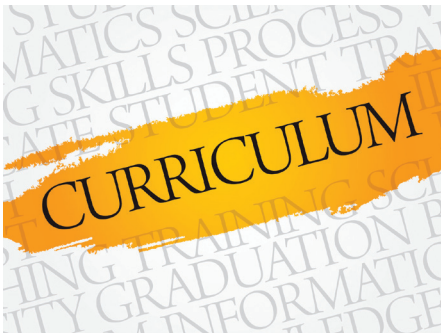
by Cara Tang,
Portland Community College

ACM Guidelines for Associate-Degree Computer Science Transfer Programs

At SIGCSE 2015, the Birds of a Feather session "Perspectives on How CS2013 Influences Two-Year College Programs" drew a standing-room-only crowd. This made it clear that ACM guidance was needed for two-year computer science transfer programs. Thus, the ACM CCECC (Committee for Computing Education in Community Colleges) in November 2015 formed a CSTransfer2017 task force to lead the development of such guidance. After two years of curriculum development work, including many cycles of feedback and revision, we are pleased to present the ACM's *Computer Science Curricular Guidance for Associate-Degree Transfer Programs with Infused Cybersecurity* [3].

Goals and Driving Factors

The guidance that is the subject of this column updates the *ACM Computing Curricula 2009: Guidelines for Associate-Degree Transfer Curriculum in Computer Science* [4]. It is aligned with ACM's *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science* [1] (CS2013) to facilitate smooth transfer with 4-year university partners. The guidance reflects the growing importance of cybersecurity in all aspects of computing, being infused with contemporary cybersecurity concepts appropriate for associate-degree computer science transfer programs. To facilitate integration into competency-based curriculum typical at many two-year schools, learning outcomes are expressed using Bloom's Revised Taxonomy [5], and accompanied by a three-tiered assessment rubric providing meaningful evaluation metrics. For a deeper explanation of the driving factors and the initial stage of the curriculum work, see Elizabeth Hawthorne's March 2016 ACM Inroads column, Updating the *ACM Associate-Degree Curricular Guidance in Computer Science* [6].



Process

The primary source document for creating these guidelines was CS2013. The 18 knowledge areas in CS2013 were divided into three groups of related areas, and a team of 5-6 community college educators was formed for each group, with the CCECC members and the CSTransfer2017 team leaders steering the process. These capable leaders included Elizabeth Hawthorne of Union County College, NJ; Cindy Tucker of Bluegrass Community and Technical College, KY; Christian Servin of El Paso Community College, TX; Teresa Moore of Volunteer State Community College, TN; Lambros Piskopos of Wilbur Wright College, IL; Markus Geissler of Cosumnes River College, CA; and myself. In addition to the CSTransfer2017 task group, significant input came from the community through survey responses (over 50 responses from 8 different countries), feedback on drafts, and interactive sessions at conferences. A summary of important milestones in the process follows.

- Nov 2015: CSTransfer2017 task group formed with three teams of 19 total community college educators
- Mar 2016: Interactive special session and workshop at SIGCSE
- Survey to a global audience of CS educators (both 2- and 4-year) on CS2013 knowledge deemed appropriate in the first two years
- Survey on cybersecurity concepts deemed appropriate for the first two years of a computer science education
- Jun 2016: StrawDog draft released for review and feedback
- Oct 2016: IronDog draft released for review and feedback
- Jan 2017: In-person meeting of the CCECC and team leaders to incorporate final feedback
- Jan 2017: Guidance endorsed by the ACM Education Board
- Mar 2017: Poster and BoF at SIGCSE presenting the guidance and soliciting program examples
- Jun 2017: Guidance published
- Jun 2017: Panel at the Community College Cybersecurity Summit (3CS) presenting the guidance and soliciting program examples

The Heart of the Guidance: Learning Outcomes

Included in the associate-degree guidance are 17 of CS2013's 18 knowledge areas (KAs). The one KA not included is intelligent systems, which consists of upper division elective content, and therefore judged not appropriate for the first two years. Two KAs,

software development fundamentals (SDF) and discrete structures (DS), are included in their entirety. For the remaining KAs, those knowledge units (KUs) and learning outcomes (LOs) determined appropriate for the first two years of a computer science program, through two rounds of public review and feedback, are included. A few changes from CS2013 are of note—The information assurance and security (IAS) knowledge area has been renamed Cybersecurity (CYB) to reflect the terminology used in the community, as well as the forthcoming ACM undergraduate curriculum guidelines for cybersecurity education. To reflect the focus on what students can do over what students know, the lists of topics from CS2013 are absent in the two-year guidance. The learning outcomes (LOs) have been rewritten to use Bloom's Revised Taxonomy [5], and a three-tiered assessment rubric for each LO is included.

The guidance consists of a total of 214 learning outcomes, with 64 related to cybersecurity as applied to computer science. The LOs in the guidance fall across the upper five levels of Bloom's taxonomy: *understanding*, *applying*, *analyzing*, *evaluating*, and *creating*. There are no LOs at the lowest level of Bloom's Revised taxonomy, remembering. Figure 1 shows the distribution of Bloom's levels in the 200+ learning outcomes. The emphasis is on student performance, with half of the learning outcomes at the applying level, and only 21% at the comprehension level of *understanding*.

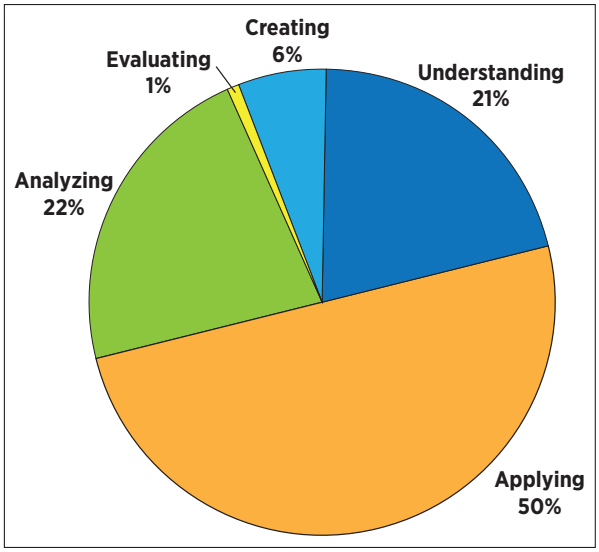


Figure 1: Distribution of learning outcomes across Bloom's levels.

Table 1 shows a selection of learning outcomes along with the associated assessment rubric from the software development fundamentals (SDF), architecture (AR), and cybersecurity (CYB) knowledge areas. Readers are invited to interactively explore the full list of learning outcomes and assessment rubrics at the ACM CCECC Guidance website [2].

Table 1: Selected learning outcomes with assessment rubric.

Learning Outcome	Emerging Standard	Developed Standard	Highly Developed Standard
SDF-01. Design an algorithm in a programming language to solve a simple problem. [Creating]	Implement an algorithm in a programming language to solve a simple problem. [Applying]	Design an algorithm in a programming language to solve a simple problem. [Creating]	Design an algorithm in a programming language to solve a complex problem. [Creating]
SDF-06. Create programs which use defensive programming techniques, including input validation, type checking, and protection against buffer overflow. [Creating]	Investigate defensive programming techniques. [Applying]	Create programs which use defensive programming techniques, including input validation, type checking, and protection against buffer overflow. [Creating]	Create complex programs which use defensive programming techniques, including input validation, type checking, and protection against buffer overflow. [Creating]
SDF-15. Apply a variety of strategies to test and debug programs. [Applying]	Explain strategies to test and debug programs. [Understanding]	Apply a variety of strategies to test and debug programs, such as unit testing and test-case generation. [Applying]	Analyze a variety of strategies to test and debug programs. [Analyzing]
AR-03. Illustrate how fixed-length number representations could affect accuracy and precision, causing vulnerabilities. [Applying]	Explain how fixed-length number representations could affect accuracy and precision, causing vulnerabilities. [Understanding]	Illustrate how fixed-length number representations could affect accuracy and precision, causing vulnerabilities. [Applying]	Examine how fixed-length number representations could affect accuracy and precision, causing vulnerabilities. [Analyzing]
AR-06. Decompose the organization and major functional units of the classical von Neumann machine. [Analyzing]	Diagram the organization of the classical von Neumann machine and its major functional units. [Applying]	Decompose the organization and major functional units of the classical von Neumann machine. [Analyzing]	Assess the organization of the classical von Neumann machine and its major functional units. [Evaluating]
CYB-04. Analyze the tradeoffs of balancing key security properties, including Confidentiality, Integrity, and Availability (CIA). [Analyzing]	Investigate the tradeoffs of balancing key security properties, including Confidentiality, Integrity, and Availability (CIA). [Applying]	Analyze the tradeoffs of balancing key security properties, including Confidentiality, Integrity, and Availability (CIA). [Analyzing]	Evaluate the tradeoffs of balancing key security properties, including Confidentiality, Integrity, and Availability (CIA). [Evaluating]
CYB-20. Illustrate key principles of social engineering, including membership and trust. [Applying]	Discuss some of the key principles of social engineering. [Understanding]	Illustrate key principles of social engineering, including membership and trust. [Applying]	Outline key principles of social engineering, including membership and trust. [Analyzing]
CYB-23. Apply cryptographic hash functions for authentication and data integrity. [Applying]	Summarize the use of cryptographic hash functions for authentication and data integrity. [Understanding]	Apply cryptographic hash functions for authentication and data integrity. [Applying]	Deconstruct a cryptographic hash function used for authentication and data integrity. [Analyzing]

ACM Guidelines for Associate-Degree Computer Science Transfer Programs

Call for Curricular Program Examples

Going beyond the curricular guidelines presented here, a collection of examples of two-year computer science programs is being assembled. A program example correlates an actual computer science transfer program with the learning outcomes in ACM's guidance, showing the ACM LOs that appear in each course that makes up the program. This is similar to the course exemplars compiled as part of CS2013. Seeing how the curricular guidance plays out in a real program can help with program updates or implementation of new programs.

If you would like to correlate your program to ACM's guidelines, or to see the existing program examples, visit our website at ccecc.acm.org/correlations. For any questions or suggestions on the process, use the contact form at ccecc.acm.org/contact or feel free to email me. ❖

References

1. ACM and IEEE Computer Society. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. (New York: ACM, 2013).
2. ACM CCECC Guidance; ccecc.acm.org/guidance/computer-science-2017. Accessed 9 July 2017.
3. ACM Committee for Computing Education in Community Colleges. *Computer Science Curricular Guidance for Associate-Degree Transfer Programs with Infused Cybersecurity*. (New York: ACM, 2017).
4. ACM Two-Year College Education Committee. *Computing Curricula 2009: Guidelines for Associate-Degree Transfer Curriculum in Computer Science*; http://ccecc.acm.org/files/publications/2009ComputerScienceTransferGuidelines.pdf. Accessed 2017 April 22.
5. Anderson, L.W. and Kratwohl, D.R. eds., *A Taxonomy for Learning, Teaching and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. (New York: Longman, 2001).
6. Hawthorne, E.K. Updating the ACM Associate-Degree Curricular Guidance in Computer Science. *ACM Inroads* 7, 1 (March 2016), 30-31.

Cara Tang
Portland Community College
12000 SW 49th Ave.
Sylvania, TCB 312
Portland, OR 97219
cara.tang@pcc.edu

DOI: 10.1145/3107408

OPINION

IS EDUCATION

by Heikki Topi,
Bentley University

Everything Changes so Fast—A Great Reason to Learn from the Past!

In computing, it is easy to emphasize the promise and importance of the future, focus on the challenges of the present, but forget what has happened in the past because—we tend to believe—everything changes so fast. Sometimes it is, however, useful to stop and consider what we can learn from history in our own fields. This does not apply only to those who do this professionally, with the methods and persistence of historians. All of us can benefit from visits to the past. One of the most significant benefits of such a pause from the frenetic pursuit of the next thing in the future is an improved understanding of our tendency to reinvent the wheel.

In information systems (IS), systems development and systems analysis and design (SA&D) form one of the core focus areas in education and organizational practice. In IS, our particular focus is on the organization and management of the SA&D processes and the mechanisms required to ensure that the outcomes of the analysis, design, and development processes truly serve the needs of the organization that engages in the development effort.

SA&D methodologies have gone through a dramatic set of changes over the past several decades. If asked about this, most of us would identify a progression from lack of any structured methodologies to waterfall to more advanced plan-driven methods (such as the spiral model) to various types of agile with prototyping, Rapid Application Development, Joint Application Development, and others considered as "alternative development methodologies" somewhere along the standard path. In IS education, we are not quite sure yet how to deal with agile development and DevOps (a compound of "development" and "operations"), given the way they require a strong integration between planning, analysis, design, coding, deployment, and operations—coding and operations have never been among our strongest focus areas.

In the context of discussions regarding the development of SA&D, we have a tendency to trivialize particularly early plan-driven forms of SA&D and present them as pure waterfall model, consisting

ILLUSTRATION: ©IQONCEPT/FOTOLIA