Computing Curricula 2005

The Overview Report

covering undergraduate degree programs in

Computer Engineering
Computer Science
Information Systems
Information Technology
Software Engineering

A volume of the *Computing Curricula Series*

The Joint Task Force for Computing Curricula 2005

A cooperative project of
The Association for Computing Machinery (ACM)
The Association for Information Systems (AIS)
The Computer Society (IEEE-CS)

30 September 2005

Sponsoring Societies

This report was made possible by
financial support from the following societies:

ACM

IEEE Computer Society

Copyright Notice

The Joint Task Force for Computing Curricula 2005

Russell Shackelford is chair of the CC2005 Task Force. He is the previous chair of the ACM Education Board. In years past, he has served as Associate Chair of the Computer Science Department at Stanford University and as Director of Undergraduate Studies at Georgia Tech's College of Computing. He was co-chair of the CC2001 Task Force.

James H. Cross II is Philpott-Westpoint Stevens Professor and Chair of Computer Science and Software Engineering at Auburn University. He is a past Vice President of the IEEE Computer Society's Educational Activities Board (EAB). He was a member of the CC2001 Task Force.

Gordon Davies recently retired after forty years of teaching, the last twenty of which were at the U.K.'s Open University. In recent years, he helped create ACM's Professional Development Center. He is now actively involved in accreditation for the British Computer Society. He was a member of the CC2001 Task Force.

John Impagliazzo is Professor of Computer Science at Hofstra University. He chaired the Accreditation Committee of the ACM Education Board for twelve years. Currently, he is editor-in-chief of *Inroads - the SIGCSE Bulletin*, is chair of the IFIP Working Group 9.7 on the History of Computing, and is an active member and treasurer of the IEEE History Committee. He was a member of the CE2004 Task Force.

Reza Kamali is an Associate Professor and Department Head of Computer Information Systems and Information Technology at Purdue University Calumet, Hammond, Indiana. He was a founding member of SITE, which later became ACM's SIGITE. He now serves as SIGITE Education Officer. He is a member of the IT2006 Task Force.

Richard LeBlanc recently retired as Professor of Computer Science, College of Computing, Georgia Tech. He now serves as Vice President for Academic Affairs, Southern Catholic College. He is a past Chair and Vice Chair of the ACM Education Board, a member of IFIP Working Group 3.2 (Informatics Education at the University Level), a Team Chair for ABET's Computing Accreditation Commission, and a Software Engineering Program Evaluator for the Engineering Accreditation Commission. He was co-chair of the SE2004 Task Force.

Barry Lunt is Associate Professor of Information Technology at Brigham Young University. He was a founding member of SITE, which later became ACM's SIGITE. He is a member of the IEEE Computer Society, the IEEE Communication Society, and ASEE. He is chair of the IT2006 Task Force.

Andrew McGettrick is Professor of the Department of Computer and Information Sciences, University of Strathclyde, Glasgow, Co-chair of the ACM Education Board, and a Vice President of the British Computer Society. He recently chaired groups that created benchmark standards for undergraduate and Masters degree programs in Computing in the U.K. He was a member of the CC2001 Task Force, the CE2004 Task Force, and the SE2004 Task Force.

Robert Sloan is Associate Professor in the Computer Science Department of the University of Illinois at Chicago. He is an active member of the Educational Activities Board of the IEEE Computer Society. He was a member of the CC2001 Task Force and the CE2004 Task Force

Heikki Topi is Associate Professor of Computer Information Systems and Director of the MSIT program at Bentley College, Waltham, MA. He active in the Association for Information Systems (AIS) and is involved in curriculum development and accreditation activities within the North American IS community. He was a member of the IS2002 Task Force.

[This page intentionally left blank.]

Table of Contents

Summary

Computing has dramatically influenced progress in science, engineering, business, and many other avenues of human endeavor. In modern times, nearly everyone needs to use computers, and many will want to study computing in some form. Computing will continue to present challenging career opportunities, and those who work in computing will have a crucial role in shaping the future of society.

It is important for society that the computing disciplines attract quality students from a broad cross-section of the population and prepare them to be capable and responsible professionals, scientists and engineers. Over the years, professional and scientific computing societies based in the U.S. have taken a leading role in providing support for higher education in various ways, including the formulation of curriculum guidelines. Several reports that define and update guidelines for computing curricula have appeared over the past four decades. Recent efforts have targeted international participation, reflecting the need for the leading professional organizations to become truly global in scope and responsibility.

Early in the process that produced *Computing Curricula 2001 (CC2001)*, it became clear that the dramatic expansion of computing during the 1990s made it no longer reasonable to produce updated curriculum reports just for the disciplines for which reports existed previously. *CC2001* called for a set of reports to cover the growing family of computing-related disciplines, including a volume for each of *computer science, information systems, computer engineering*, and *software engineering*. It was also clear that new computing disciplines would emerge over time. Since the publication of *CC2001*, *information technology* has joined the family of computing disciplines and now requires its own curriculum volume.

The *CC2001* report also called for an *Overview Report* to summarize the content of the various discipline-specific reports. This document is the first edition of that *Overview Report*. Its goal is to provide perspective for those in academia who need to understand what the major computing disciplines are and how the respective undergraduate degree programs compare and complement each other. This report summarizes the body of knowledge for undergraduate programs in each of the major computing disciplines, highlights their commonalities and differences, and describes the performance characteristics of graduates from each kind of undergraduate degree program. To create this report, we have examined curriculum guidelines for undergraduate education and have referred to the computing professions and other supporting information as necessary. We have not focused on graduate education or on the identities of the computing research communities. College-level faculty, administrators, and other community leaders are the audience for this report. It outlines the issues and challenges they will face in shaping the undergraduate programs that will serve their constituents and their communities.

Following the publication of the *Overview Report*, the Joint Task Force will publish a shorter companion report, *The Guide to Undergraduate Programs in Computing.* The *Guide* will offer guidance to a broader audience, including prospective students, their parents and guidance counselors, and others who have reason to care about the choices that await students who move from high school to college. It will provide briefer characterizations of the computing disciplines, profile factors that students might consider when choosing an area of computing study, and will be widely distributed as an independent document.

This report is the result of an unprecedented cooperative effort among the leading computer societies and the major computing disciplines. It is based on inspection and analysis of the five discipline-specific other volumes of the *Computing Curricula Series*. Because most of those documents are oriented to higher education in the United States and Canada, this report is implicitly North American-centric. We expect future generations of all such volumes to be more international in scope. Until then, this report provides context that may help those in other nations know how to best use those reports in their context. Because things change rapidly in computing, the reports will require frequent updates. Electronic copies of the most recent edition of this and other computing curricula reports can be found at http://www.acm.org/education/curricula.html and at http://computer.org/curriculum.

[This page intentionally left blank.]

Chapter 1: Introduction

1.1. Purpose of this report

This report provides an overview of the different kinds of undergraduate degree programs in computing that are currently available and for which curriculum standards are now, or will soon be, available. Teachers, administrators, students, and parents need this report because computing is a broad discipline that crosses the boundaries between mathematics, science, engineering and business, and because computing embraces important competencies that lie at the foundation of professional practice. Computing consists of several disciplines. Many respected colleges and universities offer undergraduate degree programs in several of *computer science, computer engineering, information systems, information technology, software engineering*, and more. These computing disciplines are related, but are also quite different from each other. The variety of degree programs in computing presents students, educators, administrators, and other community leaders with important choices about where to focus their efforts.

Several questions naturally arise: *What are these different kinds of computing degree programs? How are they similar? How do they differ? How can I tell what their names really mean? Which kinds of programs should our local college offer?* And so on. These are all valid questions, but to anyone unfamiliar with the breadth of computing, the responses to these queries may be difficult to articulate. This report may help in articulating some answers.

We have created this report to explain the character of the various undergraduate degree programs in computing, and to help you determine which of the programs are most suited to particular goals and circumstances. We intend this report to serve a broad and varied audience. We think it can be helpful to:

- University faculty and administrators who are developing plans and curricula for computing-related programs at their institutions, and to those who guide the accreditation of such programs.
- Responsible parties in public education, including boards of education, government officials, elected representatives, and others who seek to represent the public interest.

In addition, we will soon be preparing *The Guide to Undergraduate Degree Programs in Computing* (henceforth the *Guide*). *The Guide* will be an independent companion document that will be broadly distributed to a more general audience. It is intended to serve:

- Students who are trying to determine which path of computing study fits their interests and goals
- Parents, teachers, guidance counselors, and others who are trying to assist students in their choices
- Professionals who are considering how to continue their education in a rapidly changing, dynamic field
- Anyone who is trying to make sense of the wide range of undergraduate degree programs in computing that are now available.

1.2. Scope of this report

There are many kinds of computing degree programs. Reliable information about the number of different degree program titles currently available is hard to come by, but it is generally accepted that over the last ten years or so, there has been a dramatic increase in the number and type of degree programs available to students. It is beyond both our goal and our capability to catalog and characterize them all. In this report, we focus on five that are prominent today: *computer engineering, computer science, information systems, information technology*, and *software engineering*. These five satisfy our criterion for inclusion: each one has, or will soon have, a recent volume of undergraduate curriculum guidelines that is approved and

published by one or more international professional and scientific societies. These five also attract the overwhelming majority of all U.S. undergraduates who are majoring in computing.

We expect that in the future additional computing disciplines may satisfy our criterion. When that is the case, they may be included in future editions of this report. Candidates for future editions might include new disciplines that don't yet have such guidelines (e.g., *bioinformatics*) and more established disciplines that have not recently issued such guidelines (e.g., *computer engineering technology*).

The foundation of this report is the set of curriculum standards that exist for undergraduate degree programs in the five major computing-related disciplines mentioned above. Each one of the five discipline-specific curricula volumes represents the best judgment of the relevant professional, scientific, and educational associations, and serves as a definition of what these degree programs should be and do.

While some of those reports may be scheduled for revision, we have made no effort to update their contents, as that is beyond our mission and authority. Rather, we have taken what is given in the five current curricula volumes, compared their contents to one another, and synthesized what we believe to be essential descriptive and comparative information. The five curricula volumes contain a great deal of detailed information not included here. Readers who want detailed information about any of the five disciplines covered in this report should consult the original sources. The computing curricula volumes can be found at http://www.acm.org/education/curricula.html and http://computer.org/curriculum.

In addition to using those five reports as the basis for this report, we have referred to the computing professions and other supporting information as necessary. We have not focused on other kinds of undergraduate computing degree programs, on graduate education in computing, or on the identities of the computing research communities. Nor have we included any information or comment about non-traditional computing education, such as provided in conjunction with vendor-specific certification programs; those arenas are deserving of evaluation, but such work is beyond the scope of this project.

The remainder of this report includes the following:

- In Chapter 2, we characterize each of the five major computing disciplines.
- In Chapter 3, we flesh out the characteristics of each of these five kinds of degree program and compare them to each other. We also compare and contrast the kind of professional capabilities expected of the graduates of each kind of degree program.
- In Chapter 4, we conclude by alerting educators, administrators, and other responsible parties to some issues that may emerge in the creation of new computing disciplines.
- In Chapter 5, we tell you how to obtain online copies of the five discipline-specific curriculum reports and offer guidance about how to use them.

Following publication of this report, we will prepare and publish a *Guide to Undergraduate Programs in Computing*. This will be a shorter standalone document to be distributed more widely than the Overview Report. In it, we will provide information for prospective students, and for those who advise them, to help them make well-informed choices.

Computing itself will continue to evolve. In addition, new computing-related disciplines are likely to emerge. As we update the existing discipline-specific reports and as additional reports for new computing disciplines emerge, you can expect to see updated versions of this report. To find out if this document (*CC2005-Overview)* is the most recent edition of the *Overview Report on Computing Curricula*, go to http://www/education/curricula.html or http://computer.org/curriculum. From either of those sites, you will be able to determine if a newer version exists. If a newer version exists, you may download the newest version from either site.

1.3. Background and history

Over the last forty years, four major organizations in the U.S. have developed computing curriculum guidelines for colleges and universities:

- The Association for Computing Machinery (generally called "the ACM" or "the Association for Computing") is a scientific and professional organization founded in 1947. It is concerned with the development and sharing of new knowledge about all aspects of computing (the word "machinery" in its name is a historical artifact). It has traditionally been the professional home of computer scientists, who devise new ways of using computers and who advance the science and theory that underlies both computation itself and the software that enables it. The ACM began publishing curriculum recommendations for CS in 1968 (a preliminary version appeared in 1965) and for IS in 1972.
- The Association for Information Systems (generally called "AIS") was founded in 1994. It is a global organization serving those academics who specialize in Information Systems. Most academic members of AIS are affiliated with Schools/Colleges of Business or Management. AIS began providing curriculum recommendations for IS in cooperation with ACM and AITP in 1997.
- The Association for Information Technology Professionals (often referred to as "the AITP") was founded in 1951 as the National Machine Accountants Association. In 1962, it became the Data Processing Management Association (or DPMA). It adopted its present name in 1996. AITP focuses on the professional side of computing, serving those who use computing technology to meet the needs of business and other organizations. It first provided curriculum recommendations for IS in 1985.
- The Computer Society of the Institute for Electrical and Electronic Engineers (often referred to as "the IEEE-CS" or "the Computer Society") originated in 1946 as the committee on Large Scale Computing Devices of the American Institute of Electrical Engineers (AIEE) and in 1951 as the Professional Group on Electronic Computers of the Institute of Radio Engineers (IRE). The AIEE and the IRE merged in 1964 to become the IEEE and the two subunits joined to become the Computer Society. The Computer Society is a technical society within the IEEE that is focused on computing from the engineering perspective. Today the Computer Society's members include computer engineers, software engineers, computer technologists and computer scientists. It began providing curriculum recommendations in 1977. In recent years, there has been a large overlap in membership between the ACM and the Computer Society.

Prior to the 1990's, each society produced its own curriculum recommendations. Over time, the advantages of cooperative work among them became obvious. Today, the societies cooperate in creating curriculum standards, and in this way send a single message to the computing community. Many researchers and teachers belong to more than one of the societies.

The ACM and the IEEE-CS joined forces in the late 1980s to create a joint curriculum report for computing. Published in 1991 and known as *Computing Curricula 1991* or CC'91 (CC91), it provided guidelines for curricula for four-year Bachelor's degree programs in computer science and computer engineering. Throughout the 1990s various efforts were made to produce curricula guidelines for other programs in computing education. By 1993, the ACM had produced five reports for two-year Associate degree programs, one report for each of *computer science*, *computer engineering technology*, *information systems*, *computer support services*, and *computing for other disciplines*. [AssocDeg] Also in 1993, the ACM produced curriculum recommendations for a high school curriculum [HS]. In 1997, the ACM, AIS, and AITP [AIS] published a model curriculum and a set of guidelines for four-year Bachelors degree programs in *information systems* [IS97]. The 1990s also saw newer computing disciplines such as *software engineering* gain increased prominence in the U.S.

By the end of the 1990s, it was becoming clear that the field of computing had not only grown rapidly but had also grown in many dimensions. The proliferation of different kinds of degree programs in

computing left many people confused. Given the growing number of kinds of computing degree programs, confusion was perhaps inevitable. This diversity of computing degrees was a problem that had not existed in a significant way prior to the explosion of computing's impact in the 1990s. Because it was a new problem, there was no established way of coordinating and simplifying the choices that suddenly seemed to be appearing everywhere.

When the ACM and the IEEE-CS again joined forces in the late 1990s to produce an up-to-date curriculum report to replace CC'91, these organizations could no longer ignore the problem. The original plan called for the two societies to form a joint task force that would update the CC'91 report. ACM and IEEE-CS created a joint task force and its goal was to produce Computing Curricula 2001 (CC2001), a single report that would provide curriculum guidelines for degree programs for the various computing disciplines. However, the members of the task force soon recognized the new reality: computing had grown in so many dimensions that no single view of the discipline seemed adequate. The days when the field of computing consisted of only *computer science, computer engineering* and *information systems* were over, and the richness and breadth provided by the various computing disciplines called for a new way of defining what computing curricula should be.

The CC2001 Task Force faced this challenge by making four important decisions:

1. There should be a curriculum report (or volume) for each of the major computing disciplines, including *computer engineering, computer science, information systems,* and *software engineering;*

2. The number of computing-related disciplines is likely to grow. The curriculum report structure must accommodate not only the four computing disciplines that were established at that time (enumerated above) but also must accommodate new computing disciplines as they emerge.

3. The growing number of computing disciplines naturally causes confusion in many quarters. Therefore, in addition to the various discipline-specific volumes, there must also be an *Overview* report to serve as a practical "umbrella" guide to the discipline-specific volumes.

4. The pace of change in computing is sufficiently rapid that we must establish a process by which the organizations could update curriculum guidelines more frequently than once per decade.

The Task Force recognized that its members were primarily computer scientists and deemed itself qualified to produce a report only for computer science. It called for the ACM, the IEEE-CS, the AIS, and other professional societies to cooperate in efforts to create volumes for computer engineering, information systems, and software engineering. The work of this task force, known as *Computing Curricula 2001* (CC2001), was published in December 2001 [CC2001]. The CC2001 Report contains two important components:

- A new structure for computing curriculum guidelines encompassing the decisions taken by the Task Force, described above and henceforth referred to as "the CC2001 model".
- Detailed curricula guidelines for undergraduate degree programs in computer science.

Because the *CC2001* report included CS curriculum guidelines, those who refer to it for its computer science content might think of as *CS2001*. Beginning with the publication of the *CC2005* report, the title "Computing Curricula 20xx" will be used for Overview reports. New editions of the CS curriculum guidelines will be called "Computer Science 20xx". In all cases, "20xx" will be the year of publication.

In response to the *CC2001* model, work soon began on other discipline-specific volumes:

- The *information systems* community published its updated *IS2002* report in 2002 [IS2002].
- The *software engineering* community published its first report, *SE2004*, in 2004 [SE2004].

- The *computer engineering* community published its *CE2004* report in 2004 [CE2004].
- The CC2001 prediction of additional emerging computing disciplines has already proved correct. A report on degree programs in *information technology* is under development. We anticipate that it will be published in 2006 and thus refer to it as *IT2006*.

The diagram in Figure 1.1 represents the scope of what has become the *Computing Curricula Series*, a continuing effort to provide guidelines and standards for computing curricula. The top-level *Overview* block, *CC2005*, represents this report. Each of the first five sub-blocks represents a curriculum report for one of the existing computing disciplines. The sixth sub-block is a placeholder for future reports on additional computing disciplines as necessitated by the emergence of new computing disciplines. Online copies of the computing curricula volumes can be found at http://www.acm.org/education/curricula.html and http://computer.org/curriculum.

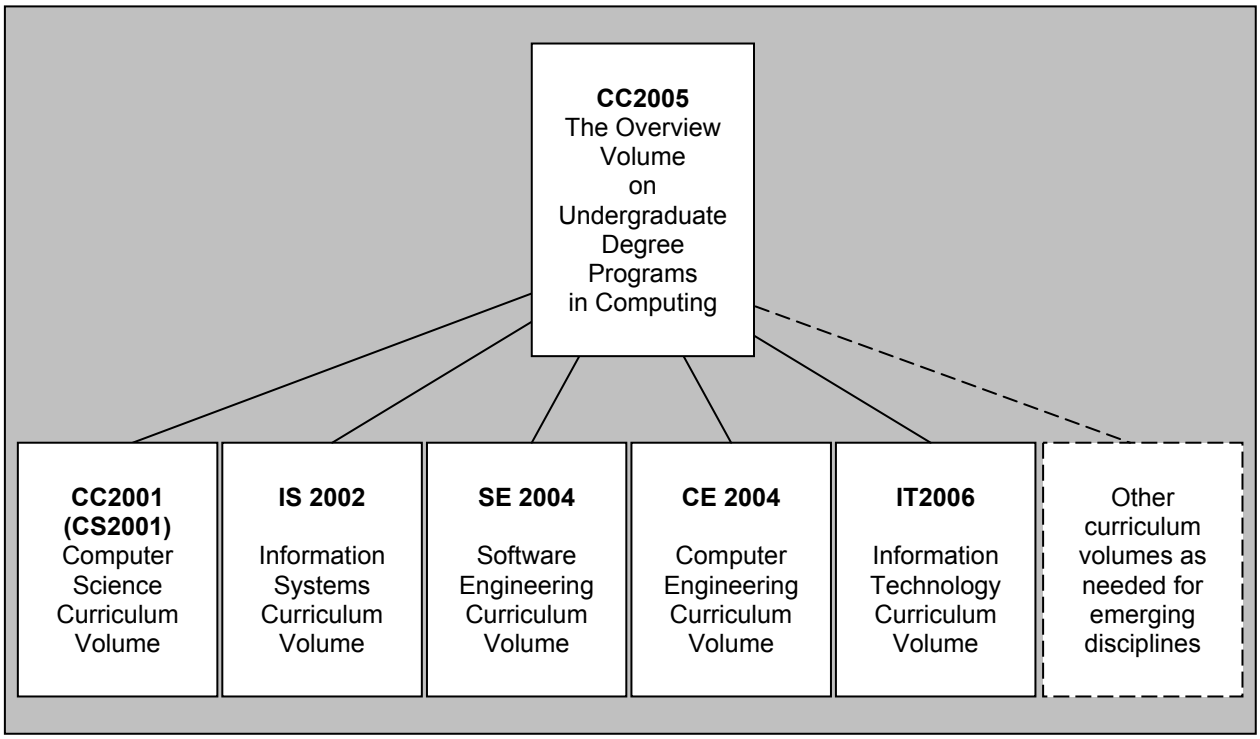

Figure 1.1. Structure of the *Computing Curricula Series*

1.4. Guiding Principles

Five principles guided the development of this report.

1. ***The dramatic growth in the number of computing disciplines, and in their collective impact on society, requires that the computing disciplines articulate a shared identity.*** Given the importance of computing to society, we in computing have a responsibility to help society understand what we do. The fact that computing offers several kinds of academic programs is a major strength and an opportunity but requires that we offer society a practical vision of our shared field, of the various disciplines within it, and of the meaningful choices that face students, educators, and their communities. The goal of this report is to articulate the shared identity, the separate identities of each computing discipline, and the choices available to students, educators, and communities.

2. ***Each computing discipline must be a participant in defining the identities and choices as articulated in this report.*** Each computing discipline must articulate its own identity, recognize the identities of the other disciplines, and contribute to the shared identity of computing.

3. ***This report must address a broad audience, not just its technically oriented constituents***. As discussed in Section 1.1, the audience for this report includes a range of people who have reason to become familiar with academic computing degree programs. Most members of that audience are not computing educators. Our goal is to paint a concise and useful picture that will illuminate the choices faced by students and by those who are responsible for shaping their educational choices. This goal is fundamentally different from the goal of reports that define curriculum guidelines for degree programs. It dictates that we must be relatively concise and that we minimize technical jargon. We ask the technically oriented reader to appreciate our need to avoid the kind of distinctions and technical emphasis expected of documents aimed at a technical audience.

4. ***We should characterize the computing disciplines by reference to the body of knowledge and skills defined in the most recent curriculum report for each of those disciplines.*** The definition of a shared characterization of the computing disciplines is unprecedented, and it is imperative that we set attainable goals. We confine our attention to the bodies of knowledge and skills defined by each computing discipline as published in the individual curriculum reports; we do not consider pedagogy or course definition. We believe that pedagogical issues and the definition of computing courses that might serve multiple audiences across the computing disciplines are important and timely concerns. However, we believe we would be ill advised to address such issues in this report. This decision should not be construed as a precedent for others to follow, and we expect that authors of subsequent reports will revisit this issue.

5. ***This report must go beyond an examination of details to generate a useful synthesis for the intended audience.*** While the findings of this report are based on examination of the bodies of knowledge in current discipline-specific curriculum volumes, we must go beyond simple examination-and-reporting to generate a synthesis that will be meaningful and useful for our audience. Our task requires representatives of each discipline to make judgments about how to form an insightful, consensus-based overview of the computing disciplines.

Chapter 2. The Computing Disciplines

There are many kinds of computing degree programs. There are dozens and perhaps hundreds around the world. The variety of names used for the programs is even broader. The programs represent a number of computing disciplines. In this report, we focus on five that are prominent today: *computer engineering, computer science, information systems, information technology,* and *software engineering*. These five satisfy our criterion for inclusion: each one has, or will soon have, a volume of undergraduate curriculum guidelines that is approved and published by one or more international professional and scientific societies. We expect that, in the future, additional computing disciplines may satisfy this criterion. When that is the case, they may be included in future editions of this report.

2.1. What is computing?

In a general way, we can define computing to mean any goal-oriented activity requiring, benefiting from, or creating computers. Thus, computing includes: designing and building hardware and software systems for any of a wide range of purposes; processing, structuring and managing various kinds of information; doing scientific studies using computers; making computer systems behave intelligently; creating and using communications and entertainment media; finding and gathering information relevant to any particular purpose, and so on. The list is virtually endless, and the possibilities are vast. Computing also has other meanings that are more specific, based on the context in which the term is used. For example, an information systems specialist will view computing somewhat differently from a software engineer. Regardless of the context, doing computing well can be complicated and difficult. Because society needs people to do computing well, we must think of computing not only as a profession but also as a discipline.

A student typically earns a bachelors degree in one of the computing disciplines to prepare for entry into the computing profession. Because computing provides such a wide range of choices, it is impossible for anyone to become proficient at all of them. Hence, an individual who wishes to become a computing professional requires some focus for his or her professional life. There are currently five major kinds of undergraduate degree programs in computing, and each one provides a different focus and perspective on the discipline of computing. In the next section, we shall see what these five computing disciplines are and how they compare in terms of their focus and the kinds of problems and issues they address.

2.2. The landscape of computing disciplines

Computing is not just a single discipline but is a family of disciplines. During the 1990s, important changes in computing and communications technology, and in the impact of that technology on society, led to important changes in this family of disciplines.

2.2.1. Before the 1990s

Undergraduate degree programs in the computing-related disciplines began to emerge in the 1960s. Originally, there were only three kinds of computing-related degree programs in North America: *computer science, electrical engineering*, and *information systems*. Each of these disciplines was concerned with its own well-defined area of computing. Because they were the only prominent computing disciplines, and because each one had its own area of work and influence, it was much easier for students to determine which kind of degree program to choose. For students who wanted to become expert in developing software or with the theoretical aspects of computing, *computer science* was the obvious choice. For students who wanted to work with hardware, *electrical engineering* was the clear option. For students who wanted to use hardware and software to solve business problems, *information systems* was the right choice.

Each of these three disciplines had its own domain. There was not any shared sense that they constituted a family of computing disciplines. As a practical matter, computer scientists and electrical engineers sometimes worked closely together, as they were both concerned with developing new technology, were often housed in the same part of the university, and sometimes required each others' help. Information systems specialists had ties with business schools and did not have much interaction with computer scientists and electrical engineers.

Before the 1990s, the only major change in this landscape in the U.S. was the development of *computer engineering*. Prior to the invention of chip-based microprocessors, *computer engineering* was one of several areas of specialization within *electrical engineering*. With the advent of the microprocessor in the mid-1970s, *computer engineering* began to emerge from within *electrical engineering* to become a discipline unto itself. For many people outside of the engineering community, however, the distinction between *electrical engineering* and *computer engineering* was not clear. Before the 1990s, therefore, when prospective students surveyed the choices of computing-related degree programs, most would have perceived the computing disciplines as shown in the top half of Figure 2.1. The distance between the disciplines indicates how closely the people in those disciplines worked with each other.

2.2.2. Significant developments of the 1990s

During the 1990s, several developments changed the landscape of the computing disciplines in North America, although in other parts of the world some of these changes occurred earlier:

- *Computer engineering* solidified its emergence from *electrical engineering*. Computer engineering emerged from electrical engineering during the late 1970s and the 1980s, but it was not until the 1990s that computer chips became basic components of most kinds of electrical devices and many kinds of mechanical devices. (For example, modern automobiles contain numerous computers that perform tasks that are transparent to the driver.) Computer engineers design and program the chips that permit digital control of many kinds of devices. The dramatic expansion in the kinds of devices that rely on chip-based digital logic helped computer engineering solidify its status as a strong field and, during the 1990s, unprecedented numbers of students applied to computer engineering programs. Outside of North America, these programs often had titles such as *computer systems engineering*.
- *Computer science* grew rapidly and became accepted into the family of academic disciplines. At most American colleges and universities, computer science first appeared as a discipline in the 1970s. Initially, there was considerable controversy about whether computer science was a legitimate academic discipline. Proponents asserted that it was a legitimate discipline with its own identity, while critics dismissed it as a vocational specialty for technicians, a research platform for mathematicians or a pseudo-discipline for computer programmers. By the 1990s, computer science had developed a considerable body of research, knowledge, and innovation that spanned the range from theory to practice, and the controversy about its legitimacy died. Also during the 1990s, computer science departments faced unprecedented demands. Industry needs for qualified computer science graduates exceeded supply by a large factor. Enrollments in CS programs grew very dramatically. While CS had already experienced cycles of increasing and decreasing enrollments throughout its brief history, the enrollment boom of the 90s was of such magnitude that it seriously stressed the ability of CS departments to handle very large numbers of students. With increased demands for both teaching and research, the number of CS faculty at many colleges and universities grew significantly.
- *Software engineering had* emerged as an area within *computer science*. As computing is used to attack a wider range of complex problems, creating reliable software becomes more difficult. With large, complex programs, no one person can understand the entire program, and various parts of the program can interact in unpredictable ways. (For example, fixing a bug in one part of a program can create new bugs elsewhere.) Computing is also used in safety-critical tasks, where a single bug can cause injury or death. Over time, it became clear that producing good software is very difficult, very expensive, and

very necessary. This lead to the creation of software engineering, a term which emanated from a NATO sponsored conference held in Garmisch, Germany in 1968. While computer science (like other sciences) focuses on creating new knowledge, software engineering (like other engineering disciplines) focuses on rigorous methods for designing and building things that reliably do what they're supposed to do. Major conferences on software engineering were held in the 1970s, and during the 1980s some computer science degree programs included software engineering courses. However, in the U.S. it was not until the 1990s that one could reasonably expect to find software engineering as a significant component of computer science study at many institutions.

- *Software engineering* began to develop as a discipline unto itself. Originally the term 'software engineering' was introduced to reflect the application of traditional ideas from engineering to the problems of building software. As software engineering matured, the scope of its challenge became clearer. In addition to its computer science foundations, software engineering also involves human processes that, by their nature, are harder to formalize than are the logical abstractions of computer science. Experience with software engineering courses within computer science curricula showed many that such courses can teach students "about the field of software engineering" but usually do not succeed at teaching them "how to be software engineers". Many experts concluded that the latter goal requires a range of coursework and applied project experience that goes beyond what can be added to a computer science curriculum. Degree programs in software engineering emerged in the United Kingdom and Australia during the 1980s, but these programs were in the vanguard. In the United States, degree programs in software engineering, designed to provide a more thorough foundation than can be provided within computer science curricula, began to emerge during the 1990s.
- *Information systems* had to address a growing sphere of challenges. Prior to the 1990s, many information systems specialists focused primarily on the computing needs that the business world had faced since the 1960s: accounting systems, payroll systems, inventory systems, etc. By the end of the 1990's, networked personal computers had become basic commodities. Computers were no longer tools only for technical specialists; they became integral parts of the work environment used by people at all levels of the organization. Because of the expanded role of computers, organizations had more information available than ever before and organizational processes were increasingly enabled by computing technology. The problems of managing information became extremely complex, and the challenges of making proper use of information and technology to support organizational efficiency and effectiveness became crucial issues. Because of these factors, the challenges faced by information systems specialists grew in size, complexity, and importance. In addition, information systems as a field paid increasing attention to the use of computing technology as a means for communication and collaborative decision making in organizations.
- *Information technology* programs began to emerge in the late 1990s. During the 1990s, computers became essential work tools at every level of most organizations and networked computer systems became the information backbone of organizations. While this improved productivity, it also created new workplace dependencies, as problems in the computing infrastructure can limit employees' ability to do their work. IT departments within corporations and other organizations took on the new job of ensuring that the organization's computing infrastructure was suitable, that it worked reliably, and that people in the organization had their computing-related needs met, problems fixed, etc. By the end of the 1990s, it became clear that academic degree programs were not producing graduates who had the right mix of knowledge and skills to meet these essential needs. Colleges and universities developed degree programs in information technology to fill this crucial void.

Collectively these developments reshaped the landscape of the computing disciplines. Tremendous resources were allocated to information technology activities in all industrialized societies because of various factors, including the explosive growth of the World Wide Web, anticipated Y2K problems and, in Europe, the launch of the Euro.

2.2.3. After the 1990s

The new landscape of computing degree programs reflects the ways in which computing-as-a-whole has matured to address the problems of the new millennium. In the U.S., *computer engineering* had solidified its status as a discipline distinct from *electrical engineering* and assumed a primary role with respect to computer hardware and related software. *Software engineering* has emerged to address the important challenges inherent in building software systems that are reliable and affordable. *Information technology* has come out of nowhere to fill a void that the other computing disciplines did not adequately address.

This maturation and evolution has created a greater range of possibilities for students and educational institutions. The increased diversity of computing programs means that students face choices that are more ambiguous than they were before the 1990s. The bottom portion of Figure 2.1 shows how prospective students might perceive the current range of choices available to them. The dotted ovals show how prospective students are likely to perceive the primary focus of each discipline.

It is clear where students who want to study hardware should go. *Computer engineering* has emerged from *electrical engineering* as the home for those working on the hardware and software issues involved in the design of digital devices. For those with other interests, the choices are not so clear-cut. In the pre-1990s world, students who wanted to become expert at software development would study computer science. The post-1990s world presents meaningful choices: *computer science, software engineering*, and even *computer engineering* each include their own perspective on software development. These three choices imply real differences: for CE, software attention is focused on hardware devices; for SE, the emphasis is creating software that satisfies robust real-world requirements, and for CS software is the

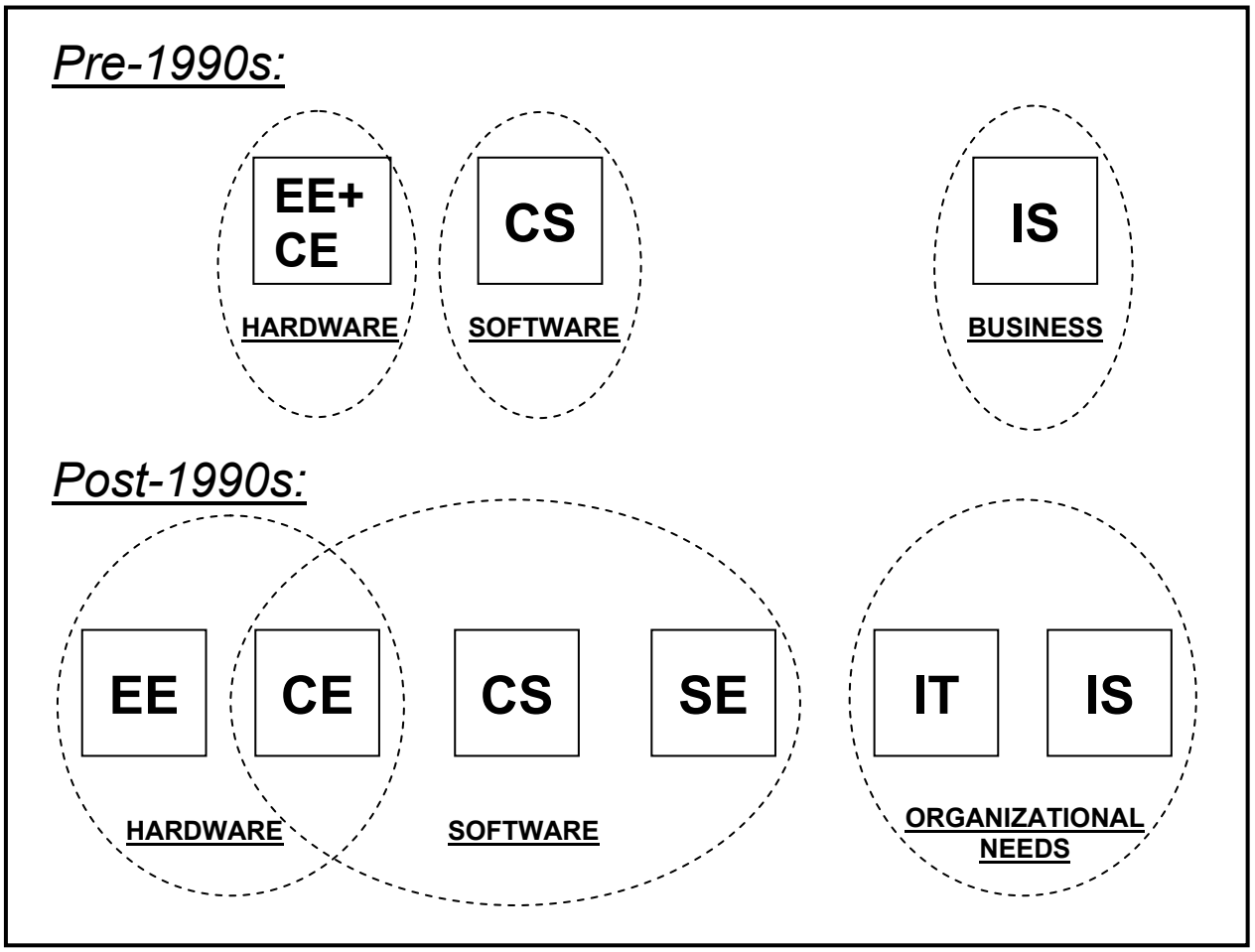

Figure 2.1. Harder choices: How the disciplines might appear to prospective students.

currency in which ideas are expressed and a wide range of computing problems and applications are explored. Such distinctions may not be visible to prospective students. Naïve students might perceive that all three disciplines share an emphasis on software and are otherwise indistinguishable.

Similarly, in the pre-1990s world, a primary area for applying computing to solve real-world problems was in business, and i*nformation systems* was the home for such work. The scope of real-world uses has broadened from business to organizations of every kind, and students can choose among *information systems* and *information technology* programs. While the IT and IS disciplines both include a focus on software and hardware, neither discipline emphasizes them "for its own sake"; rather, they use technology as critical instruments for addressing organizational needs. While IS focuses on the generation and use of information, and IT focuses on ensuring that the organization's infrastructure is appropriate and reliable, prospective students might be unaware of these important differences and see only that IS and IT share a purpose: using computing to meet the needs of technology-dependent organizations.

2.3. Descriptions of the major computing disciplines

In this section, we characterize each of the five major computing disciplines. See sections 3.4 and 3.5 for more information on how to understand this important distinction between the *names of the computing disciplines* and the *names of a particular degree program*.

2.3.1. Computer Engineering

Computer engineering is concerned with the design and construction of computers and computer-based systems. It involves the study of hardware, software, communications, and the interaction among them. Its curriculum focuses on the theories, principles, and practices of traditional electrical engineering and mathematics, and applies them to the problems of designing computers and computer-based devices.

Computer engineering students study the design of digital hardware systems including communications systems, computers and devices that contain computers. They study software development, focusing on software for digital devices and their interfaces with users and other devices. CE study may emphasize hardware more than software or there may be a balanced emphasis. CE has a strong engineering flavor.

Currently, a dominant area within computing engineering is embedded systems, the development of devices that have software and hardware embedded in them. For example, devices such as cell phones, digital audio players, digital video recorders, alarm systems, x-ray machines, and laser surgical tools all require integration of hardware and embedded software, and are all the result of computer engineering.

2.3.2. Computer Science

Computer science spans a wide range, from its theoretical and algorithmic foundations to cutting-edge developments in robotics, computer vision, intelligent systems, bioinformatics, and other exciting areas. We can think of the work of computer scientists as falling into three categories:

- They design and implement software. Computer scientists take on challenging programming jobs. They also supervise other programmers, keeping them aware of new approaches.
- They devise new ways to use computers. Progress in the CS areas of networking, database, and human-computer-interface enabled the development of the World Wide Web. Now CS researchers are working with scientists from other fields to make robots become practical, intelligent aides, to use databases to create new knowledge, and to use computers help decipher the secrets of our DNA.
- They develop effective ways to solve computing problems. For example, computer scientists develop the best possible ways to store information in databases, send data over networks, and display complex images. Their theoretical background allows them to determine the best performance possible, and their study of algorithms helps them develop new approaches that provide better performance.

Computer science spans the range from theory through programming. Curricula that reflect this breadth are sometimes criticized for failing to prepare graduates for specific jobs. While other disciplines may produce graduates with more immediately relevant job-related skills, computer science offers a comprehensive foundation that permits graduates to adapt to new technologies and new ideas.

2.3.3. Information Systems

Information systems specialists focus on integrating information technology solutions and business processes to meet the information needs of businesses and other enterprises, enabling them to achieve their objectives in an effective, efficient way. This discipline's perspective on "information technology" emphasizes *information*, and sees *technology* as an instrument to enable the generation, processing and distribution of needed information. Professionals in this discipline are primarily concerned with the information that computer systems can provide to aid an enterprise in defining and achieving its goals, and the processes that an enterprise can implement and improve using information technology. They must understand both technical and organizational factors, and must be able to help an organization determine how information and technology-enabled business processes can provide a competitive advantage.

The information systems specialist plays a key role in determining the requirements for an organization's information systems and is active in their specification, design, and implementation. As a result, such professionals require a sound understanding of organizational principles and practices so that they can serve as an effective bridge between the technical and management communities within an organization, enabling them to work in harmony to ensure that the organization has the information and the systems it needs to support its operations. Information systems professionals are also involved in designing technology-based organizational communication and collaboration systems.

A majority of *Information Systems* (IS) programs are located in business schools. All IS degrees combine business and computing coursework. A wide variety of IS programs exists under various labels which often reflect the nature of the program. For example, programs in Computer Information Systems usually have the strongest technology focus, and programs in Management Information Systems can emphasize organizational and behavioral aspects of IS. Degree programs names are not always consistent.

2.3.4. Information Technology

Information technology is a label that has two meanings. In the broadest sense, the term "information technology" is often used to refer to all of computing. In academia, it refers to undergraduate degree programs that prepare students to meet the computer technology needs of business, government, healthcare, schools, and other kinds of organizations. In some nations, other names are used for such degree programs. For example, in the U.K. such programs are often called "CIT" (for "computing and information technology") rather than "IT".

In the previous section, we said that *Information Systems* focuses on the "information" aspects of "information technology". *Information Technology* is the complement of that perspective: its emphasis is on the technology itself more than on the information it conveys. IT is a new and rapidly growing discipline which started as a grass roots response to the practical, everyday needs of business and other organizations. Today, organizations of every kind are dependent on information technology. They need to have appropriate systems in place. Those systems must work properly, be secure, and be upgraded, maintained, and replaced as appropriate. People throughout an organization require support from IT staff who understand computer systems and their software, and are committed to solving whatever computer-related problems they might have. Graduates of information technology programs address these needs.

Degree programs in information technology arose because degree programs in the other computing disciplines were not producing an adequate supply of graduates capable of handling these very real needs. IT programs exist to produce graduates who possess the right combination of knowledge and practical, hands-on expertise to take care of both an organization's information technology infrastructure and the

people who use it. IT specialists assume responsibility for selecting hardware and software products appropriate for an organization, integrating those products with organizational needs and infrastructure, and installing, customizing and maintaining those applications for the organization's computer users. Examples of these responsibilities include: the installation of networks; network administration and security; the design of web pages; the development of multimedia resources; the installation of communication components; the oversight of email products; and the planning and management of the technology life-cycle by which an organization's technology is maintained, upgraded, and replaced.

2.3.5. Software Engineering

Software engineering is the discipline of developing and maintaining software systems that behave reliably and efficiently, are affordable to develop and maintain, and satisfy all the requirements that customers have defined for them. This reflects its origins as outlined in section 2.2.2. More recently it has evolved in response to factors such as the growing impact of large and expensive software systems in a wide range of situations and the increased importance of software in safety-critical applications. Software engineering is different in character from other engineering disciplines, due to both the intangible nature of software and to the discontinuous nature of software operation. It seeks to integrate the principles of mathematics and computer science with the engineering practices developed for tangible, physical artifacts. Prospective students can expect to see software engineering presented in two contexts:

- Degree programs in computer science offer one or more software engineering courses as elements of the CS curriculum. Some offer a multi-course concentration in software engineering within CS.
- A number of institutions offer a software engineering degree program.

Degree programs in computer science and in software engineering have many courses in common. Software engineering students learn more about software reliability and maintenance and focus more on techniques for developing and maintaining software that is correct from its inception. While CS students are likely to have heard of the importance of such techniques, the engineering knowledge and experience provided in SE programs goes beyond what CS programs can provide. Such is the importance of this that one of the recommendations of the SE report is that during their program of study students of SE should participate in the development of software to be used in earnest by others. SE students learn how to assess customer needs and develop usable software that meets those needs. Knowing how to provide genuinely useful and usable software is of paramount importance.

In the workplace, the term "software engineer" is a job label. There is no standard definition for this term when used in a job description. Its meaning varies widely among employers. It can be a title equivalent to "computer programmer" or a title for someone who manages a large, complex, and/or safety-critical software project. The public must be mindful not confuse the discipline of software engineering with the ambiguous use of the term 'software engineer" as used in employment advertisements and job titles.

2.4. Snapshots: Graphical views of the computing disciplines

To illustrate the commonalities and differences among computing disciplines, we have created graphical characterizations of them. They suggest how each discipline occupies the "problem space" of computing as shown in Figure 2.2. They represent current realities, not ambitions for the future. The focus is on what people in each of the disciplines typically do after graduation, not on all topics a student might study. Some individuals will have career roles that go beyond the scenarios described by these snapshots.

The horizontal range runs from "Theory, Principles, Innovation" on the left, to "Application, Deployment, Configuration" on the right. Thus, someone who likes the idea of working in a laboratory to invent new things or in a university to develop new principles will want to work in a discipline that occupies the space to the left. Conversely, someone who wants to help people choose and use appropriate technology

or who wants to integrate off-the-shelf products to solve organizational problems will want an area that occupies space to the right. Because there are many kinds of job tasks that fall between the extremes, one should not just look only at the far left and far right, but also consider possibilities between the extremes.

The vertical range runs from "Computer Hardware and Architecture" at the bottom, to "Organizational Issues and Information Systems" at the top. As we go up this axis, the focus is on people, information, and the organizational workplace. As we move down on this axis, the focus is on devices and on the data shared among them. Thus, someone who likes designing circuits or is curious about a computer's inner workings will care about the lower portions; someone who wants to see how technology can work for people, or who is curious about technology's impact on organizations, will care about the upper portions.

We can consider the horizontal and vertical dimensions together. Someone who cares about making things work for people and is more interested in devices than organizations will be interested in the lower-right, someone who wants to develop new theories about how information affects organizations will be interested in the upper-left, and so on.

In Figures 2.3 through 2.7, we use this framework to sketch out the conceptual territory occupied by each of the five computing disciplines. These are informal illustrations used to communicate the task force's subjective interpretation of the various disciplines. They are not based on any precise quantitative foundation. Furthermore, they show only computing topics. Both *computer engineering* and *information systems* programs devote significant attention to topics which are outside of computing and not reflected in this diagram. Tables of required computing and non-computing topics are provided in Chapter 3.

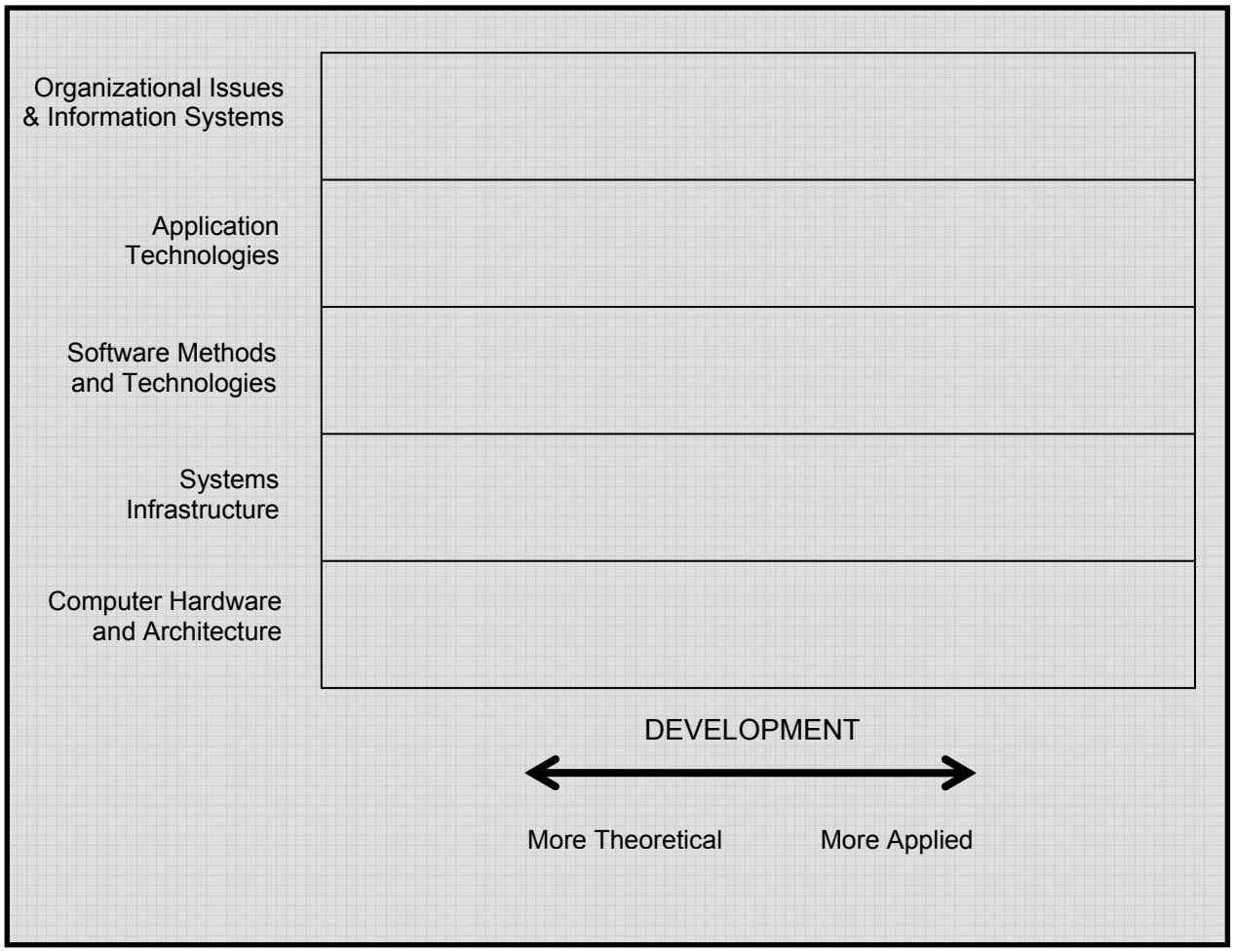

Figure 2.2. The problem space of computing

2.4.1. Computer Engineering

The shaded portion in Figure 2.3 represents the *computer engineering* discipline. It is broad across the bottom because computer engineering covers the range from theory and principles to practical application of designing and implementing products using hardware and software. It narrows towards the center as we move upwards because computer engineers' interests narrow as we move away from the hardware. By the time we get up to the level of software development, we see that computer engineers' interest has narrowed to the horizontal center because they care about software only inasmuch as they need it to develop integrated devices.

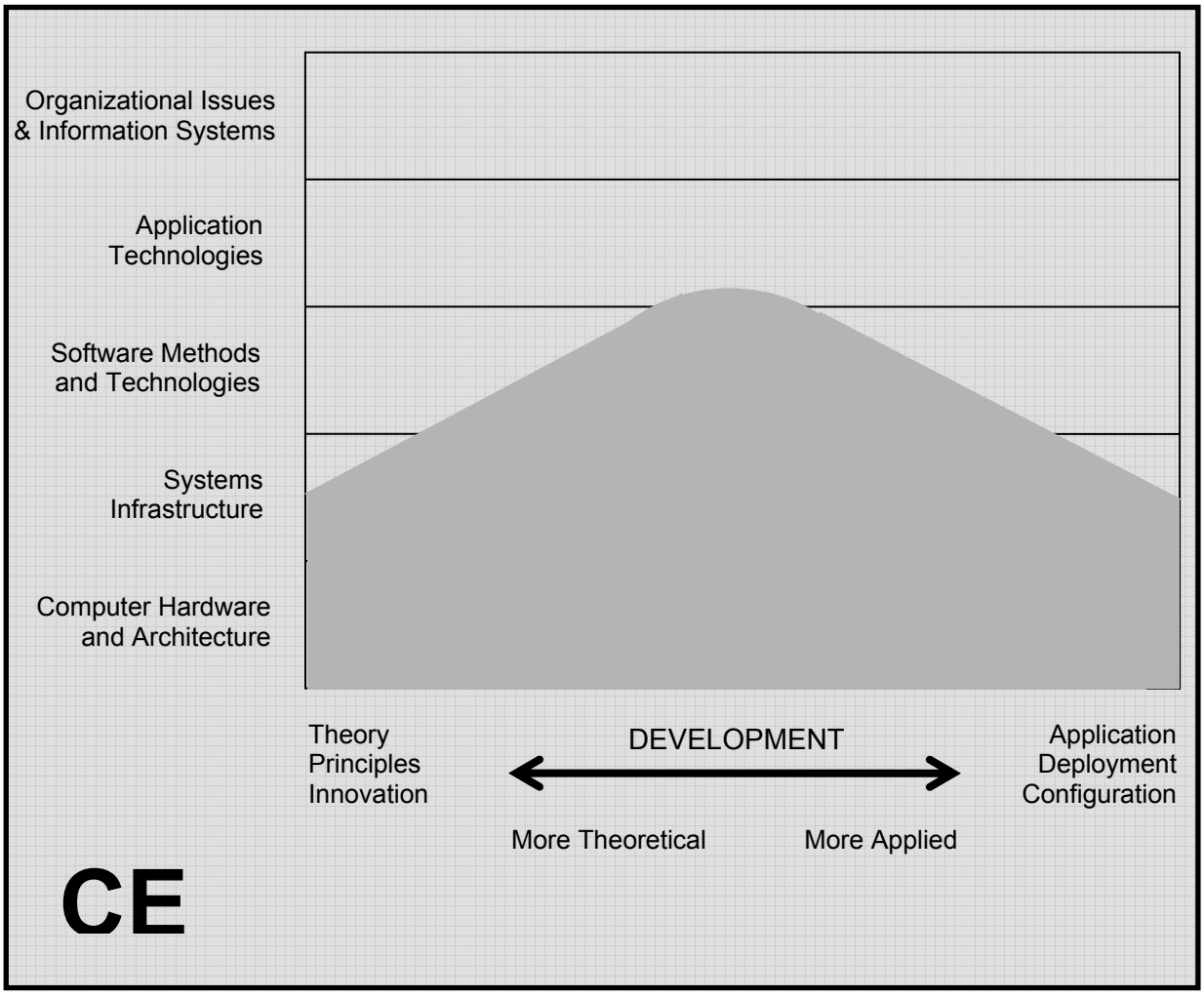

Figure 2.3. Computer Engineering

2.4.2. Computer Science

The shaded portion in Figure 2.4 represents *computer science*. Computer science covers most of the vertical space between the extreme top and extreme bottom because computer scientists generally do not deal with "just the hardware" that runs software, or about "just the organization" that make use of the information that computing can provide. As a group, computer scientists care about almost everything in between those areas (down as far as the software that enables devices to work; up as far as the information systems that help organizations operate). They design and develop all types of software, from systems infrastructure (operating systems, communications programs, etc.) to application technologies (web browsers, databases, search engines, etc.) Computer scientists create these capabilities, but they do not manage the deployment of them. Therefore, the shaded area for computer science narrows and then stops as we move to the right. This is because computer scientists do not help people to select computing products, nor tailor products to organizational needs, nor learn to use such products.

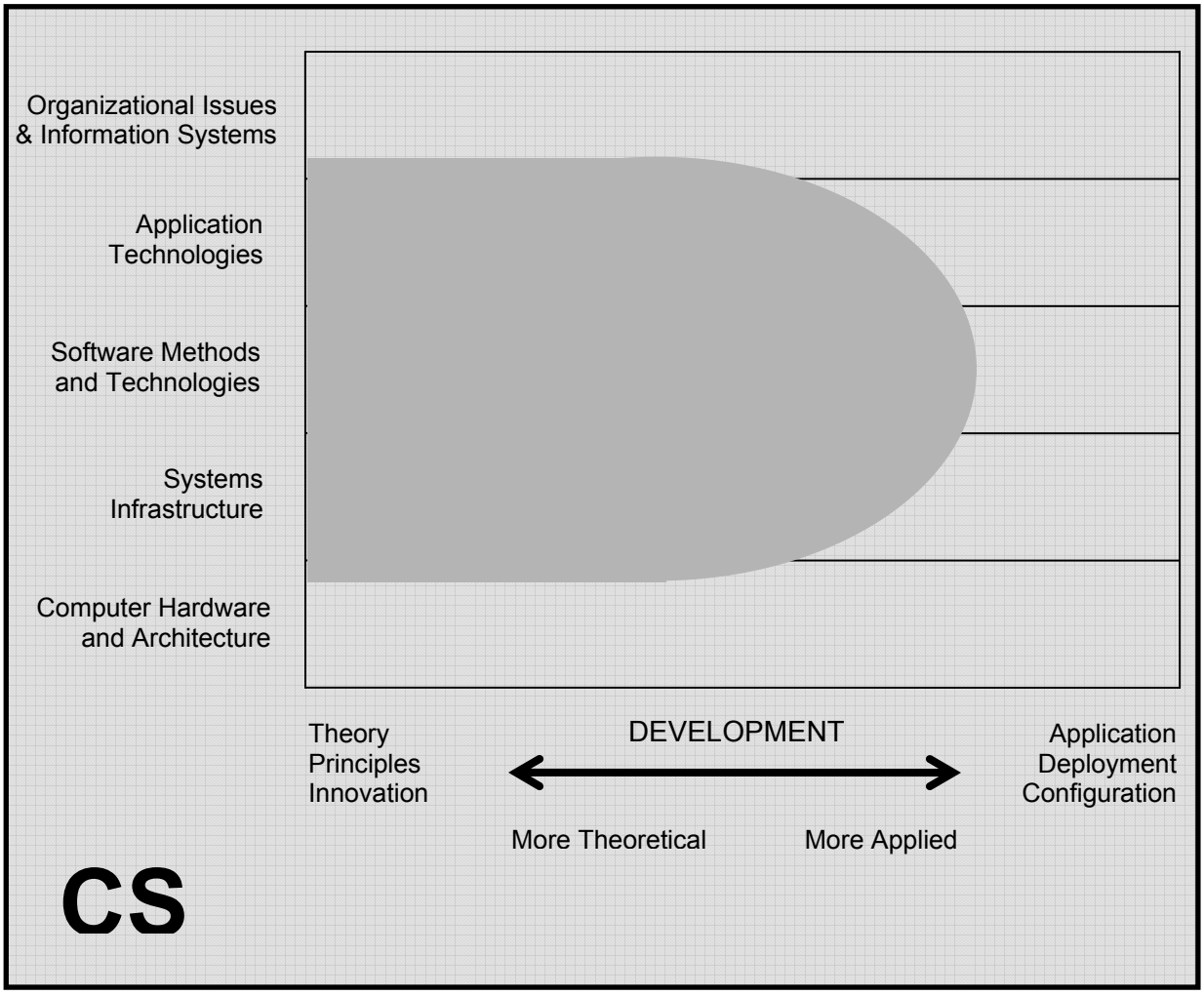

Figure 2.4. Computer Science

2.4.3. Information Systems

The shaded portion in Figure 2.5 represents the discipline of *information systems*. The shaded area extends across most of the top-most level because IS people are concerned with the relationship between information systems and the organizations that they serve, extending from theory and principles to application and development; many IS professionals are also involved in system deployment and configuration and training users. The area covered by IS dips downward, all the way through software development and systems infrastructure in the right half of the graph. This is because IS specialists often tailor application technologies (especially databases) to the needs of the enterprise, and they often develop systems that utilize other software products to suit their organizations' needs for information. (This figure does not reflect the attention that *information systems* programs devote to core business topics. See Chapter 3 for tables which summarize both computing and non-computing topics.)

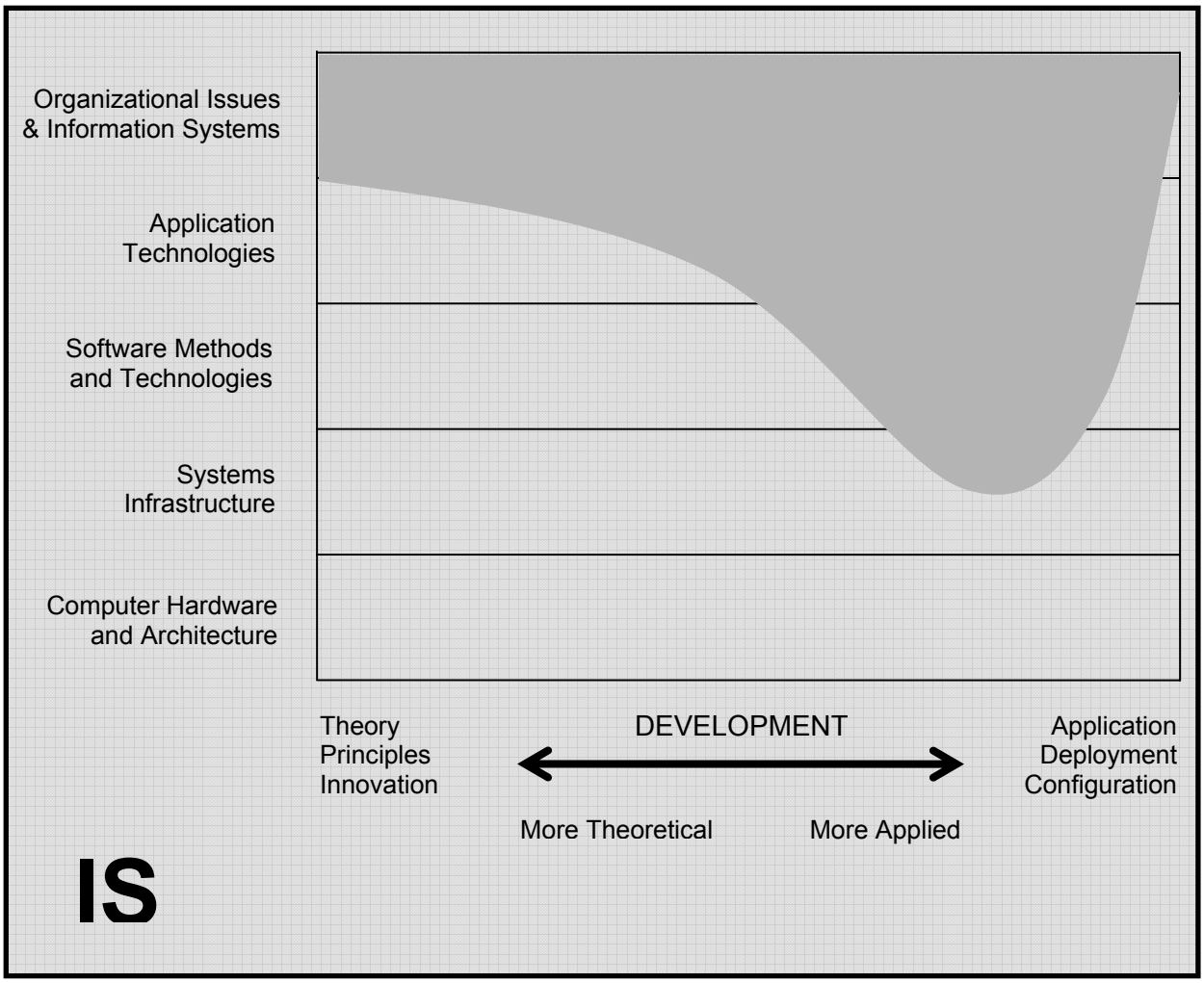

Figure 2.5. Information Systems

2.4.4. Information Technology

The shaded portion in Figure 2.6 represents the *information technology* discipline. Its shaded area extends down most of the right edge, as it focuses on the application, deployment, and configuration needs of organizations and people over a wide spectrum. Across this range (from organizational information systems, to application technologies, and down to systems infrastructure), their role has some overlap with IS, but IT people have a special focus on satisfying human needs that arise from computing technology. In addition, IT's shaded area goes leftwards, from application towards theory and innovation, especially in the area of application technologies. This is because IT people often develop the web-enabled digital technologies that organizations use for a broad mix of informational purposes, and this implies an appropriate conceptual foundation in relevant principles and theory.

Because IT is a very new discipline, its focus has been on developing educational programs that give students a foundation in existing concepts and skills. Many in the community of IT faculty assert that research in their field will grow to create and develop new knowledge in relevant areas. When that happens, an appropriate snapshot would feature a shaded area that extends significantly further to the left. However, this is an ambition and not yet an achievement. This figure reflects IT's current status.

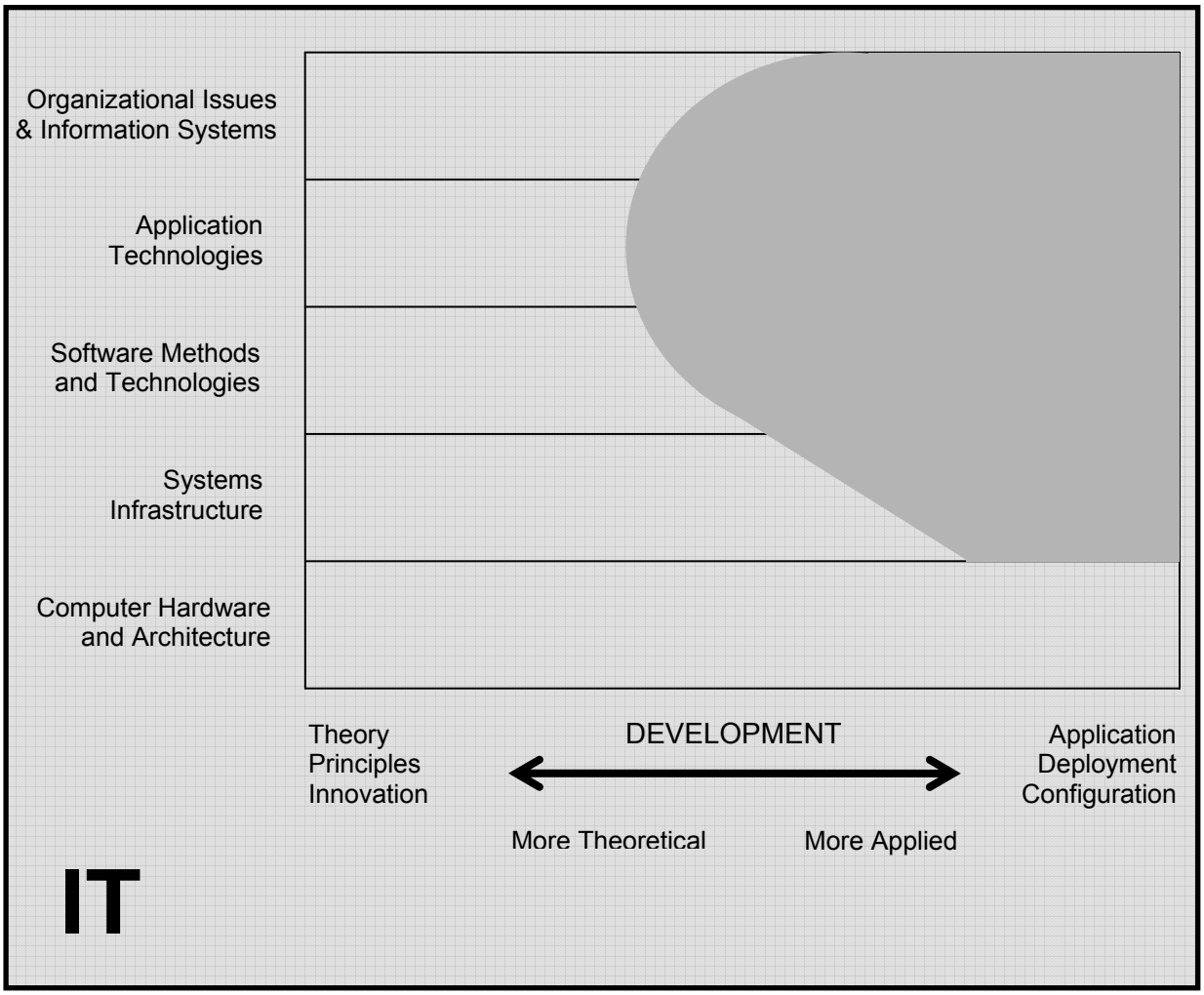

Figure 2.6. Information Technology

2.4.5. Software Engineering

The shaded portion in Figure 2.7 represents the discipline of software engineering. Just as we have seen computer engineering's area span the entire horizontal dimension at the lower hardware-related level, and IS span most of that dimension at the higher organization-related level, software engineering covers a wide range with respect to the systematic development of software. This is because SE people fill a wide range of needs in large-project software expertise. SE's main goal is to develop systematic models and reliable techniques for producing high-quality software on time and within budget, and these concerns extend all the way from theory and principles to daily practice. The domain of SE also extends downward through systems infrastructure, as SE people develop software infrastructure that is robust in operation. Its domain also extends upward into organizational issues because SE people are interested in designing and developing information systems that are appropriate to the client organization.

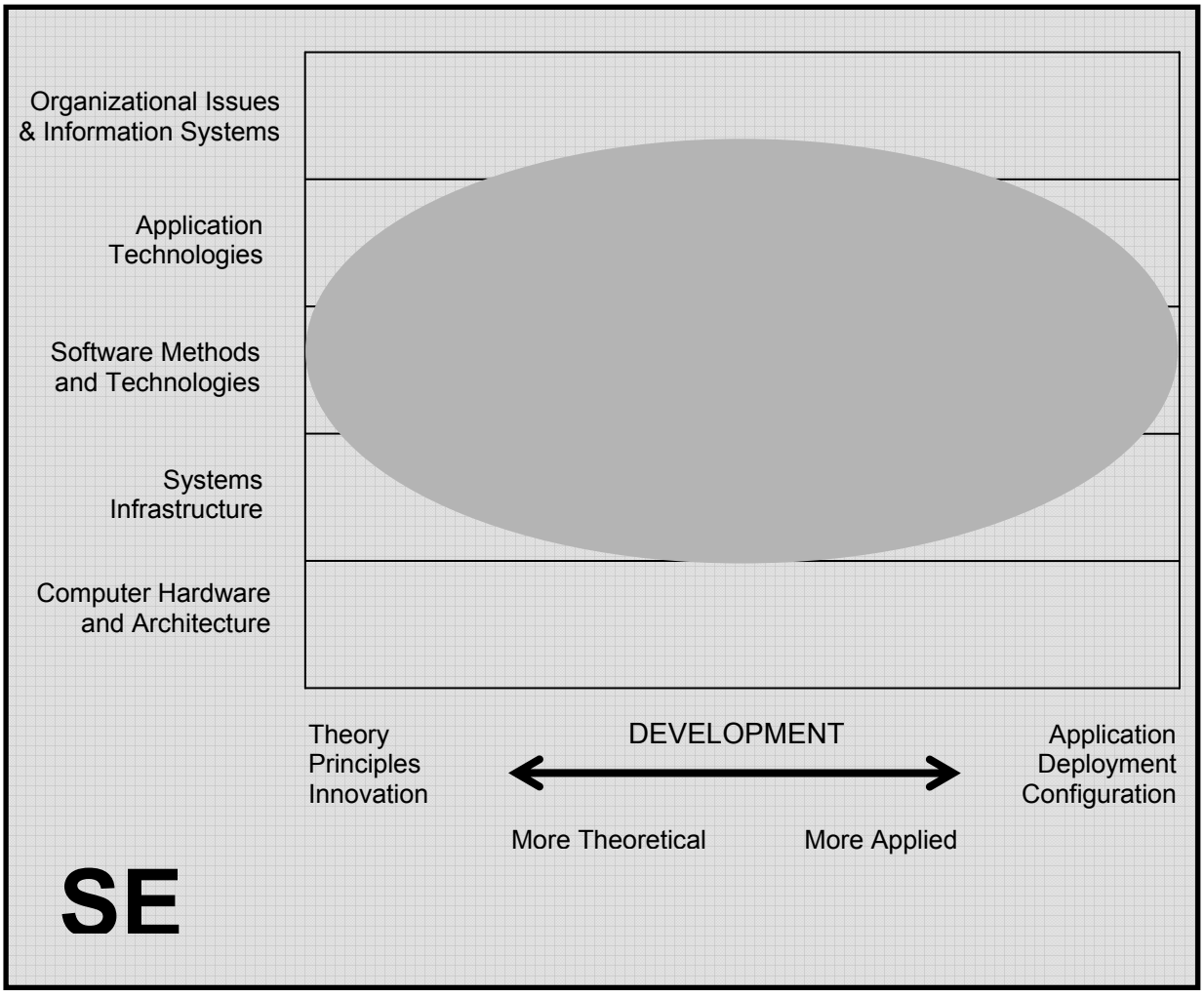

Figure 2.7. Software Engineering

[This page intentionally left blank.]

Chapter 3: Degree programs and career requirements

In this chapter, we summarize the characteristics of degree programs in each of the five major disciplines and compare them to each other in terms of both (a) the relative focus of coverage within degree programs, and (b) the capabilities we expect graduates to have. We then discuss the status and pace of institutional response to the development of the computing disciplines. After summarizing the relationship between degree programs and professional career opportunities, we summarize the elements that are common to all computing degree programs.

3.1. Curriculum summaries: A tabular view of computing degree programs

Graphical views are good for conveying information at a glance but, by their nature, they are incomplete in detail and can give imprecise impressions. In this section we provide a comparison of the computing disciplines for those who want more detail.

Table 3.1 provides a comparative view of the emphasis on *computing* topics among the five kinds of degree programs covered. The left column contains a list of 40 topics that represent areas of computing knowledge and skill that students study in computing degree programs. This list approximates a union of the computing topics specified in the five major computing curriculum reports and, thus, provides a summary of the topics studied at the undergraduate level in one of more of the computing disciplines. If you are unfamiliar with the topics, you may consult the glossary of topics provided in at the end of this document. The various curriculum reports sometimes use different language for a given topic. They also differ in the extent to which they break down a topic into subtopics. As a result, the list of topics provided in Table 3.1 is not an exact match with the topic list of any of the curriculum reports. Rather, it is a summary of topics specified across the five undergraduate computing curriculum reports.

Table 3.2 provides a similar view concerning the relative emphasis on 17 *non-computing* topics across the five kinds of computing degrees. While the curriculum guidelines for each of the five kinds of computing degree mandate coverage of some non-computing topics, two of the computing disciplines lie at the boundary between computing and other disciplines. *Computer engineering* includes elements of both computer science and electrical engineering. The *information systems* discipline spans the boundary between computing and business. Thus, students in these two degree programs devote a significant amount of study to non-computing topics, as shown in Table 3.2.

In both tables, the leftmost column lists topics and the other columns show numerical values per topic for each of the five kinds of computing degree programs. These values range between 0 (lowest) and 5 (highest), and represent the relative emphasis each kind of computing degree program might be reasonably expected to place on each given topic. The values in the tables are only illustrative. They are not intended to represent exact measures of the emphasis each discipline pays to these topics.

For each of the five kinds of degree programs, each topic contains two values: one in the “min” column and one in the “max” column.

- The “min” value represents the minimum emphasis typically placed on that topic as specified in the curriculum report for that computing discipline. The “min” value indicates a discipline’s minimum requirement relative to the minimum requirements of the other disciplines.
- The “max” value represents the greatest emphasis that can typically occur within the latitude provided by the curriculum report for that degree. Each discipline permits students some latitude in choosing an area of specialization and requires that a student’s program of study go beyond the minimums defined in the curriculum report. It report also permits each institution to establish requirements greater than those defined in the five curriculum reports. The “max” value indicates what one might reasonably expect of those who concentrate on the topic within the limits implied by other degree requirements.

Table 3.1: Comparative weight of *computing topics* across the five kinds of degree programs

Knowledge Area	*CE*		*CS*		*IS*		*IT*		*SE*	
	min	*max*	*min*	*max*	*min*	*max*	*min*	*max*	*min*	*max*
Programming Fundamentals	4	4	4	5	2	4	2	4	5	5
Integrative Programming	0	2	1	3	2	4	3	5	1	3
Algorithms and Complexity	2	4	4	5	1	2	1	2	3	4
Computer Architecture and Organization	5	5	2	4	1	2	1	2	2	4
Operating Systems Principles & Design	2	5	3	5	1	1	1	2	3	4
Operating Systems Configuration & Use	2	3	2	4	2	3	3	5	2	4
Net Centric Principles and Design	1	3	2	4	1	3	3	4	2	4
Net Centric Use and configuration	1	2	2	3	2	4	4	5	2	3
Platform technologies	0	1	0	2	1	3	2	4	0	3
Theory of Programming Languages	1	2	3	5	0	1	0	1	2	4
Human-Computer Interaction	2	5	2	4	2	5	4	5	3	5
Graphics and Visualization	1	3	1	5	1	1	0	1	1	3
Intelligent Systems (AI)	1	3	2	5	1	1	0	0	0	0
Information Management (DB) Theory	1	3	2	5	1	3	1	1	2	5
Information Management (DB) Practice	1	2	1	4	4	5	3	4	1	4
Scientific computing (Numerical mthds)	0	2	0	5	0	0	0	0	0	0
Legal / Professional / Ethics / Society	2	5	2	4	2	5	2	4	2	5
Information Systems Development	0	2	0	2	5	5	1	3	2	4
Analysis of Business Requirements	0	1	0	1	5	5	1	2	1	3
E-business	0	0	0	0	4	5	1	2	0	3
Analysis of Technical Requirements	2	5	2	4	2	4	3	5	3	5
Engineering Foundations for SW	1	2	1	2	1	1	0	0	2	5
Engineering Economics for SW	1	3	0	1	1	2	0	1	2	3
Software Modeling and Analysis	1	3	2	3	3	3	1	3	4	5
Software Design	2	4	3	5	1	3	1	2	5	5
Software Verification and Validation	1	3	1	2	1	2	1	2	4	5
Software Evolution (maintenance)	1	3	1	1	1	2	1	2	2	4
Software Process	1	1	1	2	1	2	1	1	2	5
Software Quality	1	2	1	2	1	2	1	2	2	4
Comp Systems Engineering	5	5	1	2	0	0	0	0	2	3
Digital logic	5	5	2	3	1	1	1	1	0	3
Embedded Systems	2	5	0	3	0	0	0	1	0	4
Distributed Systems	3	5	1	3	2	4	1	3	2	4
Security: issues and principles	2	3	1	4	2	3	1	3	1	3
Security: implementation and mgt	1	2	1	3	1	3	3	5	1	3
Systems administration	1	2	1	1	1	3	3	5	1	2
Management of Info Systems Organ'tion	0	0	0	0	3	5	0	0	0	0
Systems integration	1	4	1	2	1	4	4	5	1	4
Digital media development	0	2	0	1	1	2	3	5	0	1
Technical support	0	1	0	1	1	3	5	5	0	1

Again, "min" represents the minimum called for by the curriculum guidelines, and "max" represents the greatest emphasis one might expect in the typical case of a student who chooses to undertake optional work in that area or who graduates from a school that requires its students to achieve mastery beyond that required by the curriculum reports. Because the difference between the "min" and "max" values can be large, programs with the same degree name may differ substantially because of the local choices made in determining their requirements, Both "min" and "max" values refer to what can be reasonably expected in the general case. For any individual student or degree program, the "min" value might be as low as zero and the "max" value might be as high as five, regardless of prevailing curricular standards.

3.1.1. How the table values were determined

Tables 3.1 and 3.2 represent the consensus of judgment reached by the CC2005 Joint Task Force. The task force formulated this consensus from an examination of the discipline-specific body of knowledge found in the most recent curriculum volume for each of the computing disciplines: *computer engineering,*

Table 3.2: Comparative weight of *non-computing topics* across the five kinds of degree programs

Knowledge Area	CE		CS		IS		IT		SE	
	min	max	min	max	min	max	min	max	min	max
Organizational Theory	0	0	0	0	1	4	1	2	0	0
Decision Theory	0	0	0	0	3	3	0	1	0	0
Organizational Behavior	0	0	0	0	3	5	1	2	0	0
Organizational Change Management	0	0	0	0	2	2	1	2	0	0
General Systems Theory	0	0	0	0	2	2	1	2	0	0
Risk Management (Project, safety risk)	2	4	1	1	2	3	1	4	2	4
Project Management	2	4	1	2	3	5	2	3	4	5
Business Models	0	0	0	0	4	5	0	0	0	0
Functional Business Areas	0	0	0	0	4	5	0	0	0	0
Evaluation of Business Performance	0	0	0	0	4	5	0	0	0	0
Circuits and Systems	5	5	0	2	0	0	0	1	0	0
Electronics	5	5	0	0	0	0	0	1	0	0
Digital Signal Processing	3	5	0	2	0	0	0	0	0	2
VLSI design	2	5	0	1	0	0	0	0	0	1
HW testing and fault tolerance	3	5	0	0	0	0	0	2	0	0
Mathematical foundations	4	5	4	5	2	4	2	4	3	5
Interpersonal communication	3	4	1	4	3	5	3	4	3	4

computer science, information systems, information technology, and *software engineering*. It used the results of that examination to define the topical elements of the two tables. That examination also heavily influenced the numerical values assigned to each topic for each discipline. The discipline-specific bodies of knowledge provide some quantifiable data concerning the minimum coverage called for by each discipline for each topic. However, they do not provide information that is sufficient to permit any useful calculation of the *relative* emphasis each discipline places on a given topic.

To generate useful indications of the relative emphasis each discipline places on a given topic, it was necessary that we apply our best judgment as to how to integrate various hard and soft factors into a meaningful metric. Hard factors are the numerical specifications found in the discipline-specific bodies of knowledge. Soft factors include:

- Differences in the perspective endemic to the various computing disciplines. Each of the computing disciplines has its own unique perspective and agenda, as characterized in Sections 2.2 and 2.3, above.
- Differences in the meanings attached to seemingly identical terms. While each computing discipline includes coverage of numerous computing topics, discipline-specific perspectives have the effect of attaching different meanings to the same term. For example, while each discipline requires study of *mathematical foundations, programming fundamentals, networking, operating systems*, etc., the precise set of knowledge and skills associated with these topics varies, sometimes widely, among the disciplines. This issue is addressed more fully in Sections 3.2 and 3.3 and in Table 3.3. In light of the inconsistent use of terms, the only sure way to be clear about what a given topic area means for a given computing discipline is to consult the curriculum volume for that discipline.
- Differences in the latitude for studying optional topics. Despite efforts to make each of the five discipline-specific curriculum reports international in scope, most are implicitly oriented to the U.S. system of defining undergraduate degree programs. In the U.S. system, the amount of technical study that can be required for a degree program in most subjects is significantly limited by the requirement to have a significant General Education component in all degree programs. Such limitations affect all computing degree programs. In addition to this generic U.S. constraint, some computing degree programs must devote a greater portion of their limited resources to specific topics, which in turn limits how much freedom is available to study computing topics that are not required for the degree. For example, degree programs in *computer engineering* must devote significant study to topics required by (a) the engineering profession, and (b) the CE emphasis on hardware-related topics. This curtails how

much study CE students may devote to optional computing topics. Degree programs in *information systems* face similar limitations due to the requirement that IS students study business and organizational topics. The fact that not all computing disciplines have equal flexibility places constraints on degree programs that are difficult to translate in any precise or formal way into our schema of "min" and "max" topical coverage.

Due to the range and nature of these factors, the numerical values assigned in Tables 3.1 and 3.2 reflect a synthesis of hard and soft factors constructed by the CC2005 Task Force. As such they are "fuzzy numbers". One important reality-check is that each numerical value for each topic for each given discipline was deemed satisfactory not only by the representative of that computing discipline but also by the representatives of each of the other computing disciplines. Thus, each value is neither so low as to be unacceptable to the representative of the discipline in question, nor so high that representatives of other discipline found it unjustified.

3.1.2. Using the table: Two related examples

To see how to use Table 3.1, we will consider the fourth and fifth topics in the table: "Operating Systems - Principles and Design" and "Operating Systems - Configuration and Use". Both topics concern operating systems. A reader who is not familiar with the terminology may wish to consult the Glossary:

- Readers who are unsure what "operating systems" are can learn from the glossary that the term "operating system" refers to a specific kind of software (such as Windows, Linux, UNIX, Mac OS, etc.) which permits the human user to interact with a computer. It also enables a computer to manage its resources (memory, disk drives, monitor screen, network interface, etc.) so that it can run whatever application programs (word processor, spreadsheet, web browser, etc.) the user asks the computer to run. In short, an operating system is software that runs "in the background", permitting the computer to operate in useful ways.
- A reader can also learn from the glossary that the former topic, "Operating Systems - Principles and Design", refers to foundational knowledge that enables a student to understand the tasks an operating system must perform. It includes the various strategies and tactics that an operating system might employ to do those things, the kinds of mechanisms that the operating systems designer can use to implement those strategies and tactics, the strengths and weaknesses of various popular approaches, and so on. In addition, we expect that a student will complete a major programming project, either creating an operating system "from scratch" or creating a significant enhancement to an existing operating system.
- Similarly, the reader can learn from the glossary that the latter topic, "Operating Systems - Configuration and Use", is concerned with the practical mastery of the capabilities of mainstream operating system products. Rather than focus on the underlying concepts and principles for designing and implementing operating systems, this topic focuses on developing the student's ability to make full use of the various capabilities provided by existing operating systems. The goal is produce students who know the strengths and limitations of two or more mainstream operating systems, know how those attributes relate to both organizational policy and individual user needs, and know how to use operating systems capabilities to satisfy user needs and implement organizational policy.

The reader can consult Table 3.1 to see how the various disciplines compare in the emphasis they place on each of these two topics:

- For the former topic, "Operating Systems – Principles and Design", we see that both IS and IT programs give it less emphasis than do CE, CS, and SE programs. For typical degree programs in IS and IT, both the "min" and "max" values are "1", which indicates that students in those programs are exposed to some basic concepts and terminology but typically do not study OS principles and design in any substantive way. In contrast, CE, CS, and SE programs have higher "min" and "max" values,

indicating that they have higher minimum standards and a higher ceiling for student mastery. The fact that a higher "min" value is shown for CS and SE (3 each) than is shown for CE (2) indicates that the CS and SE curricula typically feature more coverage of OS principles and design than do CE programs. The fact that a higher "max" value is shown for CS (5) than for CE and SE (4 each) indicates that CS curricula typically have more room in their program of study to permit in-depth coverage of OS principles and design for those who desire it.

- For the latter topic, "Operating Systems – Configuration and Use", we see a different pattern of relative emphasis among the degree programs. While all degree programs provide some experience in using and configuring operating systems, IT programs have the highest "min" and "max" values (3 and 5, respectively). This indicates that IT programs focus their coverage of OS on configuration and use, and expect all their students to obtain significant capability in this area. The other degree programs have weaker minimum expectations (2 each) that are comparable to each other. There is slightly more room in CS and SE programs for optional mastery of this topic (max 4 each) than in CE and IS programs (max 3 each), but all place less emphasis on OS configuration and use than do IT programs.

The comparison provided in Table 3.1 allows us to conclude that a student who wants to understand the principles and design of operating systems will typically not be well served by IS and IT programs, will be better satisfied by CE, CS, or SE programs, and will have the greatest opportunity for in-depth study in CS programs. In contrast, a student who is interested primarily in the practical configuration and use of operating systems will be best served by an IT program, as each of the other degree programs provides less opportunity for in-depth mastery in this area. A student wishing to pursue both OS topics would likely gravitate toward a CS or SE degree program, where he or she will sacrifice depth with respect to practical application to obtain a better balance of principles and application.

3.2. Degree outcomes: Comparing expectations of program graduates

The previous section provides a comparative view of the emphasis of study in the five major kinds of computing degree programs. This section provides a comparative view of the performance capabilities expected of the graduates of each degree program. The previous section summarizes what a student will study; this section summarizes the expectation of the student after graduation.

Table 3.2 lists nearly 60 performance capabilities across 11 categories. For each capability, each discipline is assigned a value from 0 to 5. The value 0 represents no expectation whatsoever while 5 represents the highest relative expectation. As with the values of Tables 3.1 and 3.2, these values are "fuzzy numbers" (determined as described at the end of section 3.1.1). Table 3.3 shows that:

- *Computer engineers* should be able to design and implement systems that involve the integration of software and hardware devices.
- *Computer scientists* should be prepared to work in a broad range of positions involving tasks from theoretical work to software development.
- *Information systems* specialists should be able to analyze information requirements and business processes and be able specify and design systems that are aligned with organizational goals.
- *Information technology* should be able to work effectively at planning, implementation, configuration, and maintenance of an organization's computing infrastructure.
- *Software engineers* should be able to properly perform and manage activities at every stage of the life cycle of large-scale software systems.

In contrast to Tables 3.1 and 3.2, which summarize the *inputs* provided to students by degree programs, Table 3.3 focuses on *outputs*, summarizing the relative capability expectations of graduates.

Table 3.3. Relative Performance Capabilities of Computing Graduates by Discipline

Area	*Performance Capability*	*CE*	*CS*	*IS*	*IT*	*SE*
Algorithms	Prove theoretical results	3	5	1	0	3
	Develop solutions to programming problems	3	5	1	1	3
	Develop proof-of-concept programs	3	5	3	1	3
	Determine if faster solutions possible	3	5	1	1	3
Application programs	Design a word processor program	3	4	1	0	4
	Use word processor features well	3	3	5	5	3
	Train and support word processor users	2	2	4	5	2
	Design a spreadsheet program (e.g., Excel)	3	4	1	0	4
	Use spreadsheet features well	2	2	5	5	3
	Train and support spreadsheet users	2	2	4	5	2
Computer programming	Do small-scale programming	5	5	3	3	5
	Do large-scale programming	3	4	2	2	5
	Do systems programming	4	4	1	1	4
	Develop new software systems	3	4	3	1	5
	Create safety-critical systems	4	3	0	0	5
	Manage safety-critical projects	3	2	0	0	5
Hardware and devices	Design embedded systems	5	1	0	0	1
	Implement embedded systems	5	2	1	1	3
	Design computer peripherals	5	1	0	0	1
	Design complex sensor systems	5	1	0	0	1
	Design a chip	5	1	0	0	1
	Program a chip	5	1	0	0	1
	Design a computer	5	1	0	0	1
Human-computer interface	Create a software user interface	3	4	4	5	4
	Produce graphics or game software	2	5	0	0	5
	Design a human-friendly device	4	2	0	1	3
Information systems	Define information system requirements	2	2	5	3	4
	Design information systems	2	3	5	3	3
	Implement information systems	3	3	4	3	5
	Train users to use information systems	1	1	4	5	1
	Maintain and modify information systems	3	3	5	4	3
Information management (Database)	Design a database mgt system (e.g., Oracle)	2	5	1	0	4
	Model and design a database	2	2	5	5	2
	Implement information retrieval software	1	5	3	3	4
	Select database products	1	3	5	5	3
	Configure database products	1	2	5	5	2
	Manage databases	1	2	5	5	2
	Train and support database users	2	2	5	5	2
IT resource planning	Develop corporate information plan	0	0	5	3	0
	Develop computer resource plan	2	2	5	5	2
	Schedule/budget resource upgrades	2	2	5	5	2
	Install/upgrade computers	4	3	3	5	3
	Install/upgrade computer software	3	3	3	5	3
Intelligent systems	Design auto-reasoning systems	2	4	0	0	2
	Implement intelligent systems	2	4	0	0	4
Networking and communications	Design network configuration	3	3	3	4	2
	Select network components	2	2	4	5	2
	Install computer network	2	1	3	5	2
	Manage computer networks	3	3	3	5	3
	Implement communication software	5	4	1	1	4
	Manage communication resources	1	0	3	5	0
	Implement mobile computing system	5	3	0	1	3
	Manage mobile computing resources	3	2	2	4	2
Systems Development Through Integration	Manage an organization's web presence	2	2	4	5	2
	Configure & integrate e-commerce software	2	3	4	5	4
	Develop multimedia solutions	2	3	4	5	3
	Configure & integrate e-learning systems	1	2	5	5	3
	Develop business solutions	1	2	5	3	2
	Evaluate new forms of search engine	2	4	4	4	4

3.3. International Differences

This report and the five other volumes of the *Computing Curricula Series* on which it is based have benefited to some degree from international input, especially from the United Kingdom. Our goal has been to provide advice and illustrations that would have international relevance. While our future efforts must feature significantly expanded international participation, we can claim some modicum of success, as some interesting points of comparison arose. There are differences in the structure of the academic year, in the emphasis given to the study of computing within a degree program (for example, in the U.K. almost all classes in a computing degree program will be oriented toward computing), in the quality control mechanisms (e.g., different expectations and practices re: accreditation as discussed in section 4.5), and so on. In addition, there are different approaches to defining the focus of degree programs:

- In the US there is a very strong sense of a "core" for each discipline. The core for each discipline is intended as a specification of those elements of the discipline that are deemed fundamental and which all students of that discipline should fully understand. This approach has at least two benefits: it helps create a shared understanding of the abilities that can be expected of graduates, and it facilitates transfer between institutions. Argument and eventual agreement about the definition of the core helps support strong discipline definitions and a clear understanding of the meaning of degree titles.

- The idea of a core is far less prominent in countries such as the U.K. where degree titles are seen as providing strong marketing opportunities. As a result, by U.S. standards the U.K. offers a vast number of kinds of computing degrees. The rich variety of degree titles reflects different emphases and different career opportunities. In the U.K., guidance and quality issues associated with degrees are taken care of by criteria and mechanisms such as those outlined in section 3.5

The Joint Task Force did not have access to sufficient information about models from other nations. A thorough evaluation of various approaches applied around the globe is an important step for future work.

3.4. The pace of change in academia: The *disciplines* and the *available degrees*

As discussed previously, the landscape of the computing degree programs has undergone dramatic change as a result of the explosion of computing during the 1990s. The computing field has evolved to the point that we are seeing the emergence of degree programs that are focused on the challenges that now confront both the computing profession and our computing-dependent society as a whole.

The evolution of computing discussed in Chapter 2 often is not reflected in the kinds of degree programs that are offered. Very few North American colleges and universities offer all five major kinds of the computing degrees. The two newer kinds of degrees, IT and SE, are less common than are degrees in the more established disciplines (CE, CS, and IS). Institutions tend to be cautious and conservative, and the complex nature of academic degree programs means that it is difficult to implement significant changes rapidly. Thus, at some institutions the choices of computing degree programs look more like the "pre-1990s" view shown in the top portion of Figure 2.1 than the "post-1990s" view. This is because the pace of change in computing is quite rapid while the pace of institutional change generally is quite slow.

This natural institutional lag can create problems for students who are trying to choose a computing-related degree program that fits their interests and goals. It can also create problems for educators who are trying to ensure that their educational programs are providing up-to-date programs that serve their constituents. At many institutions, faculty seek to minimize this lag by using the final year of a degree program to offer students a chance to learn about areas of cutting-edge specialization and research.

Despite the natural lag, there is widespread evidence that important and fundamental changes have taken place. In looking at the development of computing over the last twenty or so years, one can observe that there has been a dramatic shift in emphasis towards interaction, and perhaps away from the study of the algorithm. The move towards interaction can be viewed as an important mark of computing's success: it highlights the fact that a broad range of people are now using and interacting with computers to a far greater degree than in the early days. It is very natural therefore that new programs of study reflect this fact, and both the large number of IS programs and the recent emergence of IT programs are manifestations of this. Indeed, there is scope for an even richer variety of possibilities, and we fully expect to see change in computing continue.

Because institutional lag has consequences, it is important that readers of this report recognize its existence and look for indications that a given institution is taking steps to overcome it. In general terms, there are certain metrics or marks of quality which suggest that institutional lag is not a serious problem for a given program or department. Among these indicators are:

- The existence of an active advisory board that features the involvement of advisors from industry;
- The involvement of students in committees that vet programs and help define goals for change;
- Institutional quality control mechanisms which actively seek and obtain advice from external experts;
- Employment statistics of graduates, which can provide indications of the reputation of a program's graduates and of the institution itself;
- Accreditation of degree programs (discussed in section 4.5), which provides a mark of quality and addresses issues such as the currency of the program of study

Beyond this, there are discipline-specific considerations, which we summarize these below.

3.4.1. Computer engineering

The change from the "pre-1990s view" of computing to the "post-1990s view" typically includes a more complete emergence of *computer engineering* from within *electrical engineering*. Most often, both kinds of programs exist in the same academic department, although many computer engineering programs reside in joint departments of computer science and engineering. Although not true on the international stage, in the U.S. there has always been a single home for those who study computer hardware. It used to be electrical engineering. It has become computer engineering. At most universities, the change from the "pre-1990s" to "post-1990s" view has already taken place. However, even some of the very best schools do not offer a separate computer engineering program. At some schools, electrical engineering and computer science reside in the same school, and computer engineering is seen as a natural merging of interests among faculty in those two disciplines. If a given institution's departmental structure does not feature an explicit recognition of computer engineering, it does not necessarily imply any shortcoming.

There are notably fewer programs in *computer engineering* than in *computer science* or *information systems*. In the U.S., this is because most American colleges do not offer engineering programs of any kind. American engineering programs reside disproportionately in relatively large universities or specialized institutes which can meet the requirements for offering accredited engineering degrees. We do not expect this to change significantly in the future.

3.4.2. Computer science

In the U.S., there are far more degree programs in computer science than in any other computing discipline. The great majority of colleges and universities offer a CS degree. To some extent, this is an historical artifact: *computer science* was the only substantive computing discipline that focused explicitly on software development when academic computing degree programs emerged in the 1970s. When most colleges created their computing degree programs, computer science was the only choice that had strong

ties to mathematics, science, and/or engineering. (IS programs developed around the same time, but their primary ties were to business schools.) At smaller schools, the CS degree program may be housed in a Mathematics and CS department, but almost all of these departments offer two distinct degree programs.

The increased diversity we see in the post-1990s computing disciplines is fairly localized in areas that affect computer science. The new computing disciplines target career areas that traditionally have been filled by graduates of CS programs.

Currently, there is an ongoing discussion regarding the relationship between what *computer science programs teach* and what most *graduates of computer science actually do* in their careers. To understand this discussion, it is necessary to review the characterization of computer science provided in Section 2.3.2. The work of computer scientists falls into three categories: designing and implementing software; devising new ways to use computers; and developing effective ways to solve computing problems. Let us consider what is involved in a career path in each area:

- *Career Path 1: Designing and implementing software*. This refers to the work of software development, which has grown to include aspects of web development, interface design, security issues, mobile computing and so on. This is the career path that the majority of computer science graduates choose. While a bachelor's degree is generally sufficient for entry into this kind of career, many software professionals return to school to obtain a terminal master's degree. (Rarely is a doctorate involved.) Career opportunities occur in a wide variety of settings, including large or small software companies, large or small computer services companies, and large organizations of all kinds (industry, government, banking, healthcare, etc.). Degree programs in *software engineering* also educate students for this career path.
- *Career Path 2: Devising new ways to use computers*. This refers to innovation in the application of computer technology. A career path in this area can involve advanced graduate work, followed by a position in a research university or industrial R&D lab, or it can involve entrepreneurial activity such as was evident during the "dot com" boom of the 1990s, or it can involve a combination of the two.
- *Career Path 3: Developing effective ways to solve computing problems*. This refers to the application or development of computer science theory and knowledge of algorithms to ensure the best possible solutions for computationally intensive problems. As a practical matter, a career path in the development of new computer science theory typically requires graduate work to the Ph.D. level, followed by a position in a research university or an industrial R&D laboratory.

Computer science programs generally intend to prepare students for these three career paths. In addition, there is a fourth career path that CS programs do not target but nonetheless draws many computer science graduates:

- *Career Path 4: Planning and managing organizational technology infrastructure.* This refers to the work for which the new *information technology* (IT) programs explicitly aim to educate students.

Of these four career paths, career paths 2 and 3 are important elements of the identity of *computer science* and are the kind of career paths that many computer science faculty wish to see their students choose. As a practical observation, however, only an extremely small minority of the students who earn *computer science* bachelor's degrees choose them. For those few who do, institutional lag is not an issue: a strong bachelor's degree program in computer science, followed by graduate study (probably to the doctoral level) is clearly the preferred choice.

Career paths 1 and 4 are the focus of debate. These careers draw the overwhelming majority of *computer science* graduates. They are also the focus of the new computing degree programs (*software engineering* and *information technology*, respectively) which have come into being to provide more focused alternatives to *computer science* programs at preparing students for these career paths. In addition, a

significant number of graduates in information systems have over the years selected organizational roles that are very similar to these career paths. No resolution has yet occurred to the debate about the relative value of *computer science* programs vs. *software engineering* and *information technology* programs. However, the issues that arise in the debate are well defined and are discussed in the subsequent subsections on *software engineering* and *information technology*, respectively.. For career paths 1 and 4, one should evaluate the relative value of *computer science* programs in the light of emergence of degree programs in the two new computing disciplines.

3.4.3. Information systems

The change in the role of *information systems* concerns the expanded role of information technology in organizations of all kinds. Historically, *information systems* programs prepared students to work with functionally-oriented business applications, such as payroll, accounts receivables, inventory management, etc. On the technology side, IS students could expect to become familiar with computer applications related to these traditional business areas, especially database management systems, and with spreadsheets and other off-the-shelf software products that had broad utility to business people.

Modern IS programs focus on the broader role of IT-enabled information utilization and business processes in a wide range of enterprises, while still maintaining their close association with business schools. What information does the enterprise need? How is that information generated? Is it delivered to the people who need it? Is it presented to them in ways that permits them to readily use it? Is the organization structured to be able to utilize technology effectively? Are the business processes of the organization well-designed? Do they utilize the opportunities created by information technology fully? Does the organization fully utilize the communication and collaboration capabilities of information technologies? Is the organization capable of adapting quickly enough to changing external circumstances? These are the important issues that various types of enterprises increasingly rely on IS people to address.

For IS programs, the traditional role still exists, but it is no longer sufficient. The meaningful question is: "Has an IS program broadened its scope to include an integrated view of the enterprise with complex information needs and high-level dependency on IT-enabled business processes?" Web-based distributed technologies provide the infrastructure for globally connected organizations, and modern IS programs have to address the needs of such organizations. IS students must learn how to assess and evaluate organizational information needs, specify information requirements, and design practical systems to satisfy those requirements. If a program focuses only on the design and development of narrow functional applications and the use of personal productivity tools, it is seriously behind mainstream IS programs.

In addition to these concerns, IS has to take into account the emergence of IT programs. Traditionally, many graduates of IS programs have functioned in roles that are similar to the roles for which IT programs explicitly prepare their students. As the number of IT programs grows, many IS departments will have to evaluate how to define and serve their core constituents.

3.4.4. Information technology

In the last few years, degree programs in *information technology* have emerged and developed to such an extent that they now should be an important part of any discussion of computing degree programs. As summarized in Section 2.3.4, IT programs focus on producing graduates who know how to make information technology work in a wide range of settings. Organizations of all kinds have become dependent on networked computing infrastructure to such an extent that they cannot function without that infrastructure. IT people are prepared to select, manage, and maintain that infrastructure, ensuring that it meets organizational needs. They also create digital content for that infrastructure and take care of IT-support needs of individuals who use it.

The emergence of IT programs represents a grass-roots movement by computing educators to respond to the very real needs of both their local communities and their students. IT programs exist, not because *computer science* or *information systems* programs failed to "do their job", but because those disciplines each define themselves as having a different job. The existence of IT programs reflects one part of the evolution of career opportunities in computing.

Only a few years ago, computing educators in the U.S. were not familiar with IT degree programs, although similar programs have existed for years elsewhere. Today, there are many such programs, and we expect to see this number increase further in the years to come. It was not until 2001 that college-level IT educators in North America began to organize, and in the short period since, they have formed a professional organization, held several conferences, and have made substantive progress in developing curriculum and accreditation guidelines for IT degree programs. It is no exaggeration to say that IT programs have exploded onto the scene in the U.S.

Some people question whether IT programs are a passing fad. Others ask if IT programs are too technical in nature to deserve the status of an academic discipline. People asked similar questions about computer science more than thirty years ago, yet after a number of years the great majority of North American colleges began to offer CS degrees. We may well see similar results with respect to IT. IT degree programs address an important need that is a widespread throughout society. To the extent that organizations rely on computer technology, the IT discipline has a key role to play.

There are two important issues here:

- *Rigor*: Planning and managing an organization's IT infrastructure is a difficult and complex job that requires a solid foundation in applied computing, as well as management and people skills. Those in the IT discipline require special skills – in understanding, for example, how networked systems are composed and structured, and what their strengths and weaknesses are. There are important software systems concerns such as reliability, security, usability, and effectiveness and efficiency for their intended purpose; all of these concerns are vital. These topics are difficult and intellectually demanding.
- *Acceptance*: In the U.S., the IT discipline is "the new kid on the block" and, as a result, faces problems of acceptance among the more established disciplines. This is a natural phenomenon, and it will take time and experience for those in the more established computing disciplines to evaluate and recognize the value that the IT discipline provides. IT is seeking to establish itself as a discipline with its own intellectual core, a rigorous curriculum and accreditation guidelines. To the extent that it succeeds at these challenges, acceptance and respect will naturally follow.

At many institutions, administration is motivated to see an IT program created to respond to community needs and to provide more choices for prospective students. Whenever an institution creates an IT program, it must take special care to ensure that it implements the program properly. One must ensure that the people who are responsible for an IT degree program recognize IT's importance and are excited about creating high-quality educational experiences for IT students.

3.4.5. Software engineering

The development of SE is a response to a very real problem: a shortage of degree programs that produce graduates who can properly understand and develop software systems. CS programs have shown that they can produce students who have sound skills in programming fundamentals. However, many believe that they have not been successful at reliably producing graduates able to work effectively on complex software systems which require engineering expertise beyond the level of programming fundamentals. In the post-1990s world, many software projects are large and complex, and there is a pressing need for software engineers who can apply professional practices that ensure that software is reliable, meets the goals and needs of users, and is produced on schedule and within budget. SE programs represent an effort

to make the undergraduate experience more successful at providing students with an adequate set of knowledge and skills for careers as software professionals.

As a practical matter, CS and SE degree programs often have much in common. Both disciplines recognize that the subject matter of computing has grown to the point where no single degree program can expect its students to grasp the entire field. Thirty years ago, it was reasonable to expect CS undergraduates to "study everything"; now, there is too much of it to fit in a four year (or even five year) program of study.

Both CS and SE curricula typically require a foundation in programming fundamentals and basic CS theory. They diverge in what they focus on beyond those core elements. CS programs tend to keep the core small and then expect students to choose among more advanced courses in areas of CS concentration (such as systems, networking, database, artificial intelligence, theory, etc.). In contrast, SE programs generally expect students to focus on a range of topics that are essential to the SE agenda (problem modeling and analysis, software design, software verification and validation, software quality, software process, software management, and so on). While both CS and SE programs typically require students to experience team project activity, SE programs tend to involve the students in significantly more of it, as effective team processes are essential to effective SE practices. In addition, a key requirement specified by the SE curriculum guidelines is that SE students should learn how to build software that is *genuinely* useful and usable by the customer and satisfies all the requirements defined for it.

Two questions about how SE programs will develop remain unanswered:

- *To what extent will degree programs in SE emerge from within CS departments as a side-by-side alternative to the traditional CS degree?* Some believe that such a trend is inevitable, while others do not believe that such a development is necessary. There remains a difference of opinion among faculty about the nature and amount of rigorous SE experience that a robust undergraduate education requires in order to prepare students to function as competent software professionals.
- *To what extent will the implications of using the word "engineering" cause American colleges and universities to choose a different name for their SE degree programs?* In the U.S. and certain other countries, the professional engineering community is protective of its identity and the word "engineering" looms large in that identity. In some states and in Canada, certain uses of the word "engineering" are regulated by law. Software engineering differs from the more traditional engineering disciplines (due to the intangible nature of software, and to SE's focus on human processes rather than the laws of physics, for example). Yet the traditional strength of engineering (robust methods for creating reliable artifacts) is at the core of the SE agenda. Having the name "engineering" as part of "software engineering" has implications in the U.S. for accreditation and, at some institutions, for the location of SE within the university. These implications may cause some U.S. programs to shy away from adopting the name SE. In the U.K., such issues were successfully resolved decades ago.

It appears that most CS students anticipate that their professional careers will involve doing software development work. We expect that, if a large number of SE degree programs were available as an option for CS students, many would select it. At present, that choice is not widely available. There are now about 30 SE degree programs in America. We expect this number to increase, although not at the same rate that we have seen for IT programs.

Within CS degree programs, the robustness of SE education varies widely. The most recent curriculum guidelines for CS (CC2001) require some minimal coverage of SE. Most CS degree programs go beyond the minimum and provide one or more SE courses. Some CS programs offer SE as one of several areas of CS concentration. When evaluating CS programs that provide such an option, it is important to look closely at how rigorous that option is.

3.5. The pace of change in the workplace: *Degrees* and *career opportunities*

Just as it takes time for academic institutions to adapt, it also takes time for industry and other players in the computing workplace to do the same. As a result, some job opportunities will reflect the "pre-1990s" model shown at the top of Figure 2.3. In other words, because new computing disciplines are new, no one yet expects people to have those qualifications.

With respect to CE, the new disciplines are not much of an issue. It is clear where people go for expertise in hardware and related software. CE has adopted this role from EE, and the change is largely complete. CE, like EE, is part of the professional engineering community, and in the U.S. this has implications for where one can earn a CE degree. Not every university has an engineering program. Employers looking for people with a CE background want graduates of an accredited CE program.

It is also clear where people look for expertise at the interface between the information needs of business and computing. IS occupies that role in both the "pre-1990s" and "post-1990s" landscape. It is essential that those who pursue this path make sure that they find an IS program that continues to update its vision and has an emphasis that fits the interests of the prospective student. For prospective IS students, the task of finding the right program is complicated by the variety of names used by such programs.

Most people who now function in the U.S. as serious software engineers have degrees in CS, not in SE. In large part this is because CS degrees have been widely available for 30 years and SE degrees have not. The story is similar for people working in the IT profession: most IT professionals who have computing degrees come from CS or IS programs. It is far too soon for someone who wants to work as a software engineer or as an information technology practitioner to be afraid that they won't have a chance if they don't graduate from a degree program in one of the new disciplines. In general, a CS degree from a respected program is the most flexible of degrees, and can open doors into the professional worlds of CS, SE, IT, and sometimes CE. A degree from a respected IS program allows entry to both IS and IT careers.

This situation presents ambiguity to students trying to determine what discipline to study and to educators who are trying to decide how their programs can best serve their constituents. We identify the various factors that educators should consider in *Chapter 4*. We focus on the factors students should consider in the *Guide*, which is the second part of this report and is also available as a standalone document.

Media attention to "outsourcing", "off shoring" and "job migration" has caused many to be concerned about the future of computing-related careers. It is beyond the scope of this report to address these issues. The British Computer Society's report, which addresses these issues as they impact the U.K., is available at: http://www.bcs.org/BCS/News/PositionsAndResponses/Positions/offshore/offshorereport.htm. ACM's *Job Migration Task Force* is expected to issue a report during the last quarter of 2005. The ACM report is intended to reflect an international perspective, not just a U.S-centric one. Once completed, the ACM report will be available at http://www.acm.org/jmtf.

3.6. Shared identity: Common requirements of computing degrees

As we have seen, each of the major computing disciplines has its own character. Each one is somewhat different from its siblings in emphasis, goals, and capabilities of its graduates. Yet they have much in common. Any reputable computing degree program should include each of the following elements.

1) Essential and foundational underpinnings of its discipline. These may be abstract, e.g., formal theory rooted in mathematics, or they may address professional values and principles. Regardless of their form or focus, the underpinnings must highlight those essential aspects of the discipline that remain unaltered in the face of technological change. The discipline's foundation provides a touchstone that transcends time and circumstance, giving a sense of permanence and stability to its educational mission. Students must have a thorough grounding in that foundation.

2) A foundation in the concepts and skills of computer programming. The foundation has five layers:
   a) An intellectual understanding of, and an appreciation for, the central role of algorithms and data structures;
   b) An understanding of computer hardware from a software perspective, e.g., use of the processor, memory, disk drives, display, etc.
   c) Fundamental programming skills to permit implementation of algorithms and data structures in software;
   d) Skills that are required to design and implement larger structural units that utilize algorithms and data structures and the interfaces through which these units communicate
   e) Software engineering principles and technologies to ensure that software implementations are robust, reliable, and appropriate for their intended audience.

3) Understanding of the possibilities and limitations of what computer technology (software, hardware, and networking) can and cannot do. This foundation has three levels:
   a) An understanding of what current technologies can and cannot accomplish;
   b) An understanding of computing's limitations, including the difference between what computing is inherently incapable of doing vs. what may be accomplished via future science and technology;
   c) The impact on individuals, organizations, and society of deploying technological solutions and interventions.

4) Understanding of the concept of the lifecycle, including the significance of its phases (planning, development, deployment, and evolution), the implications for the development of all aspects of computer-related systems (including software, hardware, and human computer interface), and the relationship between quality and lifecycle management.

5) Understanding of the essential concept of process, in at least two meanings of the term:
   a) Process as it relates to computing, especially program execution and system operation;
   b) Process as it relates to professional activity, especially the relationship between product quality and the deployment of appropriate human processes during product development.

6) Study of advanced computing topics that permits students to visit and understand the frontiers of the discipline. This is typically be accomplished via inclusion of learning experiences that lead students from elementary topics to advanced topics or themes that pervade cutting-edge developments.

7) The identification of and acquisition of skill sets that go beyond technical skills. Such skill sets include interpersonal communication skills, team skills, and management skills as appropriate to the discipline. To have value, learning experiences must build such skills (not just convey that they are important) and teach skills that are transferable to new situations.

8) Exposure to an appropriate range of applications and case studies that connect theory and skills learned in academia to real-world occurrences to explicate their relevance and utility.

9) Attention to professional, legal and ethical issues such that students acquire, develop and demonstrate attitudes and priorities that honor, protect, and enhance the profession's ethical stature and standing.

10) Demonstration that each student has integrated the various elements of the undergraduate experience by undertaking, completing, and presenting a capstone project.

Chapter 4: Institutional considerations

4.1. Evolution and status of computing degrees programs

Most academic institutions face important decisions about the best way to evolve their computing degree programs. However, not all colleges and universities face the same set of challenges. Fortunately, we can characterize a small number of program attributes in such a way that people at the great majority of North American colleges and universities will be able to recognize their circumstances among those parameters. We characterize institutional attributes below, taking each computing discipline in turn:

Computer Engineering: Very few schools face a dilemma about whether to offer a CE program. For institutions that have a School of Engineering (or analogous unit), a CE program almost certainly exists already in some guise. Most often, it is in the form of an explicit CE program, though some institutions provide the same concentration in the form of a combined effort of their CS and EE programs. As of mid-2004, approximately 500 programs exist in computer engineering in the U.S. and thousands more exist elsewhere. A North American institution that does not have an engineering program is not likely to succeed at creating an accredited CE program, since accreditation requirements are difficult to meet without a substantial supporting administrative structure.

Computer Science: Very few schools face a dilemma about whether to offer a CS program, simply because virtually all colleges and universities already have one. CS is by far the most ubiquitous of computing degree programs. The most common scenario is that computer science grew out of a mathematics department in the 1970s. Frequently, that department morphed into a "Mathematics and Computer Science" department for a decade or two before splitting apart into separate departments for each of mathematics and CS in the 1990s, though a considerable number of Mathematics and Computer Science departments remain today. At universities that had an electrical engineering program, it was common for CS to grow out of the electrical engineering or math department, or both. While these scenarios are most common, alternative scenarios abound. During the 1990s, a small number of institutions recognized computing's pervasive influence by creating an autonomous unit, such as a school or college of computing, a move which displayed both insight and foresight. Regardless of where the institution placed the program, most saw their CS programs struggle for local academic legitimacy in the 1970s and 80s, then emerge as established and prominent during the 1990s. At most colleges, the CS program is (or should be) facing serious choices about two closely related issues: (1) the appropriateness of a narrow CS identity in light of the issues discussed in Section 3.2; and (2) how it can best respond to the emergence of newer computing disciplines, also in light of Section 3.2 issues.

Information Systems: As of mid-2005, there are about 1000 IS programs in the U.S., although they go by a wide variety of names, such as *information systems, management information systems* and *computer information systems*. Hundreds of IS programs exist elsewhere in the world, again with a variety of names. Like computer science, the early programs appeared in the 1960s, with most appearing in the 1970s and 1980s. Also like CS programs, IS programs now face fresh competition from IT programs. Faculty at some IS programs are happy to see new IT programs take responsibility for areas that were of little interest to them; others fear the loss of influence and "turf". Regardless of the local IS reaction to the emergence of IT, there are many potentially rich opportunities for cooperation between the IS and IT communities.

Information Technology: As of mid-2005, about 70 American institutions have IT degree programs. Because of the rapid growth of such programs, it is difficult to know how many currently exist. Many of the existing programs are truly in the vanguard: they are high quality programs that have been forging ahead into uncharted territory, creating new offerings tailored to the special needs of this emerging field. Unfortunately, some IT programs are of low quality and fail to serve either their students or their

communities in a responsible way. The latter group of programs seeks to increase enrollments and/or provide the image of being responsive to local needs by "creating" an IT program that is little but a repackaging of existing courses offered by other disciplines. Most institutions don't yet have an IT program, but many are in the process of deciding whether to start one. Those who choose to do so should take special care to ensure that they are following the leadership example of the high quality IT programs, not the exploitive example of low-quality programs. A powerful driver that has lead to the development of the upcoming IT2005 curriculum volume is the desire to provide substantive guidance for those trying to develop high quality IT programs.

For those who are considering an IT program a key question is, "Does the IT agenda fit with the institution's mission?" Opting to create an IT program implies a decision to respond to community needs for well-prepared IT practitioners by creating an overtly career-oriented program. This differs from the traditional academic focus of programs in the arts and sciences, and thus implies some basic questions. What is the institution's mission? Is it local, regional, or national in scope? What responsibility, if any, does the institution have to meet local and regional needs?

Software Engineering: As of mid-2005, more than 30 American institutions have dedicated SE degree programs (far fewer than exist in the U.K. and Australia). Many of these are vanguard programs that have recognized the need to prepare students for creating large and/or safety-critical software. Their educational program focus student preparation for such needs more completely than CS and CE programs can be expected to do. We expect the number of SE programs to increase, although not at the same rate of growth we expect to see in IT programs. SE programs provide an opportunity for an institution to distinguish itself in ways that directly addresses academic and professional challenges.

Some U.S. institutions that offer an SE degree program may opt to use a degree title that does not include the word "engineering". In North America, conflicts can arise due when the word "engineering" is used in the name of a degree program that is housed outside an engineering college. As a result, we expect some high quality programs to use another title. In some cases, SE-like programs will take the form of a standalone degree program. In other cases, they may exist as an especially substantive concentration within a CS degree program. The content of the program, not the label on the degree, determines whether it provides high quality SE education in the sense described in this document.

For institutions considering a SE program, a key question is, "Does SE warrant a dedicated program?" This implies a local consensus about whether the CS agenda provides students with sufficient preparation for a career as a professional software engineer. If the local consensus sees the need for an explicit SE program, and if adequately qualified faculty are available, the question becomes one of implementation structure. Should it be handled as one of several optional concentrations within CS? Or does it imply programmatic differences in terms of structure and requirements?

4.2. The portfolio strategy

The increased diversity of computing degree programs presents important challenges and opportunities. Computing educators face more choices and tougher decisions. Being responsive implies adaptation and change and, by their nature, institutions find it hard to adapt and change. This is especially true in times of tight budgets, such as those which many institutions are facing as this document is written.

While fiscal constraints provide a justification for dismissing the prospect of revising or initiating computing degree programs, inaction can have its own consequences. The new diversity of computing programs gives institutions an unprecedented opportunity to focus their degree programs to meet the needs of their students, communities, and other constituents in the most effective way. It allows schools to demonstrate initiative, foresight, and responsiveness in very tangible ways.

One of the great potentials of the new diversity of computing degree programs is that it permits academia to bring its computing degree programs in line with the diverse needs that exist among students and in local communities. In the past, many institutions had little choice but to have a CS program on the technical side and perhaps an IS program on the business side. Now, a college might offer a portfolio of degree programs to serve various student needs more appropriately:

- A CS program to serve those students who wish to proceed as generalists in computing or who aspire to graduate study, research positions, or cross-disciplinary innovation.
- An SE program to serve students who have the intellectual and technical aptitude to excel as software developers and who want to become expert at developing large scale software, working in teams, and producing robust products that meet customer needs.
- An IT program to serve students who want a computing career that features a mix of technical and people issues rather than a unilateral focus on technology, and who are attracted to the widespread need for IT professionals in a variety of organizations and settings.
- An IS program in cooperation with other business programs to serve students who want a career that focuses on the information needs of organizations and who are interested in technology primarily as a vehicle to meet such needs.
- At those schools that have an engineering program, a CE program to serve students who want a career that is focused on developing computer-based devices.

"Diversity" and "diversify" have the same root, and recent patterns of computing enrollments suggest that a diversification strategy, useful in insulating one's investments from erratic swings in value, may be similarly useful in protecting an institution's computing programs from unpredictable enrollment swings. Recent events provide a timely illustration. CS programs have long seen cyclical enrollment patterns. In recent years, the "dot.com" boom saw unprecedented CS enrollments. When that bubble burst, CS enrollments plunged. More recently, when many expected to see a gradual return to rising CS enrollments, many programs have seen just the opposite: further declines in CS enrollments. These facts do not mean that computing degree programs are less popular. In the face of widespread reports of falling enrollments in CS, various institutions report that their SE enrollments remained steady and that IT enrollments continued to increase. Some schools that provide not only CS but also IT and/or SE programs are not reporting big downturns in aggregate computing-degree enrollments and instead report net increases in computing enrollments despite CS decreases.

Such evidence is purely anecdotal. Many factors may be in play, and hard evidence is not readily available. However, such anecdotal reports provide interesting food for thought. There is good reason to suspect that a portfolio strategy might help institutions better meet the needs of their students:

- To the extent that a given institution's student population presents a range of interests and sets of abilities, a broader range of computing degree choices may permit the institution to do a better job of serving that range of students needs.
- At many institutions, student retention is an important concern. Schools routinely report that 50% or more of those students who initially choose CS study soon decide to abandon it. It seems plausible to expect that the better the match between student interests and abilities and available degree programs, the better the retention level that can be achieved.
- Apart from retention, computing educators (and others) have long been concerned with the relatively narrow profile of students who are attracted to the computing disciplines. A broader portfolio may prove to attract a wider population. For example, some schools that offer IT programs report increased participation by women and minorities but, again, the evidence is anecdotal.

- When an academic unit offers a family of quality computing degree programs, its faculty will naturally come to reflect a variety of perspectives on computing issues and challenges. This kind of situation can set the stage for a useful cross-fertilization of ideas among the disciplines, which may in turn support creativity and innovation in both teaching and research.

Many academics are asking themselves about program diversification quite explicitly: "What portfolio of computing degree programs should we be offering?" This implies an important educational question that was rarely asked before the emergence of the new computing disciplines: "What kinds of computing education best serve our students and community?" The portfolio question also requires that decision-makers weigh the benefits against the costs. Obvious costs include the initial costs of creating new programs as well as the ongoing overhead of administering multiple programs.

How do the potential benefits and costs compare? There is no standard answer, and each institution must make a decision in light of its own mission and circumstances. For some institutions the answer will be easy, while others will face difficult choices. The key question is clear: "What options best serve our students, our community, and our future?" Even schools facing resource constraints which preclude immediate programmatic initiatives can gather key people to consider this important question.

Any serious consideration of program diversification is likely to see certain issues arise. In the next section, we discuss key factors that will likely come into play.

4.3. Institutional challenges to diversity

For any college or university trying to come to terms with the new diversity of computing degree programs, there are at least three areas in which effective leadership and a willingness to change are necessary. These areas are *faculty development and adaptation, organizational structure,* and *curricular structure*. Each of these areas involves issues that, by their nature, invite polarities of opinion among faculty. As is often the case when issues elicit strong differences of opinion, there are implicit value choices that underlie the explicit issues. Any honest and thorough planning effort concerning an institution's computing degree programs should not only face these issues themselves but should also examine the fundamental value choices that underlie differences of opinion in each of these three areas.

4.3.1. Faculty development and adaptation

When an institution that currently offers a CS degree decides to diversify by expanding its mission to include SE and/or IT programs, it is likely that some difficulty will arise with respect to finding appropriate faculty. After all, for most institutions, CS faculty are the only available computing faculty. Most of them will naturally be oriented to the CS mission that shaped their own professional growth and development. Suggesting that they broaden their mission to encompass fully the SE and/or IT agenda is likely to produce a mixed reaction along each of the following lines:

- *Legitimacy:* Some CS faculty will have the view that SE and/or IT have not yet developed to the stage where they can be considered as academic disciplines. Some will argue that SE deserves a course or two within the context of a CS major, but does not warrant or require a distinct programmatic focus. Some will argue that IT offers an agenda that is too vocational. Other CS faculty will be more aware of the important societal functions provided by each of the scientific (CS), engineering (SE), and practitioner (IT) professions, and therefore will be more inclined to see a need to broaden the computing education agenda. The latter group can play a valuable role in persuading a skeptical faculty of the validity and importance of the engineering perspective that is central to SE and the practitioner perspective that is central to IT.
- *Preparedness:* Most CS faculty will not have the necessary background to teach immediately courses which substantially differentiate SE and IT from CS. This is a natural consequence of the fact that their

background is in CS rather than SE or IT. It is important that these issues are not swept under the carpet, as faculty shortcomings can be disguised to the detriment of all. This risk is especially present because CS faculty will be knowledgeable about most computing topics but with a CS orientation that falls short of the needs of SE and/or IT students. For example, both CS and IT students need to know about computer networks. Much of the material is equally important to both CS and IT, but after a point the agenda splits: CS will emphasize underlying models and principles, while IT will emphasize practical application skills related to network management and security. In practice, most CS faculty have never managed a network or been responsible for maintaining network security. Similarly, most CS faculty will be able to teach a CS course "about SE" but have not had occasion to obtain the knowledge and experience to teach a full-fledged SE agenda that helps students become software engineers. Care must be taken in at least two dimensions:

1. A CS treatment of shared topics masquerading as SE or IT coursework is quite inappropriate;

2. If CS faculty are to develop the ability to teach SE and IT courses, it is necessary that they be provided substantial resources to help them prepare and adapt to the very different agendas implied by the engineering and practitioner perspectives.

With respect to both legitimacy and preparedness, the core challenge is to enlist faculty support for new programs and to respond by providing faculty with the support they will require. A high-level decision for a broader computing agenda may well be necessary, but it is unlikely to be sufficient. Some faculty will be more adaptable and more willing to embrace the challenges presented. Successful efforts will identify faculty who want to develop new capabilities, and then provide support for them to undertake the kind of self-education and preparation that is needed for them to succeed.

4.3.2. Organizational structure

Diversification presents choices about how and where computing degree programs should be housed. Should each discipline have its own department? Should they all reside within a single multifaceted unit, such as a Department of Computing? Is it best to bypass traditional departmental structures altogether and create a "Center" to house new programs? There are proponents for every approach. There is no compelling answer as to which is best, and each institution must consider its own context and mission.

Each of the computing disciplines features a distinct character and focus. At the same time, their teaching-and-learning agendas are far from distinct, as many topics and skills are relevant across the various kinds of degree programs. This fact creates a natural tension between the advantages of having distinct, separate departments and the advantages of offering different degrees from within a single computing department. The former approach makes it easier to ensure that each degree program is free to do its job properly without significant compromise. The latter approach may make it easier to leverage commonalities among the disciplines so that economies of scale can replace redundancy in curricula, computer labs, teaching loads, etc.

Those contemplating structural changes should be aware that we expect change and innovation to be ongoing. While SE and IT degree programs are relatively new phenomena, they won't be the last word in new computing degree programs. As institutional leaders answer the question of "What to do with SE and/or IT?" they should also ask themselves what they might do with the next new kind of computing-related program, whatever it might be. If a new organizational structure is being created, what is its purpose: to solve the immediate dilemma or set the stage for the long run?

4.3.3. Curricular structure

Regardless of whether computing degree programs are housed in the same academic unit or in different departments, the commonality that exists across the computing disciplines invites questions about the role and purpose of various courses, especially introductory courses. Two related issues naturally arise:

- *Filter vs. funnel approaches:* At the introductory level, there are two basic philosophies to course targeting which can be described as "filter" and "funnel". The *filter* approach calls for curricula that implement an ambitious, tightly focused discipline-specific agenda from the very first course. *Filter curricula* use introductory courses to lay disciplinary foundations early and/or establish a rigorous performance standard from the beginning and immediately filter out students who don't rise to the standard. In contrast, the *funnel* approach calls for curricula that serve a student audience that is broader than those who will concentrate and succeed in a given discipline. *Funnel curricula* use introductory courses to provide students with learning experiences that will help them make a well informed choice as to whether a given discipline is suitable for them.

  The *filter approach* calls for parallel discipline-specific introductory course sequences, one for each computing discipline. The main motivation for the filter approach is rooted in factors that leave faculty with what they feel is insufficient time to provide students with necessary preparation for their chosen field. Filter proponents generally report that their program must establish a strong disciplinary foundation early and cannot afford a common introductory course for all computing disciplines. Alternatively, the school/college context (such as traditional engineering for CE and business for IS) may dictate a curriculum framework that leaves little or no room for a common computing introduction. In addition, some proponents of the filter approach argue for parallel curricula on the grounds that students can be best challenged if they are segmented as soon as possible. According to this view, not only do you want parallel introductory sequences, one for each audience, but you also want the early courses to serve as filters that keep students from following a course of study for which they do not initially show high aptitude. Thus, we find different opinions among filter proponents: some think the filter approach is unfortunate but necessary, while others believe it is preferable on its merits. Regardless of motivation and rationale, filter proponents argue that students can and must make an appropriate choice of degree program at a very early stage of their undergraduate career, often *before* they have any coursework or other experience related to the discipline they are choosing.

  For filter curricula to serve students responsibly, degree program must provide students with some reasonable and substantive support for making an informed choice about the degree program, and this must be done before the student enters the program. It is neither reasonable nor responsible to expect young people to make important, life-shaping decisions based primarily on the names of degree programs and the names of introductory courses. While active, effective means of student advising is always valuable, it is especially important for a curriculum that features a filter approach.

  Proponents of the *funnel approach* argue for a common introductory sequence on the grounds that most students cannot be expected to have clarity about their choice of major as a freshman and, furthermore, that the best way for students to obtain clarity is via course experiences designed to give them a feel for the computing disciplines. According to this approach, not only do you want an integrated introductory sequence, you need one if you want students to make well informed choices among degree programs. In the U.K., a shared first year of introductory computing courses is very common; experience there has shown that delaying the decision about choice of major can be beneficial in helping with retention.

  Another motivation for having shared courses is the high cost of maintaining distinct sets of courses for each degree program. In addition to a reduction in course maintenance costs, reducing the number of courses that must be maintained makes the challenge of keeping each course optimized more tractable.

  While all funnel approaches brings a broad population of students through an introductory experience, funnel designs can differ in what happens after that. One design may have students choose among several computing degree programs. Another design might give students only the choice: continuing in a single computing discipline (often *computer science* in the U.S.) or ceasing computing study. In either case, the introductory experience helps students make well informed decisions; what varies is the number of computing degree programs about which they can make an informed decision.

The funnel approach can present some important challenges, depending on its scope. A funnel approach is easier to design if it is targeted at only a single discipline, e.g., a CS program that uses its introductory courses to serve of broad campus-wide audience. In this scenario, undecided students can get a rigorous introduction to programming fundamentals while simultaneously getting a feel for whether they wish to pursue further CS study. In this case, the funnel courses are targeted to serve a single degree program, so it is easier to know how to best focus the material.

Curriculum integration across multiple computing degree programs presents faculty with a more challenging design problem. The goal is to give students direct experience with the kind of work that is featured in advanced courses from various computing disciplines. This can give students the best possible basis for decision from among available computing degree programs, but it requires that those who devise and implement such courses "think outside the box" and be sensitive to the needs of each of the computing disciplines, especially those other than their own. The worst possible scenario is that one computing discipline will hijack the introductory sequence to serve its students, in effect creating a filter that directs those who don't succeed in that "preferred discipline" to one of the other disciplines.

We know of no curriculum model that provides a successful funnel approach that serves all five computing disciplines. However, faculty at various schools, including many in the U.K., report success with funnel courses that serve two or three disciplines. The absence of solid models that serve all five disciplines is likely a reflection of two factors: the inherent difficulty implied in the task, and the simple fact that few schools have much experience as yet in providing degrees in all five disciplines. We expect that future generations of the various discipline-specific curriculum guidelines will investigate and assess introductory models that have demonstrated success at serving a broad funnel agenda.

- *Granularity of curriculum components:* While many of the topics and skills relevant across the various kinds of degree programs are found in the first year or two of study, commonalities extend throughout the entire degree program. For example, programs in each of the computing disciplines need to have advanced courses about each of operating systems, networks, databases, and other areas. For each such area, some material is appropriate to students of all the disciplines, while other coverage requires a discipline-specific treatment. While the commonalities are numerous, they cluster in chunks that are considerably smaller than a traditional semester course and, as a result, fall through the cracks of traditional course designs. This can cause each degree program to have its own custom course despite the fact that its course has much in common with similar courses for other computing majors.

  Reason suggests that there may be great opportunities in devising courses of shorter duration that can be combined as appropriate for the different disciplines. One can imagine a handful of short (perhaps of a few weeks duration) modular courses on various aspects of networking such that, for example, CS students might have some modules in common with IT students, with other modules focusing explicitly on what is unique to the CS or IT agendas. This approach offers obvious potential benefits but also presents various local logistical challenges. While this is seemingly a sound idea, considerable experience is required before we can judge its utility.

Because the diversity of computing programs is new, we have little collective experience in sorting out these issues. Hence, there is great need for innovation and experimentation. Many who favor the funnel approach to curricula feel that the most pressing need is for a new model of introductory courses that can differentially direct students to the right degree program. The challenge is to develop a sequence that is neither preferential nor pejorative to any discipline but instead gives students knowledge and experience that foreshadow what they will see-and-do in each computing degree program. Developing such a sequence seems an important challenge, one that is likely to involve new insights into what is common throughout computing in addition to programming. To date, we do not yet have a compelling model for how to achieve this successfully. We hope to see educators make progress in this area over the next few years, as successful models will provide compelling benefits to students and institutions alike.

4.4. Academic integrity and market forces

Market forces impact academic programs in various ways, some of which are beyond the scope of this report. For example, in recent years various forms of certification have become popular. The term "certification" applies to a wide range of offerings which vary in important ways. Some certifications are vendor-specific (e.g., those from Microsoft, Cisco, etc.). Other certifications are available through professional organizations (e.g., IEEE-CS, BCS) and other organizations (e.g., ICCP). In some of its forms, certification competes with academic programs. It is clear that certification is a major trend. As with anything else, some certifications are more respected, others are more controversial, etc. When degree-granting institutions partner with vendor-specific certification programs, academic integrity becomes an issue. The reader should be aware that such partnerships invite controversy about academic integrity and ethics. It is beyond the scope of this report to address issues related to the broad range of certifications. As certification is addressed by other project reports, we will reference such work in updated versions of this report. Our focus is limited to undergraduate computing degree programs.

The fact that we see new kinds of computing degree programs emerging to address societal needs shows that the dynamism of our society is not lost in academia. The newer computing disciplines provide proof-by-example that academia can and will evolve and be responsive to societal needs. Sadly, that same emergence of new degree programs also provides examples of what can go wrong when academic initiatives are driven by political agendas, fiscal imperatives, and media hype rather than by recognition that substantive innovation is needed to address important concerns.

Fortunately, it is easy to distinguish between these two kinds of phenomena. When we look at the emergence of new computing degree programs, we see examples of both high- and low-quality programs:

- When we look at high-quality programs, we see coherent programs that are driven and developed from within. Faculty and local administrators contribute because they have looked beyond the boundaries of conventional subject-matter areas, recognized that their students and their community need something new and different, and innovated to solve what they see as a legitimate, substantive problem. Faculty in such programs tend to identify themselves as faculty of the new discipline, despite the fact that their background is invariably in another discipline. They value their students, see student and community needs as legitimate, and strive to hold students to high standards appropriate to the discipline.
- When we look at low-quality programs, we see programs that are driven from without. One scenario involves a top-down process wherein someone in power decrees that new programs will be created, perhaps to fit an arbitrary timeline. Faculty and administrators contribute because they are told to do so. They do not see intrinsic positive value in the initiative; they do not see it addressing legitimate needs of students or community. Hence, they "innovate" to provide the superficial appearance of innovation, often by creating "new programs" that are nothing but collections of existing courses from other departments. Faculty involved in such programs tend to identify themselves as faculty in the older, more established discipline from whence they came.

An important lesson here is that good programs do not emerge on the basis of top-down directives alone. While a top-down directive may be a necessary catalyst, good programs require care and nurture from faculty who *can* and *do* see opportunities to do work that has important potential and inherent legitimacy. While fiscal pressures can make "creating programs to meet demand" an attractive option, academic integrity requires more than just a superficial response to market forces. *It is critical that creation of new computing programs is treated as a substantive development effort, undertaken by people who care and who are supported with resources sufficient to permit the development of substantive, coherent programs.*

4.5. Accreditation and computing curricula

Academic accreditation is a process that is used to support improvement of institutions and their degree programs, to demonstrate that a degree program meets certain external requirements, and to increase the level of confidence the public has in them.

In some countries, accreditation can occur at different levels of an academic institution. In those cases, institution-wide accreditations certify that a university meets minimum standards for resources (e.g., library) and operating procedures (e.g., admissions policies) required of any legitimate institution of higher learning. Similar guidelines may exist for an entity within the institution (e.g. a business school) that encompasses degree programs in related fields. Accreditation for a school that houses a group of programs is similar to institutional accreditation but with greater specificity.

The most stringent form of accreditation concerns the evaluation of individual degree programs. This involves the participation of independent organizations or government agencies that establish quality standards and criteria for degree programs in a specific discipline. Discipline-specific (or program) accreditation involves an evaluation of specific degree programs and certifies that a degree program meets established criteria and has rigorous processes for ongoing improvement. Accreditation does not exist for every discipline, but it does exist for computing degree programs.

In nations where accreditation can occur at different levels, an institution may be accredited by an organization that accredits colleges and universities, while its computing degree programs may not be accredited by the body that evaluates the quality of computing degree programs. For example, a U.S. university may have unaccredited degree programs even though the university as a whole is accredited. The distinction to keep in mind is that the accreditation of a college or university *does not imply* that its computing degree programs meet the standards of quality established for the computing disciplines.

4.5.1. Benefits of discipline-specific accreditation

Discipline-specific accreditation provides two important benefits for programs and for the institutions in which they reside.

1) It certifies that a degree program meets minimum quality standards as established by independent professional or scientific societies or by government agencies. This helps an institution market its programs and it gives the public and prospective students reason to be confident in a particular degree program's quality.
2) The program receives an onsite consultation by a visiting team that provides expert opinion about a program's strengths and weaknesses, and about its specific needs for improvement. This interaction helps an institution have full understanding of how its programs are performing and what must be done to improve their quality.

Thus, accreditation provides the benefits of both a marketing aid for attracting students and an expert consultation focused on improving quality. Some institutions may not need or desire the former benefit. Of these, some are committed to accreditation solely because the accreditation process helps them maintain and improve the quality of their programs, which in turn further cements their reputation. In some nations, institutions have no choice because accreditation is a requirement for program existence.

Discipline-specific accreditation processes determine whether a candidate degree program meets certain criteria. Not only does accreditation determine whether the program provides sufficient qualified teachers with acceptable workloads, it also determines how the program uses materials and assignments, evaluates assignments and examinations, and engages itself in continuous evaluation and improvement.

Professional bodies also use this accreditation to ensure that degree programs meet, at least in part, the requirements for membership of their profession. In some cases, graduation from an accredited degree program is a requirement of individuals before they can practice in a particular profession. This means

that it is not sufficient for a student who wishes to practice a profession simply to earn a degree in the appropriate discipline; rather, he or she must have earned that degree from an accredited degree program.

A given degree program does not choose whether its accreditation has such "professional" elements; its accreditation process is determined by what is customary for its discipline in its nation. For example, in the U.S. accreditation for engineering programs includes professional aspects, those for other kinds of degree programs often do not, and most arts and sciences disciplines do not have accreditation at all.

Perhaps the greatest misconception about accreditation is the belief that institutions pursue program accreditation only to obtain a "stamp of approval". Those unfamiliar with discipline-specific accreditation often do not understand the important role that the accreditation process plays in helping a program know exactly what it must do to improve the quality of both its offerings and its graduates.

4.5.2. Accreditation and quality

Discipline-specific accreditation is a means of demonstrating that a degree program meets an independent standard of quality, but the meaning of that standard varies. Its rigor is determined by the accrediting body's policies and practices, and by any government regulations that may apply. In some cases, accreditation certifies that a degree program has met a minimum quality standard. In other cases, there exist both minimum standards and higher standards.

While discipline-specific accreditation is concerned with program quality, it is important not to reach unwarranted conclusions about the relationship between accreditation and quality. One must be familiar with both the discipline and the national context in order to reach appropriate conclusions.

Does the absence of accreditation indicate that a program is of low quality?

The answer depends on the discipline and on the national context. For example, virtually every U.K. computing program is accredited (see section 4.5.4). The situation in the U.S. is much more varied: while every reputable CE program is accredited, most CS programs are not accredited, and accreditation of IS, IT, and SE programs is only now beginning (see section 4.5.5).

Does the fact that a program is accredited indicate that it is of high quality?

The answer depends on national context. In some nations, accreditation indicates only that a program has met a standard of minimum acceptable quality; in others, accreditation recognizes different quality levels.

Do all accreditation processes involve the same procedures?

No. There are important differences in quality assurance practices between nations. For example, in the U.S., an accreditation team must review and assess a program at all its levels, from specific class assignments to high-level programmatic issues, as part of the accreditation visit. In contrast, U.K. quality assurance practices include the involvement of external examiners as part of daily operation. In the British system, all class assignments and examinations, and their evaluation criteria, are reviewed and evaluated in advance by faculty from other institutions. This means that students in British classes never receive an assignment or an examination that was prepared spontaneously or that lacks appropriate evaluation criteria. If an assignment or its evaluation criteria does not satisfy the external examiner, it must be improved until it does. While these quality assurance practices are unrelated to accreditation, the written evaluations produced by external examiners are available to the accreditation team. This provides the accreditation team with evaluation data obtained over time, which frees the accreditation team from studying low-level details and permits them to focus on quality issues at a higher programmatic level.

Apart from accreditation, what are some indications of high- and low- program quality?

There are several aspects of high quality: good teachers, a workload that permits teachers to adequately focus on their classes and remain current in their field, sufficient infrastructure, and so on. One of the

most important things to look for is evidence of rigorous procedures for monitoring and improving quality in an ongoing way. It is important that adequate processes be in place to recognize where improvements are needed, and that information about quality is actually used to produce demonstrable improvement.

In a good program, quality-monitoring processes are integrated with initiatives for improving quality to form a continuous cycle: each improvement effort is monitored for effect, new improvement efforts are then planned and implemented, the results evaluated, and the cycle repeats. Doing this properly is not difficult, but a measure of commitment, discipline, and information-sharing and -use are required.

It is difficult for someone who is not privy to the internal operations of a degree program to know what actually occurs. In terms of more accessible features, one should inquire about items such as:

- A professional advisory board: Is there one? Who is on it? Does it focus on educational aspects of the program, or is it concerned only with a department's research program?
- The use of information obtained from students: Does the program do more than collect data from end-of-term student satisfaction surveys? Are students involved in various academic committees? Does the program use exit interviews to acquire the opinions of graduating students about program's strengths and weaknesses? Are graduates of the program consulted after they work in industry for a few years? Does evidence exist that such information is used in ways that leads to demonstrable improvement?
- The processes for systemic evaluation and improvement: Are procedures in place to evaluate the effectiveness of each course? What methods are deployed to assess the strengths and weaknesses of a program's graduating students? What processes ensure that such information is actually utilized?

Meaningful quality improvement requires more than simply calculating student grade point averages and collecting data from end-of-term student satisfaction surveys. If a program has little to point to beyond collection of these basic data, it is reasonable to have doubts about whether there is an adequate focus on systematically improving the quality of both the degree program itself, and of the graduates it produces.

4.5.3. National traits and international cooperation

Many countries have embraced accreditation. The details vary but there is a common thread: a panel of experts who represent a profession evaluates a program's quality against established standards and criteria.

Nations vary with respect to whether accreditation is mandatory, strongly encouraged, or voluntary. Some countries have rigorous program criteria and require that accreditation standards apply to every program offered at any college or university. In other countries, accreditation is voluntary. Among countries in which accreditation is voluntary, the extent to which accreditation is expected differs.

The administration of the accreditation process also varies. In some countries (e.g., Australia, Canada, and the U.K.) professional societies conduct program accreditation for their respective fields. In other countries (e.g., the U.S.) a designated organization monitors and/or performs accreditation. In some countries (e.g. Estonia and the United Arab Emirates), the government conducts the accreditation process.

In some computing disciplines, accreditation agencies also cooperate across national borders. For example, in addition to its accreditation activities in the U.S., ABET has assisted other nations in the evaluation of their programs for more than 20 years. Mutual recognition of evaluation and accreditation processes has encouraged a range of international agreements such as the Washington Accord, the Sydney Accord, the Dublin Accord, the European Federation of National Engineering Associations (FEANI), and the International Register of Professional Engineers (IRPE). Such agreements have a range of signatories but they share a common goal: to facilitate the movement of professionals across nations. For example, the Washington Accord is an agreement among the organizations that accredit engineering degree

programs in Australia, Canada, Hong Kong, Ireland, New Zealand, South Africa, U.K., and U.S.; it recognizes the substantial equivalence of programs accredited by those bodies.

4.5.4. Computing accreditation in the United Kingdom

The U.K. Engineering Council (ECUK) is responsible for accreditation of engineering degree programs. Its responsibilities include setting standards (of competence and commitment) for the accreditation of engineering degrees and approving "nominating bodies" that carry out detailed accreditation on its behalf. The British Computer Society (BCS) carries out accreditation of Information Systems Engineering degree programs, which encompasses most of what we understand by computing, on behalf of the Engineering Council. The Institute of Electrical Engineers (IEE) carries out the accreditation of electrical engineering degree programs. Degree programs in computer engineering could be accredited by either society, though perhaps more often by IEE. However, joint accreditation by both societies is common.

In addition, the U.K. government has instituted its own quality assurance procedures for all disciplines, using the Quality Assurance Agency (QAA). The QAA is responsible for benchmarking degrees. Each institution is required to demonstrate that their degrees meet the benchmark standards for that discipline. One example of these benchmark standards is [UKQAA, 2000] which defines both threshold (minimal) and modal (average) expectations with respect to demonstrated student knowledge, skills, and judgment.

4.5.5. Computing accreditation in the United States

Until recently, computing accreditation in the U.S. was limited and fragmented. It is now more united and more extensive. Historically, two organizations accredited computing degree programs. The Accreditation Board for Engineering and Technology (commonly know as "ABET") accredited undergraduate engineering programs of all kinds, including *electrical engineering* and *computer engineering*. The Computing Sciences Accreditation Board (known as "CSAB") accredited undergraduate *computer science* programs. Recently, CSAB, recreated as "CSAB Inc.", has become a member society of a renamed "ABET Inc.", and now ABET accredits engineering, technology, computing, and applied science programs. Currently, accreditation of CE, CS, IS, and SE programs is ongoing, and accreditation of IT programs has begun.

Accreditation in the U.S. is voluntary in the sense that no law or regulation requires a degree program to be accredited. As a practical matter, it is "more voluntary" in some computing disciplines than in others. In engineering for example, a strong sense of a professional community exists and state-regulated licensure of engineers can require applicants to hold an engineering degree from an ABET accredited program. As a result, we expect accreditation status for all CE programs, and any CE programs that are not accredited are unlikely to be credible. The situation is less clear for SE programs because SE is new and, by its nature, has far fewer ties to traditional engineering.

In contrast, the CS community is more of a loosely organized network of scientists, researchers, and programmers than a tightly organized body of practicing professionals. Historically, there has been no compelling professional pressure for accreditation of CS programs; demand comes from universities seeking it. Currently, less than 10% of U.S. CS programs are accredited, where nearly 100% of U.K. computing programs are accredited. Recently, however, demand for CS accreditation has increased. The IS discipline is often associated with schools of business that have their own tradition of accreditation. The IT discipline is new, and because most IT professionals are primarily concerned with technical competence, it would not be surprising to see IT programs seeking accreditation.

The philosophy of ABET accreditation has recently shifted from prescriptive criteria to outcomes-based approaches. The new approach requires each program to define its objectives and demonstrate that its processes ensure that it is achieving those objectives.

Chapter 5: Next steps

This list summarizes curriculum-related developments that are planned for the near future.

- Publication of the IT2006 report.

  The current draft is available for review and comment at http://www.acm.org/education/curricula.html. Publication of the final version is expected in 2006.

- Publication of *The Guide to Undergraduate Degree Programs in Computing*.

  The *Guide* will be a smaller companion report to the *Overview Report*. It will be aimed at a broader audience, including prospective students, their parents and guidance counselors, and others who have reason to care about the choices that await students who move from high school to college. It will provide brief characterizations of the computing disciplines, profile factors that prospective students may consider when choosing an area of computing study, and will be widely distributed as an independent document. The Joint Task Force plans to present drafts for public review and comment early in 2006 and to have a final version ready for publication by mid-2006.

- Implementation of new timetables for revision to each volume in the *Computing Curricula Series*.

  In the past, not only were curriculum reports limited to a much smaller range of computing disciplines, those guidelines were revised only about once every 10-to-12 years. Given the rapid pace of change in computing, more frequent revisions are required. The long term goal is the development of processes that will permit us to issue revisions as needed, driven by evolution in subject matter.

  While we develop appropriate processes (see below), as an initial approximation of an appropriate schedule we intend to cut the previous cycle in half, with revisions to each of the discipline-specific volumes appearing every 5-to-6 years instead of 10-to-12. Thus, a task force will be formed in mid-2005 to update the *computer science* volume (generating what will become *CS2007*), followed by a task force to update *IS2002* (generating *IS2007* or *IS2008*). Whenever an updated discipline-specific volume is published, an updated version of this *Overview* volume will follow.

- Initiation of new processes for capturing feedback in an ongoing way about each volume in the *Computing Curricula Series* (including the *CE, CS, IS, IT, SE,* and *Overview* volumes).

  In the past, opportunities for members of the computing community to provide input and feedback re: curriculum recommendations were limited. Beginning with the *SE2004* volume, and continuing with this volume and the *IT2006* project, we have begun to provide opportunities for anyone, anywhere to submit suggestions and critique via online tools. We intend to refine the tools, improve their visibility to the larger computing community, and make them consistently available. The goal is to permit motivated persons to provide the benefit of their experience and opinion throughout the life of each report, rather than only during an explicit report development effort.

- Fueled in part by such feedback, new processes for ongoing evaluation of the adequacy of each volume in the *Computing Curricula Series,* including processes for specifying needed changes to each volume.

  It is clear that curriculum reports must be updated more frequently than once per decade, but we do not want to issue updated volumes in an arbitrary fashion. New volumes call for change at the local level, and each set of changes imply costs. In an ideal world, new volumes would be issued in response to some critical mass of change and evolution in the field, driven by need rather than by a schedule. For this to occur, we require processes that permit us to monitor evolution in the field, identify specific needs for curriculum revision, and issue revisions as needed. The societies that cooperated on the creation of these volumes are in the process of creating a task force to address this important issue.

- Improved support for frequent updating of curricula.

  Not only must we develop capabilities for updating each volume as needed, we also require effective ways of looking across the various volumes. Our current practices imply relatively independent efforts for each discipline-specific volume, which is very labor intensive. At the same time, changes in computing often apply across disciplinary boundaries. The *Computing Ontology Project* is currently developing a framework for modeling all computing subject matter across the computing disciplines. The goal is to produce both schema and tools that will permit us to see the superset of computing's problem space, update the particulars in that shared space in response to ongoing developments in computing, identify which disciplines are affected by those developments, and trigger responsive attention to curricular issues. In addition, having appropriate schema and tools will permit us to evaluate and characterize emerging new computing disciplines and better leverage existing relationships and infrastructure in their development. The *Computing Ontology Project* is well underway. You may hear of its progress via publications and conference presentations in the future.

In the meantime, you may access the set of current volumes in the *Computing Curricula Series* at http://www.acm.org/education/curricula.html and http://computer.org/curriculum. At those sites, you will also find available news about various existing and upcoming projects, links to feedback opportunities that are available, and news about curriculum-related presentations.

References

[ANSI/IEEE Standard 729-1983] IEEE Standard Glossary of Software Engineering Terminology, The Institute of Electrical and Electronics Engineers, Inc., NY, 1983.

[AssocDeg] ACM Two-Year College Computing Curricula Task Force. Computing Curricula Guidelines for Associate-Degree Programs. New York, NY: ACM Press, 1993. (http://www.acm.org/education/curricula.html)

[CC91] ACM/IEEE-CS Joint Curriculum Task Force. Computing Curricula 1991. Association for Computing Machinery and the Computer Society of the Institute of Electrical and Electronics Engineers, 1991. (http://www.acm.org/education/curricula.html or http://www.computer.org/curriculum)

[CC2001] ACM/IEEE-Curriculum 2001 Task Force. Computing Curricula 2001, Computer Science. IEEE Computer Society Press and ACM Press, December 2001. (http://www.computer.org/curriculum or http://www.acm.org/education/curricula.html)

[CE2004] IEEE/ACM Joint Task Force on Computing Curricula. Computer Engineering 2004. Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering. IEEE Computer Society Press and ACM Press, December 2004. (http://www.computer.org/curriculum or http://www.acm.org/education/curricula.html)

[HS] Pre-College Task Force Committee of the Education Board of the ACM. ACM Model High School Computer Science Curriculum. Communications of the ACM, 36(5): 87-90, May1993. (http://www.acm.org/education/curricula.html)

[IEEE Std 610.12-1990] Standard Glossary of Software Engineering Terminology. The Institute of Electrical and Electronics Engineers, Inc., NY, 1990.

[IS97] IS '97 Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems. Association for Computing Machinery and Association for Information Technology Professionals, 1997. (http://www.acm.org/education/curricula.html)

[IS2002] ACM/AIS/AITP Joint Task Force on Information Systems Curricula. IS2002 Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems, Association for Computing Machinery, Association for Information Systems, and Association for Information Technology Professionals, 2002. (http://www.acm.org/education/curricula.html or http://www.computer.org/curriculum)

[IT2006] The ACM SIGITE Task Force on IT Curriculum. Information Technology 2006, Curriculum Guidelines for Undergraduate Degree Programs in Information Technology. As this is written, the IT curriculum report is in development. We expect the review and development process to be completed in 2006. At the moment, the current draft is called *IT2005* and is available for public review and comment. The most current link to the IT volume can be found at: http://www.acm.org/education/curricula.html.

[K-12] ACM K-12 Task Force Curriculum Committee. A Model Curriculum for K-12 Computer Science. ACM Press, 2004. (http://csta.acm.org/Curriculum/sub/k12final1022.pdf)

[SE2004] IEEE/ACM Joint Task Force on Computing Curricula. Software Engineering 2004, Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering, IEEE Computer Society Press and ACM Press, August 2004. (http://www.computer.org/curriculum or http://www.acm.org/education/curricula.html)

[UKQAA, 2000] Quality Assurance Agency for Higher Education Computing. A Report on Benchmark Levels for Computing. Southgate House, Gloucester, England, April 2000. (http://www.qaa.ac.uk/academicinfrastructure/benchmark/honours/computing.asp)

Glossary

Algorithms and Complexity – Computational solutions (algorithms) to problems; time and space complexity with respect to the relationship between the run time and input and the relationship between memory usage and input as the size of the input grows.

Analysis of Business Requirements – The process through which an information systems or software application development project determines the optimal capabilities of the target system or application based on the business goals of the individual user(s) or the user organization(s).

Analysis of Technical Requirements – The process through which a computing development project determines the computing and communications hardware and software based on the goals of the individual user(s) or the user organization(s).

Business Models – Various structures, processes, and other mechanisms that businesses and other organizations use for organizing the way they interact with their primary external stakeholders (e.g., customers and suppliers) to achieve their primary goal (e.g., maximization of profit).

Circuits and Systems – The computing and communications hardware and software components that constitute a computing project or solution.

Computer Architecture and Organization – Form, function, and internal organization of the integrated components of digital computers (including processors, registers, memory, and input/output devices) and their associated assembly language instructions sets.

Computer Systems Engineering – A computing discipline that is more prominent in Europe than in North America. It integrates aspects of CE, CS, and SE, and focuses on the development of complex systems that require close integration of computer hardware and software. Areas of special emphasis include design and implementation of embedded and real-time systems, the use of formal methods for specification of computer systems, and the implementation of systems on specialized-purpose circuits.

Decision Theory – A field of study that develops knowledge and analytical models that together will help decision makers select among various alternatives that are known (or thought) to lead to specific consequences.

Digital logic – Sequential and non-sequential logic as applied to computer hardware including circuits and basic computer organization.

Digital Media Development – The field of computing that deals with the portable storage of digital information.

Digital Signal Processing – The field of computing that deals with digital filters, time and frequency transforms, and other digital methods of handling analog signals.

Distributed Systems – Theory and application of multiple, independent, and cooperating computer systems.

E-business – The use of information and communication technology solutions to implement business models and internal and external business processes. In a more narrow sense the term is often used to refer to the use of Internet technologies to conduct business between firms (B2B), between firms and consumers (B2C), or among consumers (C2C).

Electronics – The hardware that constitutes the computing and communications circuits which either directly operate on electronic signals, or run the software which operates on electronic signals. The fields of computing and communications presently rely completely on electronics.

Embedded Systems - Hardware and software which forms a component of some larger system and which may be expected to function with minimal human intervention (e.g., an automobile's cruise control system)

Engineering Economics for SW – Cost models for the software engineering life cycle including development, maintenance, and retirement of software systems.

Engineering Foundations for SW – Engineering design, process, and measurement as applied to software systems.

Evaluation of Business Performance – The activities that an organization uses to determine how successful it has been in achieving its goals.

Functional Business Areas – Accounting, finance, marketing, human resource management, manufacturing, and logistics are examples of functional business areas. Each of these is responsible for a set of connected business activities, which as a whole help a business achieve a specific functional goal (such as providing a reliable and appropriate set of internal and external business performance measures in accounting).

General Systems Theory – A field of study that explores the general characteristics of systems in various areas of human behavior and natural sciences with a special focus on complexity and system component interdependency. General systems theory had its origins in physics, biology, and engineering, but it has been utilized in many other fields, such as economics, organizational theory, philosophy, sociology, and information systems.

Graphics and Visualization – Theory and application of computer generated graphics and graphical representation of data and information including static, dynamic, and animated techniques.

Hardware Testing and Fault Tolerance – The field of study that deals with faster, cheaper, and more efficient ways of testing hardware (see also **Electronics** and **Circuits and Systems**), as well as ways of making hardware more fault tolerant (able to continue functioning as specified in spite of hardware or software faults).

Human-Computer Interaction – An organizational practice and academic field of study that focuses on the processes, methods, and tools that are used for designing and implementing the interaction between information technology solutions and their users.

Information Management (DB) Theory – Theoretical models for information representation, storage, and processing.

Information Management (DB) Practice – The activities associated with the analysis, design, implementation, and management of organizational information resources, such as operational databases, data warehouses, and knowledge management systems.

Information Systems Development – The human activities -- including requirements analysis, logical and physical design and system implementation -- that together lead to the creation of new information systems solutions.

Integrative Programming –Uses the fundamentals of programming to focus on bringing together disparate hardware and software systems, building a system with them that smoothly accomplishes more than the separate systems can accomplish.

Intelligent Systems (AI) – Computer applications based on artificial intelligence theory and techniques including rule-based systems, genetic and evolutionary computation, and self-organizing systems.

Interpersonal Communication – An area of study that helps computing students improve their oral and written communication skills for teamwork, presentations, interaction with clients and other informants, documentation, sales and marketing activities, etc.

Legal / Professional / Ethics / Society – The areas of practice and study within the computing disciplines that help computing professionals make ethically informed decisions that are within the boundaries of relevant legal systems and professional codes of conduct.

Management of Information Systems Organization – The processes and structures that are used to organize and manage the employees and contractors within the organization whose primary organizational role is to create, maintain, administer, or manage organizational information systems solutions.

Mathematical Foundations – Those aspects of mathematics that underlie work in the computing disciplines. The subsets of mathematics that are most relevant to computing vary from one computing discipline to another. Depending on the discipline, mathematical foundations may include algebra (linear and abstract), calculus, combinatorics, probability, and/or statistics. The term "mathematical foundations" sometimes also includes the fields of study and research that are interdisciplinary between mathematics and computer science, such as discrete mathematics, graph theory, and computational complexity theory.

Net Centric: Principles and Design – Includes a range of topics including computer communication network concepts and protocols, multimedia systems, Web standards and technologies, network security, wireless and mobile computing, and distributed systems.

Net Centric: Use and Configuration – The organizational activities associated with the selection, procurement, implementation, configuration, and management of networking technologies.

Operating Systems Principles & Design – Underlying principles and design for the system software that manages all hardware resources (including the processor, memory, external storage, and input/output devices) and provides the interface between application software and the bare machine.

Operating Systems Configuration & Use – Installation, configuration, and management of the operating system on one or more computers.

Organizational Behavior – A field of study within the business discipline of management that focuses on individual and group level human behavior in organizations. The core topics include, for example, individual and group decision making, problem solving, training, incentive structures, and goal setting.

Organizational Change Management – A field of study often associated with the business discipline of management that focuses on topics that help employees in organizations to manage and cope with organizational change, whether it is a result of internal organizational actions or forces in the external environment.

Organizational Theory – A field of study within the business discipline of management that focuses on the structure of the organizations. This field helps managers decide what types of organizational

structures to use and understand why certain types of structures tend to work better than others. Key questions focus on centralization/decentralization of power, the selection and use of coordination and control mechanisms, and breadth and dept of the organizational reporting structures.

Platform Technologies – The field of study which deals with the computing hardware and operating systems which underlie all application programs.

Programming Fundamentals - Fundamental concepts of procedural programming (including data types, control structures, functions, arrays, files, and the mechanics of running, testing, and debugging) and object-oriented programming (including objects, classes, inheritance and polymorphism).

Project Management – An organizational practice and academic field of study that focuses on the management approaches, organizational structures and processes, and tools and technologies that together lead to the best possible outcomes in work that has been organized as a project.

Risk Management (Project, safety risk) – An organizational practice and academic field of study that focuses on the processes, management approaches and technologies for identifying risks, determining their severity level, and choosing and implementing the proper course of action for each risk.

Scientific Computing (Numerical Methods) – Algorithms and the associated methods for computing discrete approximations used to solving problems involving continuous mathematics.

Security: Issues and Principles – Theory and application of access control to computer systems and the information contained therein.

Security: Implementation and Management – The organizational activities associated with the selection, procurement, implementation, configuration, and management of security processes and technologies for IT infrastructure and applications.

Software Design - An activity that translates the requirements model into a more detailed model that represents a software solution which typically includes architectural design specifications and detailed design specifications. [Alternatively: In software engineering, the process of defining the software architecture (structure), components, modules, interfaces, test approach, and data for a software system to satisfy specified requirements. [ANSI/IEEE Standard 729-1983]]

Software Evolution (Maintenance) - (1) The process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment. (2) The process of retaining a hardware system or component in, or restoring it to, a state in which it can perform its required functions. [IEEE Std 610.12-1990]

Software Modeling and Analysis – An activity that attempts to model customer requirements and constraints with the objective of understanding what the customer actually needs and thus defining the actual problem to be solved with software.

Software Process - (1) A sequence of steps performed for a given purpose; for example, the software development process. (2) An executable unit managed by an operating system scheduler. (3) To perform operations on data. [IEEE Std 610.12-1990]

Software Quality (Analysis) - (1) A planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements. (2) A set of activities designed to evaluate the process by which products are developed or manufactured. [IEEE Std 610.12-1990]

Software Verification and Validation - The process of determining whether the requirements for a system or component are complete and correct, the products of each development phase fulfill the requirements or conditions imposed by the previous phase, and the final system or component complies with specified requirements. [IEEE Std 610.12-1990]

Systems Administration – The field of study which deals with the management of computing and communications resources, including networks, databases, operating systems, applications, and Web delivery. The management issues include installation, configuration, operation and maintenance.

Systems Integration – The field of study which deals with the incorporation of computing and communications resources to create systems that meet specific needs. Elements include organizational issues, requirements, system architecture, acquisition issues, testing, and quality assurance.

Technical Support – The field of study which deals with solving the problems of the end user of a computing and/or communications product or system, after the product or system has been delivered and installed.

Theory of Programming Languages – Principles and design of programming languages including grammars (syntax), semantics, type systems, and various language models (e.g., declarative, functional, procedural, and object-oriented).

VLSI Design – The field of study which deals with creating **electronics** solutions to computing and communications problems or needs. This includes custom integrated circuit (IC) design (which includes microprocessors and microcontrollers), application-specific IC design (including standard cells and gate arrays), and programmable hardware (including FPGAs, PGAs, PALs, GALs, etc.).