Computer Science Curricula 2013

Strawman Draft

(February 2012)

The Joint Task Force on Computing Curricula
Association for Computing Machinery
IEEE-Computer Society

CS2013 Steering Committee

ACM Delegation

Mehran Sahami, *Chair* (Stanford University)
Andrea Danyluk (Williams College)
Sally Fincher (University of Kent)
Kathleen Fisher (Tufts University)
Dan Grossman (University of Washington)
Beth Hawthorne (Union County College)
Randy Katz (UC Berkeley)
Rich LeBlanc (Seattle University)
Dave Reed (Creighton University)

IEEE-CS Delegation

Steve Roach, *Chair* (Univ. of Texas, El Paso)
Ernesto Cuadros-Vargas (Univ. Catolica San Pablo)
Ronald Dodge (US Military Academy)
Robert France (Colorado State University)
Amruth Kumar (Ramapo Coll. of New Jersey)
Brian Robinson (ABB Corporation)
Remzi Seker (Univ. of Arkansas, Little Rock)
Alfred Thompson (Microsoft)

Table of Contents

Chapter 1: Introduction

Continuing a process that began over 40 years ago with the publication of Curriculum 68 [1], the major professional societies in computing—ACM and IEEE-Computer Society—have sponsored efforts to establish international curricular guidelines for undergraduate programs in computing on roughly a 10-year cycle. As the field of computing has grown and diversified, so too have the curricular recommendations, and there are now curricular volumes for Computer Engineering, Information Systems, Information Technology, and Software Engineering in addition to Computer Science [3]. These volumes are updated regularly with the aim of keeping computing curricula modern and relevant. The last complete Computer Science curricular volume was released in 2001 (CC2001) [2], and an interim review effort concluded in 2008 (CS2008) [4].

This volume, Computer Science Curricula 2013 (CS2013), represents a comprehensive revision. CS2013 redefines the knowledge units in CS, rethinking the essentials necessary for a Computer Science curriculum. It also seeks to identify exemplars of actual courses and programs to provide concrete guidance on curricular structure and development in a variety of institutional contexts.

The development of curricular guidelines for Computer Science is particularly challenging given the rapid evolution and expansion of the field: material dates fast. Moreover, the growing diversity of topics in Computer Science and the increasing integration of computing with other disciplines create additional challenges. Balancing topical growth with the need to keep recommendations realistic and implementable in the context of undergraduate education is particularly difficult. As a result, it is important to engage the broader computer science education community in a dialog to better understand new opportunities, local needs, and to identify successful models of computing curriculum – whether established or novel. One aim of this Strawman report is to provide the basis for such engagement, by providing an early draft of the CS2013 volume that can be scrutinized by members of the computing community with the goal of augmenting and refining the final report.

Charter

The ACM and IEEE-Computer Society chartered the CS2013 effort with the following directive:

> *To review the Joint ACM and IEEE-CS Computer Science volume of Computing Curricula 2001 and the accompanying interim review CS 2008, and develop a revised and enhanced version for the year 2013 that will match the latest developments in the discipline and have lasting impact.*
>
> *The CS2013 task force will seek input from a diverse audience with the goal of broadening participation in computer science. The report will seek to be international in scope and offer curricular and pedagogical guidance applicable to a wide range of institutions. The process of producing the final report will include multiple opportunities for public consultation and scrutiny.*

Consequently, the CS2013 task force welcomes review of, and comment on, this draft report.

High-level Themes

In developing CS2013, several high-level themes provided an overarching guide for this volume. These themes, which embody and reflect the CS2013 Principles (described in detail in another section of this volume) are:

- *The "Big Tent" view of CS.* As CS expands to include more cross-disciplinary work and new programs of the form "Computational Biology," "Computational Engineering," and "Computational X" are developed, it is important to embrace an outward-looking view that sees CS as a discipline actively seeking to work with and integrate into other disciplines.
- *Managing the size of the curriculum.* Although the field of Computer Science continues to grow unabated, it is not feasible to proportionately expand the size of the curriculum. As a result, CS2013 seeks to re-evaluate the essential topics in computing to make room for new topics without requiring more total instructional hours than the CS2008 guidelines. At the same time, the circumscription of curriculum size promotes more flexible models for curricula without losing the essence of a rigorous CS education.
- *Actual course exemplars.* CS2001 took on the significant challenge of providing descriptions of six *curriculum models* and forty-seven possible *course descriptions* variously incorporating the knowledge units as defined in that report. While this effort was valiant, in retrospect such course guidance did not seem to have much impact on actual course design. CS2013 plans to take a different approach: to identify and describe existing successful courses and curricula to show how relevant knowledge units are addressed and incorporated in actual programs.

- *Institutional needs.* CS2013 aims to be applicable in a broad range of geographic and cultural contexts, understanding that curricula exist within specific institutional needs, goals, and resource constraints. As a result, CS2013 allows for explicit flexibility in curricular structure through a tiered set of core topics, where a small set of Core-Tier 1 topics are considered essential for all CS programs, but individual programs choose their coverage of Core-Tier 2 topics. This tiered structure is described in more detail in Chapter 4 of this report.

Knowledge Areas

The CS2013 Body of Knowledge is organized into a set of 18 Knowledge Areas (KAs), corresponding to topical areas of study in computing. The Knowledge Areas are:

- AL - Algorithms and Complexity
- AR - Architecture and Organization
- CN - Computational Science
- DS - Discrete Structures
- GV - Graphics and Visual Computing
- HC - Human-Computer Interaction
- IAS - Information Assurance and Security
- IM - Information Management
- IS - Intelligent Systems
- NC - Networking and Communications
- OS - Operating Systems
- PBD - Platform-based Development
- PD - Parallel and Distributed Computing
- PL - Programming Languages
- SDF - Software Development Fundamentals
- SE - Software Engineering
- SF - Systems Fundamentals
- SP - Social and Professional Issues

Many of these Knowledge Areas are derived from CC2001/CS2008 but have been revised—in some cases quite significantly—in CS2013; others are new. There are three major causes of KA change: the reorganization of existing KAs, the development of cross-cutting KAs, and the creation of entirely new KAs. Reorganized KAs are a refactoring of existing topics to better reflect coherent units of knowledge as the field of Computer Science has evolved. For example, Software Development Fundamentals is a significant reorganization of the previous Programming Fundamentals KA. Cross-cutting KAs are a refactoring of existing KAs that extract and integrates cross-cutting foundational topics into their own KA rather than duplicating them across many others. Examples include SF-System Fundamentals and IAS-Information Assurance and Security. Finally, new KAs reflect emerging topics in CS that have become sufficiently prevalent to be included in the volume. PBD-Platform-based Development is an example of such a KA. Chapter 5 contains a more comprehensive overview of these changes.

Previous Input

To lay the groundwork for CS2013, we conducted a survey of the usage of the CC2001 and CS2008 volumes. The survey was sent to approximately 1500 Computer Science (and related discipline) Department Chairs and Directors of Undergraduate Studies in the United States and an additional 2000 Department Chairs internationally. We received 201 responses, representing a wide range of institutions (self-identified):

- research-oriented universities (55%)
- teaching-oriented universities (17.5%)
- undergraduate-only colleges (22.5%)
- community colleges (5%)

The institutions also varied considerably in size, with the following distribution:

- less than 1,000 students (6.5%)
- 1,000 to 5,000 students (30%)
- 5,000 to 10,000 students (19%)
- more than 10,000 students (44.5%)

In examining the *usage* of the CC2001/CS2008 reports, survey respondents reported that the Body of Knowledge (i.e., the outline of topics that should appear in undergraduate Computer Science curricula) was the most used aspect. When questioned about new topical areas that should be added to the Body of Knowledge, survey respondents indicated a strong need to add the topics of *Security* as well as *Parallel and Distributed Computing*. Indeed, feedback during the CS2008 review had also indicated the importance of these two areas, but the CS2008 steering committee had felt that creating new KAs was beyond their purview and deferred the development of those areas to the next full curricular report. CS2013 includes these two new KAs (among others): *Information Assurance and Security*, and *Parallel and Distributed Computing*.

Coming Attractions in CS2013

The final version of the CS2013 volume is, naturally enough, scheduled for release in 2013. Hence, this Strawman draft is—by design—incomplete. Not only will the final report include revisions of the Body of Knowledge presented here, based on community feedback, it will also include several sections which do not yet exist. Here we provide a timeline for CS2013 efforts and outline some of the "coming attractions" (i.e., additional sections) that are planned for inclusion in future drafts.

Timeline

The 2013 curricular guidelines will comprise several sorts of materials: the Body of Knowledge, Exemplars of Curricula and Courses, Professional Practice, and Institutional Challenges. These are being developed in offset phases, starting with the Body of Knowledge.

A summary of the CS2013 timeline is as follows:

Fall 2010:	CS2013 chartered and effort begins
February 2011:	CS2013 Principles outlined and Body of Knowledge revision begins
February 2012:	CS2013 Strawman report released Includes: Body of Knowledge, Characteristics of Graduates
July 15, 2012:	Comment period for Strawman draft closes
February 2013:	CS2013 Ironman report planned for release Includes: Body of Knowledge, Characteristics of Graduates, Curricula and Course Exemplars, Professional Practice, Institutional Challenges
June 2013:	Comment period for Ironman draft closes
Summer 2013:	CS2013 Final report planned for release

Exemplars of Curricula and Courses

Perhaps the most significant section of the CS2013 final report that is not included in the Strawman draft is the presentation of actual curricula and courses that embody the topics in the CS2013 Body of Knowledge. The CS2013 Ironman draft will include examples used in practice—from a variety of universities and colleges—to illustrate how topics in the Knowledge Areas may be covered and combined in diverse ways.

Importantly, we believe that the identification of such exemplary courses and curricula provides a tremendous opportunity for further community involvement in the development of the CS2013 volume. We invite members of the computing community to contribute courses and curricula

from their own institutions (or other institutions that they may be familiar with). Those interested in potentially mapping courses/curricula to the CS2013 Body of Knowledge are encouraged to contact members of the CS2013 steering committee for more details.

Professional Practice

The education that undergraduates in Computer Science receive must adequately prepare them for the workforce in a more holistic way than simply conveying technical facts. Indeed, "soft skills" (such as teamwork and communication) and personal attributes (such as identification of opportunity and risk) play a critical role in the workplace. Successfully applying technical knowledge in practice often requires an ability to tolerate ambiguity and work well with others from different backgrounds and disciplines. These overarching considerations are important for promoting successful professional practice in a variety of career paths. We will include suggestions for, and examples of, ways in which curricula encourage the development of such skills, including professional competencies and entrepreneurship, as part of an undergraduate Computer Science program in the CS2013 Ironman draft.

Institutional Challenges

CS departments and programs often face institutional challenges in implementing a curriculum: they may have too few faculty to cover all the knowledge areas, insufficient number of students for a full program, and/or inadequate institutional resource for professional development. This section will identify such challenges and provide suggestions for their amelioration.

Opportunities for Involvement

We believe it is essential for endeavours of this kind to engage the broad computing community to review and critique successive drafts. To this end, the development of this Strawman report has already benefited from the input of more than 100 contributors beyond the steering committee. We welcome further community engagement on this effort in multiple ways, including (but not limited to):

- Comments on the Strawman draft, especially with respect to the Body of Knowledge.
- Contribution of exemplar courses/curricula that are mapped against the Body of Knowledge.
- Descriptions of pedagogic approaches and instructional designs (both time-tested and novel) that address professional practice within undergraduate curricula.
- Sharing of institutional challenges, and solutions to them.

Comments on all aspects of this report are welcome and encouraged via the CS2013 website:

http://cs2013.org

References

[1] ACM Curriculum Committee on Computer Science. 1968. Curriculum 68: Recommendations for Academic Programs in Computer Science. *Comm. ACM* 11, 3 (Mar. 1968), 151-197.

[2] ACM/IEEE-CS Joint Task Force on Computing Curricula. 2001. ACM/IEEE Computing Curricula 2001 Final Report. http://www.acm.org/sigcse/cc2001.

[3] ACM/IEEE-CS Joint Task Force for Computer Curricula 2005. Computing Curricula 2005: An Overview Report. http://www.acm.org/education/curric_vols/CC2005-March06Final.pdf

[4] ACM/IEEE-CS Joint Interim Review Task Force. 2008. Computer Science Curriculum 2008: An Interim Revision of CS 2001, Report from the Interim Review Task Force. http://www.acm.org/education/curricula/ComputerScience2008.pdf

Acknowledgments

The CS2013 Strawman report has benefited from the input of many individuals, including: Alex Aiken (Stanford University), Ross Anderson (Cambridge University), Florence Appel (Saint Xavier University), Helen Armstrong (Curtin university), Colin Armstrong (Curtin university), Krste Asanovic (UC Berkeley), Radu F. Babiceanu (University of Arkansas at Little Rock), Mike Barker (Massachusetts Institute of Technology), Michael Barker (Nara Institute of Science and Technology), Paul Beame (University of Washington), Robert Beck (VIllanova University), Matt Bishop (University of California, Davis), Alan Blackwell (Cambridge University), Don Blaheta (Longwood University), Olivier Bonaventure (Universite Catholique de Louvain), Roger Boyle (University of Leeds), Clay Breshears (Intel), Bo Brinkman (Miami University), David Broman (Linkoping University), Kim Bruce (Pomona College), Jonathan Buss (University of Waterloo), Netiva Caftori (Northeastern Illinois University, Chicago), Paul Cairns (University of York), Alison Clear (Christchurch Polytechnic Institute of Technology), Curt Clifton (Rose-Hulman and The Omni Group), Yvonne Cody (University of Victoria), Tony Cowling (University of Shefffield), Joyce Currie-Little (Towson University), Ron Cytron (Washington University in St. Louis), Melissa Dark (Purdue University), Janet Davis (Grinnell College), Marie DesJardins (University of Maryland, Baltimore County), Zachary Dodds (Harvey Mudd College), Paul Dourish (University of California, Irvine), Lynette Drevin (North-West Universit), Scot Drysdale (Dartmouth College), Kathi Fisler (Worcester Polytechnic Institute), Susan Fox (Macalester College), Edward Fox (Virginia Tech), Eric Freudenthal (University of Texas El Paso), Stephen Freund (Williams College), Lynn Futcher (Nelson Mandela Metropolitan University), Greg Gagne (Wesminister College), Dan Garcia (UC Berkeley), Judy Gersting (Indiana University-Purdue University Indianapolis), Yolanda Gil (University of Southern California), Michael Gleicher (University Wisconsin, Madison), Frances Grodzinsky (Sacred Heart University), Anshul Gupta (IBM), Mark Guzdial (Georgia Tech), Brian Hay (University of Alaska, Fairbanks), Brian Henderson-Sellers (University of Technology, Sydney), Matthew Hertz (Canisius College), Tom Hilburn (Embry-Riddle Aeronautical University), Tony Hosking (Purdue University), Johan Jeuring (Utrecht University), Yiming Ji (University of South Carolina Beaufort), Maggie Johnson (Google), Matt Jones (Swansea University), Frans Kaashoek (MIT), Lisa Kaczmarczyk (ACM Education Council), Jennifer Kay (Rowan

University), Scott Klemmer (Stanford University), Jim Kurose (University of Massachusetts, Amherst), Doug Lea (SUNY Oswego), Terry Linkletter (Central Washington University), David Lubke (NVIDIA), Bill Manaris (College of Charleston), Samuel Mann (Otago Polytechnic), C. Diane Martin (George Washington University), Andrew McGettrick (University of Strathclyde), Morgan Mcguire (Williams College), Keith Miller (University of Illinois at Springfield), Narayan Murthy (Pace University), Kara Nance (University of Alaska, Fairbanks), Todd Neller (Gettysburg College), Reece Newman (Sinclair Community College), Christine Nickell (Information Assurance Center for Computer Network Operations, CyberSecurity, and Information Assurance), James Noble (Victoria University of Wellington), Peter Norvig (Google), Joseph O'Rourke (Smith College), Jens Palsberg (UCLA), Robert Panoff (Shodor.org), Sushil Prasad (Georgia State University), Michael Quinn (Seattle University), Matt Ratto (University of Toronto), Penny Rheingans (U. Maryland Baltimore County), Carols Rieder (Lucerne University of Applied Sciences), Eric Roberts (Stanford University), Arny Rosenberg (Northeastern and Colorado State University), Ingrid Russell (University of Hartford), Dino Schweitzer (United States Air Force Academy), Michael Scott (University of Rochester), Robert Sedgewick (Princeton University), Helen Sharp (Open Univeristy), Robert Sloan (University of Illinois, Chicago), Ann Sobel (Miami University), Carol Spradling (Northwest Missouri State University), Michelle Strout (Colorado State University), Alan Sussman (University of Maryland, College Park), Blair Taylor (Towson University), Simon Thompson (University of Kent), Johan Vanniekerk (Nelson Mandela Metropolitan University), Christoph von Praun (Georg-Simon-Ohm Hochschule Nürnberg), Rossouw Von Solms (Nelson Mandela Metropolitan University), John Wawrzynek (UC Berkeley), Charles Weems (Univ. of Massachusettes, Amherst), David Wetherall (University of Washington), Michael Wrinn (Intel)

Additionally, review of various portions of the Strawman report took part in several venues, including: the 42nd ACM Technical Symposium of the Special Interest Group on Computer Science Education (SIGCSE-11), the 24th IEEE-CS Conference on Software Engineering Education and Training (CSEET-11), the 2011 IEEE Frontiers in Education Conference (FIE-11), the 2011 Federated Computing Research Conference (FCRC-11), the 2nd Symposium on Educational Advances in Artificial Intelligence (EAAI-11), the Conference of ACM Special Interest Group on Data Communication 2011 (SIGCOMM-11), the 2011 IEEE International Joint Conference on Computer, Information, and Systems Sciences and Engineering (CISSE-11),

the 2011 Systems, Programming, Languages and Applications: Software for Humanity Conference (SPLASH-11), the 15th Colloquium for Information Systems Security Education, the 2011 National Centers of Academic Excellence in IA Education (CAE/IAE) Principles meeting, and the 7th IFIP TC 11.8 World Conference on Information Security Education (WISE).

Several more conference special sessions to review and comment on drafts of CS2013 are planned for the coming year, including 43rd ACM Technical Symposium of the Special Interest Group on Computer Science Education (SIGCSE-12), the Special Session of the Special Interest Group on Computers and Society at SIGCSE-12, Computer Research Association Snowbird Conference 2012, and the 2012 IEEE Frontiers in Education Conference (FIE-12), among others.

A number of organizations also provided valuable feedback to the CS2013 Strawman effort, including: the ACM Education Board and Council, the IEEE-CS Educational Activities Board, the ACM SIGPLAN Education Board, the ACM Special Interest Group Computers and Society, and the NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing Committee

Chapter 2: Principles

Early in its work, the 2013 Steering Committee agreed upon a set of principles to guide the development of this volume. The principles adopted for CS2013 overlap significantly with the principles adopted for previous curricular efforts, most notably CC2001 and CS2008. As with previous ACM/IEEE curricula volumes, there are a variety of constituencies for CS2013, including individual faculty members and instructors at a wide range of colleges, universities, and technical schools on any of six continents; CS programs and the departments, colleges, and institutions where they are housed; accreditation and certification boards; authors; and researchers. Other constituencies include pre-college preparatory schools and advanced placement curricula as well as graduate programs in computer science.

The principles were developed in consideration of these constituencies, as well as issues related to student outcomes, development of curricula, and the review process. The order of presentation is not intended to imply relative importance.

1. *Computer Science curricula should be designed to provide students with the flexibility to work across many disciplines.* Computing is a broad field that connects to and draws from many disciplines, including mathematics, electrical and systems engineering, psychology, statistics, fine arts, linguistics, and physical and life sciences. Computer Science students should develop the flexibility to work across disciplines.

2. *Computer Science curricula should be designed to prepare graduates for a variety of professions, attracting the full range of talent to the field.* Computer Science impacts nearly every modern endeavour. CS2013 takes a broad view of the field that includes topics such as "computational-x" (e.g., computational finance or computational chemistry) and "x-informatics" (e.g., eco-informatics or bio-informatics). Well-rounded CS graduates will have a balance of theory and application, as described in Chapter 3: Characteristics of Graduates.

3. *CS2013 should provide guidance for the expected level of mastery of topics by graduates*. It should suggest outcomes indicating the intended level of mastery and provide exemplars of fielded curricula covering topics in the Body of Knowledge.

4. *CS 2013 must provide realistic, adoptable recommendations that provide guidance and flexibility, allowing curricular designs that are innovative and track recent developments in the field.* The guidelines are intended to provide clear, implementable goals, while also providing the flexibility that programs need in order to respond to a rapidly changing field. CS2013 is intended as guidance, not as a minimal standard against which to evaluate a program.

5. *The CS2013 guidelines must be relevant to a variety of institutions.* Given the wide range of institutions and programs (including 2-year, 3-year, and 4-year programs; liberal arts, technological, and research institutions; and institutions of every size), it is neither possible nor desirable for these guidelines to dictate curricula for computing. Individual programs will need to evaluate their constraints and environments to construct curricula.

6. *The size of the essential knowledge must be managed*. While the range of relevant topics has expanded, the size of undergraduate curricula has not. Thus, CS2013 must carefully choose among topics and recommend the essential elements.

7. *Computer Science curricula should be designed to prepare graduates to succeed in a rapidly changing field.* Computer Science is rapidly changing and will continue to change for the foreseeable future. Curricula must prepare students for lifelong learning and must include professional practice (e.g. communication skills, teamwork, ethics) as components of the undergraduate experience. Computer science students must learn to integrate theory and practice, to recognize the importance of abstraction, and to appreciate the value of good engineering design.

8. *CS2013 should identify the fundamental skills and knowledge that all computer science graduates should possess while providing the greatest flexibility in selecting topics.* To this end, we have introduced three levels of knowledge description: Tier-1 Core, Tier-2 Core, and Elective. For a full discussion of Tier-1 Core, Tier-2 Core, and Elective, see Chapter 4: Completing the Curriculum.

9. *CS2013 should provide the greatest flexibility in organizing topics into courses and curricula.* Knowledge areas are not intended to describe specific courses. There are many

novel, interesting, and effective ways to combine topics from the Body of Knowledge into courses.

10. *The development and review of CS2013 must be broadly based.* The CS2013 Task Force must include participation from many different constituencies including industry, government, and the full range of higher education institutions involved in computer science education. It must take into account relevant feedback from these constituencies.

Chapter 3: Characteristics of Graduates

Graduates of Computer Science programs should have fundamental competency in the areas described by the Body of Knowledge (see Chapter 5), particularly the core topics contained there. However, there are also competences that graduates of CS programs should have that are not explicitly listed in the Body of Knowledge. Professionals in the field typically embody a characteristic style of thinking and problem solving, a style that emerges from the experiences obtained through study of the field and professional practice. Below, we describe the characteristics that we believe should be met at least at an elementary level by graduates of computer science programs. These characteristics will enable their success in the field and further professional development. Some of these characteristics and skills also apply to other fields. They are included here because the development of these skills and characteristics must be explicitly addressed and encouraged by Computer Science programs.

This list is based on a similar list in CC2001 and CS2008. The substantial changes that led to this new version were influenced by responses to a survey conducted by the CS2013 Steering Committee.

At a broad level, the expected characteristics of computer science graduates include the following:

Technical understanding of Computer Science

Graduates should have a mastery of computer science as described by the core of the Body of Knowledge.

Familiarity with common themes and principles

Graduates need understanding of a number of recurring themes, such as abstraction, complexity, and evolutionary change, and a set of general principles, such as sharing a common resource, security, and concurrency. Graduates should recognize that these themes and principles have broad application to the field of computer science and should not consider them as relevant only to the domains in which they were introduced.

Appreciation of the interplay between theory and practice

A fundamental aspect of computer science is understanding the interplay between theory and practice and the essential links between them. Graduates of a computer science program need to understand how theory and practice influence each other.

System-level perspective

Graduates of a computer science program need to think at multiple levels of detail and abstraction. This understanding should transcend the implementation details of the various components to encompass an appreciation for the structure of computer systems and the processes involved in their construction and analysis. They need to recognize the context in which a computer system may function, including its interactions with people and the physical world.

Problem solving skills

Graduates need to understand how to apply the knowledge they have gained to solve real problems, not just write code and move bits. They should also realize that there are multiple solutions to a given problem and that selecting among them is not a purely technical activity, as these solutions will have a real impact on people's lives. Graduates also should be able to communicate their solution to others, including why and how a solution solves the problem and what assumptions were made.

Project experience

To ensure that graduates can successfully apply the knowledge they have gained, all graduates of computer science programs should have been involved in at least one substantial project. In most cases, this experience will be a software development project, but other experiences are also appropriate in particular circumstances. Such projects should challenge students by being integrative, requiring evaluation of potential solutions, and requiring work on a larger scale than typical course projects. Students should have opportunities to develop their interpersonal communication skills as part of their project experience.

Commitment to life-long learning

Graduates of a computer science program should realize that the computing field advances at a rapid pace. Specific languages and technology platforms change over time. Therefore, graduates need to realize that they must continue to learn and adapt their skills throughout their careers. To develop this ability, students should be exposed to multiple programming languages, tools, and technologies as well as the fundamental underlying principles throughout their education.

Commitment to professional responsibility

Graduates should recognize the social, legal, ethical and cultural issues involved in the deployment and use of computer technology. They should respond to these issues from an informed perspective, guided by personal and professional principles. They must further recognize that social, legal, and ethical standards vary internationally.

Communication and organizational skills

Graduates should have the ability to make succinct presentations to a range of audiences about technical problems and their solutions. This may involve face-to-face, written, or electronic communication. They should be prepared to work effectively as members of teams. Graduates should be able to manage their own learning and development, including managing time, priorities, and progress.

Awareness of the broad applicability of computing

Platforms range from embedded micro-sensors to high-performance clusters and distributed clouds. Computer applications impact nearly every aspect of modern life. Graduates should understand the full range of opportunities available in computing.

Appreciation of domain-specific knowledge

Graduates should understand that computing interacts with many different domains. Solutions to many problems require both computing skills and domain knowledge. Therefore, graduates need to be able to communicate with, and learn from, experts from different domains throughout their careers.

Chapter 4: Constructing a Complete Curriculum

This chapter provides high-level guidelines on how to use the Body of Knowledge to create an institution's undergraduate curriculum in computer science. It does not propose a particular set of courses or curriculum structure -- that is the role of the (forthcoming) course/curriculum exemplars. Rather, this chapter emphasizes the flexibility that the Body of Knowledge allows in adapting curricula to institutional needs and the continual evolution of the field. In computer-science terms, one can view the Body of Knowledge as a specification of content to cover and a curriculum as an implementation. A large variety of curricula can meet the specification.

The following points are elaborated:

- Knowledge Areas are not intended to be in one-to-one correspondence with particular courses in a curriculum: We expect curricula will have courses incorporating topics from multiple Knowledge Areas.
- Topics are identified as either "core" or "elective" with the core further subdivided into "tier-1" and "tier-2."
    - A curriculum should include all topics in the tier-1 core and ensure that all students cover this material.
    - A curriculum should include all or almost all topics in the tier-2 core and ensure that all students cover the vast majority of this material.
    - A curriculum should include significant elective material: Covering only "core" topics is insufficient for a complete curriculum.
- Because it is a hierarchical outline, the Body of Knowledge under-emphasizes some key issues that must be considered when constructing a curriculum.

Knowledge Areas are Not Necessarily Courses (and Important Examples Thereof)

It is naturally tempting to associate each Knowledge Area with a course. We explicitly discourage this practice in general, even though many curricula will have some courses containing material from only one Knowledge Area or, conversely, all the material from one Knowledge Area in one course. We view the hierarchical structure of the Body of Knowledge as a useful way to group related information, not as a stricture for organizing material into courses. Beyond this general flexibility, in several places we expect many curricula to integrate material from multiple Knowledge Areas, in particular:

- *Introductory courses:* There are diverse successful approaches to introductory courses in computer science. Many focus on the topics in Software Development Fundamentals together with a subset of the topics in Programming Languages or Software Engineering, while leaving most of the topics in these other Knowledge Areas to advanced courses. But *which* topics from other Knowledge Areas are covered in introductory courses can vary. Some courses use object-oriented programming, others functional programming, others platform-based development (thereby covering topics in the Platform-Based Development Knowledge Area), etc. Conversely, there is no requirement that all Software Development Fundamentals be covered in a first or second course, though in practice most topics will usually be covered in these early courses.

- *Systems courses:* The topics in the Systems Fundamentals Knowledge Area can be covered in courses designed to cover general systems principles or in courses devoted to particular systems areas such as computer architecture, operating systems, networking, or distributed systems. For example, an Operating Systems course might spend considerable time on topics of more general use, such as low-level programming, concurrency and synchronization, performance measurement, or computer security. Such courses may draw on material in several Knowledge Areas. Certain fundamental systems topics like latency or parallelism will likely arise in many places in a curriculum. While it is important that such topics do arise, preferably in multiple settings, the Body of Knowledge does not specify the particular settings in which to teach such topics.

- *Parallel computing:* Among the many changes to the Body of Knowledge compared to previous reports is a new Knowledge Area in Parallel and Distributed Computing. An alternative structure for the Body of Knowledge would place relevant topics in other Knowledge Areas: parallel algorithms with algorithms, programming constructs in software-development focused areas, multi-core design with computer architecture, and so forth. We chose instead to provide guidance on the essential parallelism topics in one place. Some, but not all, curricula will likely have courses dedicated to parallelism, at least in the near term.

Tier-1 Core, Tier-2 Core, Elective: What These Terms Mean, What is Required

As described at the beginning of this chapter, computer science curricula should cover all of the core tier-1 topics, all or almost all of the core tier-2 topics, and significant depth in many of the elective topics (i.e., the core is not sufficient for an undergraduate degree in computer science). Here we provide additional perspective on what "tier-1 core," "tier-2 core", and "elective" mean, including motivation for these distinctions.

Motivation for subdividing the core: Earlier versions of the ACM/IEEE Computer Science Curricula had only "core" and "elective" with every topic in the former being required. We departed from this strict interpretation of "everything in the core must be taught to every student" for these reasons:

- It did not sufficiently reflect reality: Many strong computer science curricula were missing at least one hour of core material. It is misleading to suggest that such curricula are outside the definition of an undergraduate degree in computer science.
- As the field has grown, there is ever-increasing pressure to grow the core and allow students to specialize in areas of interest. Doing so simply becomes impossible within the short time-frame of an undergraduate degree. Providing some flexibility on coverage of core topics enables curricula and students to specialize if they choose to do so.

Conversely, we could have allowed for *any* core topic to be skipped provided that the vast majority was part of every student's education. By retaining a smaller tier-1 core of required

material, we provide additional guidance and structure for curriculum designers. In the tier-1 core are the topics that are fundamental to the structure of any computer-science program.

On the meaning of tier-1: A tier-1 topic should be a required part of every computer-science curriculum for every student. This is not to say that tier-2 or even elective topics should not be, but the tier-1 topics are those with widespread consensus for inclusion. Moreover, at least preliminary treatment of most of these topics typically comes in the first two years of a curriculum, precisely because so much of the field relies on these topics. However, introductory courses need not cover all tier-1 material and will usually draw on tier-2 and elective material as well.

On the meaning of tier-2: Tier-2 topics are generally essential in an undergraduate computer-science degree. Requiring the vast majority of them is a *minimum* expectation, and we encourage institutions to cover all of them for every student. That said, computer science programs can allow students to focus in certain areas in which some tier-2 topics are not required. We also acknowledge that resource constraints, such as a small number of faculty or institutional limits on degree requirements, may make it prohibitively difficult to cover every topic in the core while still providing advanced elective material. **A computer-science curriculum should aim to cover 90-100% of the tier-2 topics for every student, with 80% considered as a minimum.**

There is no expectation that tier-1 topics necessarily precede tier-2 topics in a curriculum. In particular, we expect introductory courses will draw on both tier-1 and tier-2 (and possibly elective) material and that some core material will be delayed until later courses.

On the meaning of elective: A program covering only core material would provide insufficient breadth and depth in computer science, but most programs will not cover all the elective material in the Body of Knowledge and certainly few, if any, students will cover all of it within an undergraduate program. Conversely, the Body of Knowledge is by no means exhaustive, and advanced courses may often go beyond the topics and learning outcomes contained in it. Nonetheless, the Body of Knowledge provides a useful guide on material appropriate for a computer-science undergraduate degree, and all students of computer science should deepen their understanding in multiple areas via the elective topics.

A curriculum may well require material designated elective in the Body of Knowledge. Many curricula, especially those with a particular focus, will require some elective topics, by virtue of them being covered in required courses.

The size of the core: The size of the core (tier-1 plus tier-2) is a few hours larger than in previous versions of the computer-science curriculum, but this is counterbalanced by our more flexible treatment of the core. As a result, we are not increasing the number of required courses a curriculum should need. Indeed, a curriculum covering 90% of the tier-2 hours would have the same number of core hours as a curriculum covering the core in the CS2008 volume, and a curriculum covering 80% of the tier-2 hours would have fewer core hours than even a curriculum covering the core in the CC2001 volume (the core grew from 2001 to 2008).

A note on balance: Computer science is an elegant interplay of theory, software, hardware, and applications. The core in general and the tier-1 core in particular, when viewed in isolation, may seem to focus on programming, discrete structures, and algorithms. This focus results from the fact that these topics typically come early in a curriculum so that advanced courses can use them as pre-requisites. Essential experience with systems and applications can be achieved in more disparate ways using elective material in the Body of Knowledge. Because all curricula will include appropriate elective material, an overall curriculum can and should achieve an appropriate balance.

Further Considerations

As useful as the Body of Knowledge is, it is important to complement it with a thoughtful understanding of cross-cutting themes in a curriculum, the "big ideas" of computer science. In designing a curriculum, it is also valuable to identify curriculum-wide objectives, for which the Principles and the Characteristics of Graduates chapters of this volume should prove useful.

In the last few years, two on-going trends have had deep effects on many curricula. First, the continuing growth of computer science has led to many programs organizing their curricula to allow for *intradisciplinary* specialization (using terms such as threads, tracks, vectors, etc.). Second, the importance of computing to almost every other field has increasingly led to the creation of *interdisciplinary* programs (joint majors, double majors, etc.) and incorporating interdisciplinary material into computer-science programs. We applaud both trends and believe

a flexible Body of Knowledge, including a flexible core, support them. Conversely, such specialization is not required: Many programs will continue to offer a broad yet thorough coverage of computer science as a distinct and coherent discipline.

Chapter 5: Introduction to the Body of Knowledge

Process for Updating the Body of Knowledge

The CS2013 Steering Committee constituted a subcommittee for each KA, chaired by a member of the Steering Committee, and initially including at least two other members of the Steering Committee. Individual subcommittee Chairs then invited expert members (outside the CS2013 Steering Committee) to join the work of defining and reviewing each KA; drafts of KAs were also presented in various conference panel and special session presentations. The KA subcommittee Chairs (as members of the CS2013 Steering Committee) worked to resolve conflicts, eliminate redundancies and appropriately categorize and cross-reference topics between the various KAs. This year-long process ultimately converged to the draft version of the Body of Knowledge presented here.

As noted in the introduction to this report, we are soliciting continued community feedback which will be considered and incorporated into future drafts of the CS2013 report.

The CS2013 Body of Knowledge is presented as a set of Knowledge Areas (KAs), organized on topical themes rather than by course boundaries. Each KA is further organized into a set of Knowledge Units (KUs), which are summarized in a table at the head of each KA section. We expect that the topics within the KAs will be organized into courses in different ways at different institutions.

Here, we provide background for understanding how to read the Body of Knowledge, and we give an overview of the number of core hours in each KA. We also highlight the KAs that have significant cross-topic components and those that are new to this volume. Chapter 4 presents essential background on how the Body of Knowledge translates into actual curricula.

Overview of New Knowledge Areas

While computer science encompasses technologies that change rapidly over time, it is defined by essential concepts, perspectives, and methodologies that are constant. As a result, much of the core Body of Knowledge remains unchanged from earlier curricular volumes. However, new developments in computing technology and pedagogy mean that some aspects of the core evolve over time, and some of the previous structures and organization may no longer be appropriate for describing the discipline. As a result, CS2013 has modified the organization of the curriculum in various ways, adding some new KAs and restructuring others. We highlight these changes in the remainder of this section.

IAS-Information Assurance and Security

IAS is a new KA in recognition of the world's reliance on information technology and its critical role in computer science education. IAS as a domain is the set of controls and processes, both technical and policy, intended to protect and defend information and information systems. IAS draws together topics that are pervasive throughout other KAs. Topics germane to *only* IAS are presented in depth in this KA, whereas other topics are noted and cross referenced to the KAs that contain them. As such, this KA is prefaced with a detailed table of cross-references to other KAs.

NC-Networking and Communication

CC2001 introduced a KA entitled "Net-Centric Computing" which encompassed a combination of topics including traditional networking, web development, and network security. Given the growth and divergence in these topics since the last report, we renamed and refactored this KA to focus specifically on topics in networking and communication. Discussions of web applications and mobile device development are now covered in the new PBD-Platform-Based Development KA. Security is covered in the new IAS-Information Assurance and Security KA.

PBD-Platform-Based Development

PBD is a new KA that recognizes the increasing use of platform-specific programming environments, both at the introductory level and in upper-level electives. Platforms such as the Web or mobile devices enable students to learn within and about environments constrained by hardware, APIs, and special services (often in cross-disciplinary contexts). These environments are sufficiently different from "general purpose" programming to warrant this new (wholly elective) KA.

PD-Parallel and Distributed Computing

Previous curricular volumes had parallelism topics distributed across disparate KAs as electives. Given the vastly increased importance of parallel and distributed computing, it seemed crucial to identify essential concepts in this area and to promote those topics to the core. To highlight and coordinate this material, CS2013 dedicates a KA to this area. This new KA includes material on programming models, programming pragmatics, algorithms, performance, computer architecture, and distributed systems.

SDF-Software Development Fundamentals

This new KA generalizes introductory programming to focus on the entire software development process, identifying concepts and skills that should be mastered in the first year of a computer science program. As a result of its broad purpose, the SDF KA includes fundamental concepts and skills that could appear in other software-oriented KAs (e.g., programming constructs from Programming Languages, simple algorithm analysis from Algorithms and Complexity, simple development methodologies from Software Engineering). Likewise, each of those KAs will contain more advanced material that builds upon the fundamental concepts and skills in SDF. Compared to previous volumes, key approaches to programming -- including object-oriented programming, functional programming, and event-driven programming -- are kept in one place, namely the PL KA, even though many curricula will cover some of these topics in introductory courses.

SF-Systems Fundamentals

In previous curricular volumes, the interacting layers of a typical computing system, from hardware building blocks, to architectural organization, to operating system services, to application execution environments (particularly for parallel execution in a modern view of applications), were presented in independent knowledge units. The new Systems Fundamentals KA presents a unified systems perspective and common conceptual foundation for other KAs (notably Architecture and Organization, Network and Communications, Operating Systems, and Parallel and Distributed Algorithms). An organizational principle is "programming for performance": what a programmer needs to understand about the underlying system to achieve high performance, particularly in terms of exploiting parallelism.

How to Read the Body of Knowledge

Curricular Hours

Continuing in the tradition of CC2001/CS2008, we define the unit of coverage in the Body of Knowledge in terms of **lecture hours**, as being the sole unit that is understandable in (and transferable to) cross-cultural contexts. An "hour" corresponds to the time required to present the material in a traditional lecture-oriented format; the hour count does not include any additional work that is associated with a lecture (e.g., in self-study, lab classes, assessments, etc.). Indeed, we expect students to spend a significant amount of additional time outside of class developing facility with the material presented in class. As with previous reports, we maintain the principle that the use of a lecture-hour as the unit of measurement does not require or endorse the use of traditional lectures for the presentation of material.

The specification of topic hours represents the **minimum** amount of time we expect such coverage to take. Any institution may opt to cover the same material in a longer period of time as warranted by the individual needs of that institution.

Courses

Throughout the Body of Knowledge, when we refer to a "course" we mean an institutionally-recognised unit of study. Depending on local circumstance, full-time students will take several "courses" at any one time, typically eight or more per academic year. While "course" is a common term at some institutions, others will use other names, for example "module" or "paper".

Guidance on Learning Outcomes

Each KU within a KA lists both a set of topics and the learning outcomes students are expected to achieve with respect to the topics specified. Each learning outcome has a *level of mastery* associated with it. There are three levels of mastery, defined as:

- *Knowledge*: The student understands what a concept is or what it means. This level of mastery provides a basic awareness of a concept as opposed to expecting real facility with its application.
- *Application*: The student is able to apply a concept in a concrete way. Applying a concept may include, for example, the ability to implement a programming concept, use a particular proof technique, or perform a particular analysis.
- *Evaluation*: The student is able to consider a concept from multiple view points and/or justify the selection of a particular approach to solve a problem. This level of mastery implies more than the application of a concept; it involves the ability to select an appropriate approach from understood alternatives.

As a concrete, although admittedly simplistic, example of these levels of mastery, we consider the notion of iteration in software development, for example for-loops, while-loops, iterators. At the level of "Knowledge," a student would be expected to know what the concept of iteration is in software development and why it is a useful technique. In order to show mastery at the "Application" level, a student should be able to write a program using a form of iteration. Understanding iteration at the "Evaluation" level would require a student to understand multiple methods for iteration and be able to appropriately select among them for different applications.

Core Hours in Knowledge Areas

An overview of the number of core hours (both Tier1 and Tier2) by KA in the CS2013 Body of Knowledge is provided below (for a discussion of Tier1 and Tier2, see Chapter 4). For comparison, the number of core hours from both the previous CS2008 and CC2001 reports are provided as well.

Knowledge Area	CS2013 Tier1	CS2013 Tier2	CS2008 Core	CC2001 Core
AL-Algorithms and Complexity	19	9	31	31
AR-Architecture and Organization	0	16	36	36
CN-Computational Science	1	0	0	0
DS-Discrete Structures	37	4	43	43
GV-Graphics and Visual Computing	2	1	3	3
HC-Human-Computer Interaction	4	4	8	8
IAS-Security and Information Assurance	2	6	--	--
IM-Information Management	1	9	11	10
IS-Intelligent Systems	0	10	10	10
NC-Networking and Communication	3	7	15	15
OS-Operating Systems	4	11	18	18
PBD-Platform-based Development	0	0	--	--
PD-Parallel and Distributed Computing	5	10	--	--
PL-Programming Languages	8	20	21	21
SDF-Software Development Fundamentals	42	0	47	38
SE-Software Engineering	6	21	31	31
SF-Systems Fundamentals	18	9	--	--
SP-Social and Professional Issues	11	5	16	16
Total Core Hours	**163**	**142**	**290**	**280**

All Tier1 + All Tier2 Total	**305**
All Tier1 + 90% of Tier2 Total	**290.8**
All Tier1 + 80% of Tier2 Total	**276.6**

As seen above, in CS2013 the total Tier1 hours together with the entirety of Tier2 hours slightly exceeds the total core hours from previous reports. However, it is important to note that the tiered structure of the core in CS2013 explicitly provides the flexibility for institutions to select

topics from Tier2 (to include at least 80%). As a result, it is possible to implement the CS2013 guidelines with slightly fewer hours than previous curricular guidelines