, A., Turner, J. and Young, P. (1989). Computing as a discipline. *Communications of the ACM*, 32 (1), 9-23.

Light, R. (1992). Explorations with Students and Faculty about Teaching, Learning and Student Life. In *The Harvard Assessment Seminars* (pp. 1-50). Cambridge: Harvard University Press.

Locklair, G.H. (1991). The introductory computer science course. *Proceedings of the Twenty-Second SIGCSE Technical Symposium on Computer Science Education* (pp. 235-239).

Motil, J. (1991). Begin-BIG: An approach to the introductory computing course. *Proceedings of the Twenty-Second SIGCSE Technical Symposium on Computer Science Education* (pp. 226-230).

Pratt, T.W. (1990). Upgrading CS1: An alternative to the proposed COCS survey course. *Proceedings of the Twenty-First SIGCSE Technical Symposium on Computer Science Education*, (pp. 68-71).

Weiss, E. (1990). Self-Assessment XXII. *Communications of the ACM*, 33 (11), 110-132.