The Development of the IEEE/ACM Software Engineering Curricula

1.0 Introduction

The recent surge in the creation of software engineering programs and their accreditation has resulted in the development of the Joint IEEE/ACM Computing Curricula - Software Engineering (IEEE/ACM CCSE) [13]. The primary purpose of IEEE/ACM CCSE is to provide guidance to academic institutions and accreditation agencies about what should constitute an undergraduate software engineering curriculum and its implementation.

A curriculum is a well defined process by which the knowledge and skills required for a discipline or a profession can be defined and taught by a number of coherent courses according to a specific degree program.

The IEEE/ACM Joint Task Force on the development of the Software Engineering Curriculum (JTF-SEC) was established in 1998. The definition of software engineering and its implication and extension are discussed in the group.

IEEE defined software engineering as "The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software [12]." The author perceives that:

> "*Software Engineering* is a discipline that adopts engineering approaches, such as established methodologies, processes, architectures, measurement, tools, standards, organization methods, management methods, quality assurance systems and the like, in the development of large-scale software, seeking to result in high productivity, low cost, controllable quality, and measurable development schedule [14, 15]."

The JTF-SEC adopted a set of principles to guide the work on the development of the Software Engineering Education Knowledge (SEEK) and the Computing Curricula - Software Engineering (CCSE). The design philosophy and principles of CCSE are as follows [13]:

- **Basic Knowledge and Skills:** "All software engineering students must learn to integrate theory and practice, to recognize the importance of abstraction and modeling, to be able to acquire special domain knowledge beyond the computing discipline for the purposes of supporting software development in specific domains of applications, and to appreciate the good engineering design."

  "CCSE must support the identification of the fundamental skills and knowledge that all software graduates must possess."

- **Dynamic Curricula:** "The rapid evolution and the professional nature of software engineering require an ongoing review of the corresponding curriculum."

  "Development of a software engineering curriculum must be sensitive to changes in technology, new developments in pedagogy, and the importance of lifelong learning."

- **Professionalism:** "CCSE must include exposure to aspects of professional practice as an integral component of the undergraduate curriculum."

- **Pedagogy and Implementation Support:** "CCSE must include discussions of strategies and tactics for implementation, along with high-level recommendations."

It is recognized that software engineering draws its foundations from a wide variety of disciplines. Although undergraduate study of software engineering relies on many areas in computer science for its theoretical and conceptual foundations, it also requires students to utilize concepts from a variety of other fields, such as mathematics, engineering and project management.

2.0 Sources of IEEE/ACM CCSE

The development of IEEE/ACM Computing Curricula - Software engineering (CCSE) is on the basis of a number of international initiatives and a wide range of educational experiences. The major sources of CCSE are the IEEE/ACM SWEBOK [5], CCCS [9], IS2002 [4], and surveys on international software engineering programs.

by   *Yingxu Wang*
University of Calgary, Calgary, AB.

Abstract

A recent important international effort in software engineering education is the work of the IEEE/ACM Joint Task Force on the development of the Software Engineering Curriculum (JTF-SEC). This paper reports the design and development of the Computing Curricula - Software Engineering (CCSE), and presents the philosophy and insides behind CCSE based on the experience as a member of the committee.

The architecture of CCSE encompasses the guiding principles, curricular models, software engineering education knowledge areas, curriculum design, pedagogy guidelines, professional practice, program implementation and accreditation. The major components of CCSE are the Software Engineering Education Knowledge (SEEK), a formulate guidance for pedagogy, and methodologies for course design and accreditation. Software engineering education in SEEK is categorized into a set of knowledge areas, known as foundations, requirements, design, construction, maintenance, process, quality, and management.

CCSE is featured not only by its knowledge structures, but also by its studies on pedagogy for software engineering. CCSE provide a comprehensive set of principles for curriculum design, methodologies for software engineering course development, and core themes and models of the software engineering curriculum for both directors of software engineering programs and instructors of software engineering courses. It is noteworthy, however, that a number of important areas have not yet been modeled in the CCSE curriculum, such as software engineering notations, measurement and metrics, and theoretical foundations of software engineering.

Sommaire

Un effort international important et récent en enseignement du génie logiciel est celui du groupe de travail conjoint IEEE/ACM sur le développement du Curriculum de génie logiciel (JTF-SEC). Cet article présente la conception et le développement des Curriculums en informatique - génie logiciel (Computing Curricula - Software Engineering, CCSE), ainsi que la philosophie et les dessous du CCSE d'après l'expérience d'un membre du comité. L'architecture du CCSE comprend les principes directeurs, des modèles de curriculums, les domaines de connaissance en enseignement du génie logiciel, la conception de curriculums, les lignes directrices pédagogiques, la pratique professionnelle, les programmes d'implantation et l'accrédiation. Les composantes majeures du CCSE sont les Connaissances en enseignement du génie logiciel (SEEK), une formule d'orientation pédagogique, et des méthodologies pour la conception de cours et l'accrédiation. L'enseignement du génie logiciel dans SEEK est catégorisé en un ensemble de domaines de connaissances: fondations, spécifications, conception, construction, maintenance, processus, qualité et gestion.

Le CCSE se caractérise non seulement par ses structures de connaissances, mais aussi par ses recherches sur la pédagogie pour le génie logiciel. CCSE fournir un ensemble exhaustif de principes pour la conception des curriculums, des méthodologies pour le développement de cours en génie logiciel, et des thèmes et modèles centraux des curriculums de génie logiciel pour les directeurs de programmes de génie logiciel et les enseignants. Il est à noter, cependant, qu'un nombre important de domaines n'ont pas encore été modélisés dans le curriculum du CCSE tels la notation en génie logiciel, les mesures et métriques, et les fondations théoriques du génie logiciel.

2.1 IEEE/ACM SWEBOK

The software engineering body of knowledge (SWEBOK) [5] is developed by an IEEE/ACM joint committee in 2001. SWEBOK provides a comprehensive description of the knowledge needed for the practice of software engineering. The IEEE/ACM JTF-SEC has chosen SWEBOK as one of the primary sources for the development of SEEK

Although CCSE was significantly influenced by SWEBOK, the major difference between SWEBOK and CCSE is that the former is process-centered and the latter is water-fall-model based. In addition, SWEBOK was designed for professionals working in the software industry and CCSE is oriented to undergraduate students in software engineering programs.

2.2 IEEE/ACM CCCS

IEEE-CS and ACM established a task force on Computing Curricula in 1998, that results in the Computing Curricula 2001 (CC2001) [1]. Over the past fifty years, computing has become an extremely broad designation that extends well beyond the boundaries of computer science to encompass such independent disciplines as computer engineering, software engineering, and information systems. In representing this trend, the computing curricula are divided into two volumes known as the Computing Curricula - Computer Science (CCCS) [9] and the Computing Curricula - Software Engineering (CCSE) [13].

2.3 ACM/AIS/AITP IS2002

Related to the international effort on CCCS, the Model Curriculum and Guidelines for Undergraduate Degree Program in Information Systems (IS 2002) is published in 2002 [4], jointly developed by a task force chartered by ACM, the Association for Information Systems (AIS), and the Association of Information Technology Professionals (AITP). IS 2002 is perceived to be useful for both students major in information technology (engineering), and computer engineering [3].

2.4 IEEE/ACM JTF-SEC

The IEEE/ACM Joint Task Force on the development of the Software Engineering Curriculum (JTF-SEC) conducted a survey of international undergraduate programs of software engineering. In this comprehensive survey, 32 programs in North America, Europe, Asia and Australia have been comparatively analyzed. This work forms the empirical foundation and a global view on the design of CCSE and SEEK.

3.0 IEEE/ACM CCSE

The software engineering curriculum is a set of carefully defined programs by which the knowledge and skills required for software engineering as a profession can be defined and taught by a number of coherent courses for a specific degree in software engineering or computer science.

IEEE/ACM Computing Curricula - Software Engineering (CCSE) concentrates on the knowledge and pedagogy associated with a software engineering curriculum. The architecture and the recommended curriculum of CCSE are described below.

3.1 The Architecture of CCSE

The architecture of CCSE is shown in Figure 1. CCSE encompasses (a) The guideline principles and professional practice; (b) Software engineering education knowledge, curriculum models, and curriculum design; and (c) program implementation and assessment, and student outcomes.

As shown in Figure1, the center of CCSE is the Software Engineering Education Knowledge (SEEK) and the recommended curricula on software engineering.

3.2 The Recommended Curriculum

A software engineering program designed based on IEEE/ACM CCSE may be divided into the introductory, core, and completing modules as shown in Table 1, where the recommended courses and their levels in each module are also described.
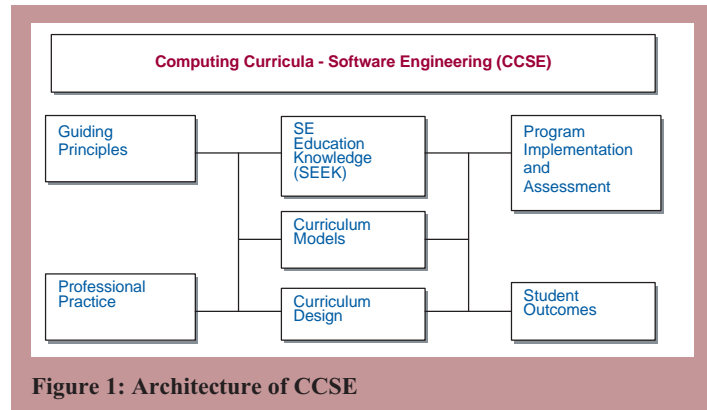

Figure 1: Architecture of CCSE

4.0 IEEE/ACM SEEK

Software Engineering Education Knowledge (SEEK) is a set of core and fundamental knowledge that every software engineering graduate must know [13]. This section describes the architecture of SEEK and the knowledge areas and units modeled in SEEK.

4.1 The Architecture of SEEK

The architecture of SEEK is shown in Figure 2, where 11 knowledge areas have been modeled. Each of these knowledge areas is explained below.

- **A1: Fundamentals**

Fundamentals of software engineering cover the theoretical and mathematical foundations of software and software engineering technologies. This area focuses on engineering design, where mathematics and engineering sciences are applied to optimally convert resources to meet a stated objective [13].

- **A2: Professional Practice**

Professional practice in software engineering is represented by knowledge, skills, attitudes, and professionalism that software engineers must possess. It also includes the areas of technical communication, team working, psychology, and social and professional responsibilities.
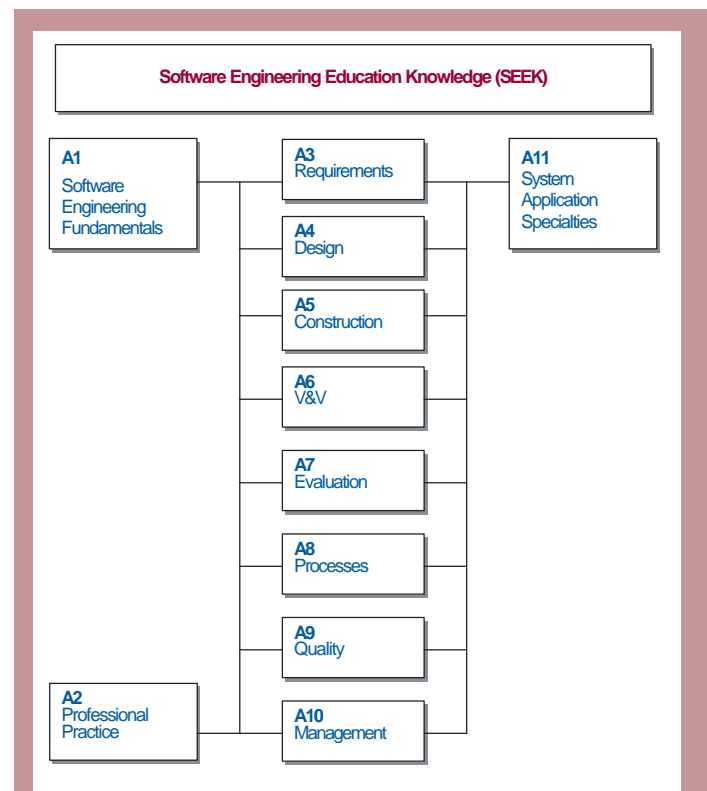

Figure 2: Architecture of SEEK

Table 1: The Recommended Curriculum in CCSE

Cat #	Course Title	Description
1	**Introductory**	
SE101	Introduction to Software Engineering and Computing	A first course in software engineering and computing for the software engineering student who has taken no prior computer science at the university level. Introduces fundamental programming concepts as well as basic concepts of software engineering.
SE102	Software Engineering and Computing II	A second course in software engineering, delving deeper into software engineering concepts, while continuing to introduce computer science fundamentals.
SE200	Software Engineering and Computing III	Continues a broad introduction to software engineering and computing concepts.
CS103	Data Structures and Algorithms	Any variant of CS 103 from the CCCS can be used (e.g., those from the imperativefirst or objects-first sequences).
CS105	Discrete Structures I	Standard first course in discrete mathematics. Taught in a way that shows how the material can be applied to software and hardware design
CS106	Discrete Structures II	Continues the discussion of discrete mathematics introduced in CS105. Topics in the second course include predicate logic, recurrence relations, graphs, trees, matrices, computational complexity, elementary computability, and discrete probability.
MA271	Statistics and Empirical Methods	Applied probability and statistics in the context of computing. Experiment design and the analysis of results.
2	**Core**	
2.1	**Package I**	
SE211	Software Construction	Covers low-level design issues, including formal approaches
SE212	Software Engineering Approach to Human Computer Interaction	Covers a wide variety of topics relating to designing and evaluating user interfaces, as well as some of the psychological background needed to understand people
SE311	Software Design and Architecture	Advanced software design, particularly aspects relating to distributed systems and software architecture
SE321	Software Quality Assurance	Broad coverage of software quality and testing
SE322	Software Requirements Analysis	Broad coverage of software requirements, applied to a variety of types of software
SE323	Software Project Management	In-depth course about project management
2.2	**Package II**	
SE213	Design and Architecture of Large Software Systems	Modeling and design of large-scale, evolvable systems; managing and planning the development of such systems – including the discussion of configuration management and software architecture
SE221	Software Testing	In-depth course on all aspects of testing, as well as other aspects of verification and validation, including specifying testable requirements, reviews, and product assurance
SE312	Low-Level Design of Software	Techniques for low-level design and construction, including formal approaches. Detailed design for evolvability
SE324	Software Process and Management	Software processes in general; requirements processes and management; evolution processes; quality processes; project personnel management; project planning
SE313	Formal Methods in Software Engineering	Approaches to software design and construction that employ mathematics to achieve higher levels of quality. Mathematical foundations of formal methods; formal modeling; validation of formal models; formal design analysis; program transformations
SE400	Software Engineering Capstone Project	Provides students, working in groups, with a significant project experience in which they can integrate much of the material they have learned in their program, including matters relating to requirements, design, human factors, professionalism, and project management
3	**Completing**	
3.1	CCCS 2xx	Intermediate fundamental computer science courses
3.2	Non-technical compulsory courses	
NT272	Engineering economics	This is a standard engineering economics course as taught in many universities.
NT181	Group dynamics and communication	Communication and writing skills are highly regarded in the software industry, but they are also fundamental to success in collegiate careers.
NT291	Professional software engineering practice	Professional Practice is concerned with the knowledge, skills, and attitudes that software engineers must possess to practice software engineering in a professional, responsible, and ethical manner.
3.3	Mathematics courses that are not SE core	-
3.4	Technical (SE/CS/IT/CE) courses that are not SE core	-
3.5	Science/engineering courses covering non-SEEK topics	-
3.6	General non-technical courses	-

- **A3: Requirements**

"Software requirements identify the purpose of a system and the contexts in which it will be used [13]." Requirements play an important role in software engineering since it is the initial conceptual model of a software system. It is noteworthy that requirements for a software engineering project are a moving target. Therefore, treatment of requirements as static items in software engineering courses should be avoided.

- **A4: Design**

System design is the process that creates a conceptual and abstract model of a software system with its architecture and behaviors, which meets users' requirements. System design includes the specification activities of internal interfaces among software components, architectural design, data design, user interface design, and special tool design.

- **A5: Software Construction**

Software construction is a process that implements a system design by composing suitable components on selected platform. This area covers the knowledge on design refinement, component selection, coding, programming languages, tests, and simulation.

- **A6: Software Verification and Validation**

Software verification and validation are processes that ensure the quality of software. The former checks whether an implementation of the system conforms with the specifications of the system; The latter checks whether an implementation of the system meets the customers requirements. In Verification and Validation, both static and dynamic behaviors of a system should be checked.

- **A7: Software Evolution**

"Software evolution provides cost-effective mission support during pre- and post-delivery stages while maintaining acceptable and satisfactory behavior and validity of assumptions [13]." Software evolution may be implemented by a number of planned and coherent releases for a given system. A number of techniques may adopted in this area, such as program comprehension, release planning, changes identification and control, re-engineering, reverse engineering, maintenance review, migration, system trial, and system replacement /retirement.

- **A8: Software Process**

Software process is a set of durable and repeatable practices in software engineering, which cover the whole life-cycle of software development. Software processes can be classified into three categories known as organization, technology, and management. Software process may be perceived as the infrastructure of process-based software engineering. Software engineering process establishment, assessment, and improvement are major knowledge and techniques for software engineering students [14, 15].

- **A9: Software Quality**

Software quality is one of the basic characteristics and requirements in software engineering. It is a pervasive concept that affects, and is affected by all aspects of software development, support, revision, and maintenance. Software quality can be modeled by a set of attributes such as usability, reliability, safety, security, maintainability, flexibility, efficiency, performance and availability. Software quality is not only implemented in code and other work products, but also influenced by software engineering process and environment. Software quality assurance is at the center of software engineering technologies and practices.

- **A10: Software Management**

Software project management covers practices of project planning, organization, and monitoring. Software project management may be implemented by a set of organizational and management processes. The essence of software management is the synchronization of process activities in software engineering [15].

4.2 Knowledge Areas and Units of SEEK

Further detailed description of the SEEK knowledge areas is provided in Table 2 where each knowledge area is refined by a number of units [13]. The recommended load of each area and unit is given by the number of hours for study.

It is noteworthy that although software engineering programs are offered with a wide range variety of loads, the SEEK recommendation is 494 hours in total, which cover 20+ courses in a program.

5.0 Software Engineering Pedagogy

CCSE is featured not only by its knowledge structures, but also by its studies on pedagogy. CCSE provide a comprehensive set of principles for curriculum design, methodologies for software engineering course development, and core themes and models of the software engineering curriculum for both directors of software engineering programs and instructors of software engineering courses.

The task of the Pedagogy Focus Area Group of IEEE/ACM JTF-SEC is focused on curriculum recommendations based on SEEK. The pedagogy of CCSE encompasses the pedagogy guidelines, principles of software engineering curriculum design, curriculum models, international adaptation, and requirement for professional skills in addition to SEEK

5.1 Pedagogy Guidelines for Software Engineering

A set of 18 guidelines has been developed in CCSE for supporting the development of a specific software engineering program. The guidelines as shown in Table 3 can be classified into those for generic pedagogy, curricula designers, instructors, and students.

5.2 Student Outcomes

A set of expected outcomes is specified in CCSE for an undergraduate curriculum in software engineering. Graduates of an undergraduate software engineering program are required to be able to meet the following criteria [13]:

- Work as part of a team to develop and deliver executable artifacts,
- Understand the process of determining client needs and translating them to software requirements,
- Reconcile conflicting objectives, finding acceptable compromises within limitations of cost, time, knowledge, existing systems, and organizations,
- Design appropriate solutions in one or more application domains using engineering approaches that integrate ethical, social, legal, and economic concerns,
- Understand and be able to apply current theories, models, and techniques that provide a basis for software design, development, implementation and verification,
- Negotiate, work effectively, provide leadership where necessary, and communicate well with stakeholders in a typical software development environment, and
- Learn new models, techniques and technologies as they emerge.

5.3 Program Accreditation

It is recognized that, in order to maintain a quality curriculum, a software engineering program should be assessed on a regular basis. The formal assessment and accreditation of a software engineering program may cover the following areas: faculty, curriculum, laboratory and computing resources, students, institutional support, and program effectiveness. Accreditation may be carried out by periodic external reviews of programs. The aim of accreditation is to assure that a software engineering program meets the minimum requirement as adhered to the standard of a certain accreditation organization.

A number of curriculum guidance and accreditation criteria are available from accreditation organizations of a variety of nations and sectors [2, 6, 7, 8, 11]. For example, the IEEE/ACM Accreditation Criteria for Software Engineering may be referred to [17].

6.0 Conclusions

This paper has reported the design and development of the IEEE/ACM Computing Curricula - Software Engineering (CCSE), and presented the philosophy and insides behind CCSE based on the experience as a member of the committee. The history of CCSE development and related resources of CCSE have been reviewed.

CCSE has been developed to encompass the guiding principles, curricular models, software engineering education knowledge areas, curriculum design, pedagogy guidelines, professional practice, program implementation and accreditation.

Table 2: SEEK Knowledge Areas and Knowledge Units

No.	Knowledge Area	Knowledge Unit	Rec'd load (hrs)	
1	Computing Essentials	Computer Science foundations	140	172
		Construction technologies	20	
		Construction tools	4	
		Formal construction methods	8	
2	Mathematical & Engineering Fundamentals	Mathematical foundations	56	89
		Engineering foundations for software	23	
		Engineering economics for software	10	
3	Professional Practice	Group dynamics / psychology	5	35
		Communications skills (specific to SE)	10	
		Professionalism	20	
4	Software Modeling and Analysis	Modeling foundations	19	53
		Types of models	12	
		Analysis fundamentals	6	
		Requirements fundamentals	3	
		Eliciting requirements	4	
		Requirements specification & documentation	6	
		Requirements validation	3	
5	Software Design	Design concepts	3	45
		Design strategies	6	
		Architectural design	9	
		Human computer interface design	12	
		Detailed design	12	
		Design support tools and evaluation	3	
6	Software V & V	V&V terminology and foundations	5	42
		Reviews	6	
		Testing	21	
		Human computer UI testing and evaluation	6	
		Problem analysis and reporting	4	
7	Software Evolution	Evolution processes	6	10
		Evolution activities	4	
		Evolution processes	6	
8	Software Process	Process concepts	3	13
		Process implementation	10	
9	Software Quality	Software quality concepts and culture	2	16
		Software quality standards	2	
		Software quality processes	4	
		Process assurance	4	
		Product assurance	4	
10	Software Management	Management concepts	2	19
		Project planning	6	
		Project personnel and organization	2	
		Project control	4	
		Software configuration management	5	
	Total			494

The Software Engineering Education Knowledge (SEEK), a key component of CCSE, has been elicited by a set of knowledge areas that cover all aspects of software engineering.

CCSE has been featured not only by its knowledge structures, but also by its studies on pedagogy for software engineering. CCSE has provided a comprehensive set of principles for curriculum design, methodologies for software engineering course development, and core models of the software engineering curriculum for both directors of software engineering programs and instructors of software engineering courses. Although CCSE is a comprehensive software engineering curriculum, a number of important areas have not yet been modeled in it, such as software engineering notations, measurement, and theoretical foundations of software engineering [10, 14-16].

8.0 References

[1]. ACM/IEEE-Curriculum 2001 Task Force, Computing Curricula 2001, Computer Science, December 2001. (http://www.computer.org/education/cc2001/final/index.htm)

[2]. British Computer Society, Guidelines On Course Exemption & Accreditation For Information For Universities And Colleges, August 2001. (http://www1.bcs.org.uk/link.asp?sectionID=1114)

[3]. Davis, G.B., et. al., IS'97 Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems, Association of Information Technology Professionals, 1997. (http://webfoot.csom.umn.edu/faculty/gdavis/curcomre.pdf)

[4]. Gorgone, J. T., et al., IS 2002: Model Curriculum for Undergraduate Degree Programs in Information Systems, published by the ACM, 2002.

[5]. IEEE/ACM (2001), Software Engineering Body of Knowledge (SWEBOK), V.0.95, May, pp. 1-213.

[6]. Institution of Engineers, Australia, Manual for the Accreditation of Professional Engineering Programs, October 1999. (http://www.ieaust.org.au/membership/res/downloads/AccredManual.pdf)

[7]. Japan Accreditation Board for Engineering, Criteria for Accrediting Japanese Engineering Education Programs 2002-2003. (http://www.jabee.org/english/OpenHomePage/e_criteria&procedures.htm)

[8]. King, W.K., Engel, G., Report on the International Workshop on Computer Science and Engineering Accreditation, Salt City, Utah, 1996, Computer Society, 1997

[9]. ACM/IEEE-Curriculum 2001 Task Force, Computing Curricula 2003: Guidelines for Associate-Degree Curricula in Computer Science, Dec 2002. (http://www.acmtyc.org/reports/TYC_CS2003_report.pdf)

[10]. Bagert, D., et. al., Guidelines for Software Engineering Education, Version 1.0, CMU/SEI-99-TR-032, Software Engineering Institute, Carnegie Mellon University, 1999.

[11]. Canadian Engineering Accreditation Board, Accreditation Criteria and Procedures, Canadian Council of Professional Engineers, 2002. (http://www.ccpe.ca/e/files/report_ceab.pdf)

[12]. IEEE STD 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology, IEEE Computer Society, 1990.

[13]. IEEE/ACM JTF-SEC (2003), Computing Curricula - Software Engineering (CCSE), http://sites.computer.org/ccse/.

[14]. Wang, Y. (2000), Software Engineering Processes: Principles and Applications, CRC Software Engineering Series, Vol.1, CRC Press, USA.

[15]. Wang, Y. (2005), Software Engineering Foundations: A Transdisciplinary and Rigorous Perspective, CRC Software Engineering Series, Vol. 2, CRC Press, USA.

[16]. Wang, Y. (2005), Software Engineering Measurement and Analysis: An Applied Framework of Software Metrics, CRC Software Engineering Series, Vol. 3, CRC Press, USA, to appear.

[17]. Barnes, B., et al., "Draft Software Engineering Accreditation Criteria", Computer, April 1998.

About the author

Yingxu Wang is Professor of Software Engineering and Cognitive Informatics, and Director of Theoretical and Empirical Software Engineering Research Center (TESERC) at the University of Calgary. He received a Ph.D. in Software Engineering from The Nottingham Trent University, UK, in 1997, and a B.Sc. in Electrical Engineering from Shanghai Tiedao University in 1983.

Dr. Wang is a Fellow of WIF, a P.Eng, a Senior Member of IEEE, and a member of ACM, ISO/IEC JTC1, and the Canadian Advisory Committee (CAC) for ISO. He is the founder and steering committee chair of the annual IEEE International Conference on Cognitive Informatics (ICCI). He is editor in chief of World Scientific Book Series on Cognitive Informatics and the editor of CRC Book Series in Software Engineering. He was the Chairman of the Computer Chapter of IEEE Sweden during 1999-2000. He has accomplished a number of EU, Canadian, and industry-funded research projects as principal investigator and/or coordinator, and has published over 250 papers and 6 books in software engineering and cognitive informatics. He has won dozens of research achievement, best paper, and teaching awards in the last 25 years, particularly the IBC 21st Century Award for Achievement "in recognition of outstanding contribution in the field of Cognitive Informatics and Software Science." The author can be reached at yingxu@ucalgary.ca

Table 3: IEEE/ACM CCSE Guidelines for Curricula Design

No.	User	Guideline
1	Instructors	Instructors must have sufficient relevant knowledge and experience and understand the character of software engineering.
2	Designers	Curriculum designers and instructors must think in terms of outcomes.
3		Curriculum designers must strike an appropriate balance between coverage of material, and flexibility to allow for innovation.
4		Many SE concepts, principles, and issues should be taught as recurring themes throughout the curriculum to help students develop a software engineering mindset.
5		Learning certain software engineering topics requires maturity, so these topics should be taught towards the end of the curriculum, while other material should be taught earlier to facilitate gaining that maturity.
6	Students	Students must learn some application domain or domains outside of software engineering.
7		Software engineering must be taught in ways that emphasize its engineering nature.
8		Students should be trained in certain personal skills that transcend the subject matter.
9		Students should be instilled with the ability and eagerness to learn.
10		Software engineering must be taught as a problem-solving discipline.
11		The underlying and enduring principles of software engineering should be emphasized, rather than details of the latest or specific tools.
12		The curriculum must be taught so that students gain experience using appropriate and up-to-date tools, even though tool details are not the focus of the learning.
13		Material taught in a software engineering program should, where possible, be grounded in sound research and mathematical or scientific theory, or else widely accepted good practice.
14		The curriculum should have a significant real-world basis.
15		Ethical, legal, and economic concerns, and the notion of what it means to be a professional, should be raised frequently.
16	General	In order to ensure that students embrace certain important ideas, care must be taken to motivate students by using interesting, concrete and convincing examples.
17		Software engineering education in the 21st century needs to move beyond the lecture format: It is therefore important to encourage consideration of a variety of teaching and learning approaches.
18		Important efficiencies and synergies can be achieved by designing curricula so that several types of knowledge are learned at the same time.

INTERNATIONAL CONFERENCE FOR UPCOMING ENGINEERS (ICUE) 2005

The International Conference for Upcoming Engineers (ICUE) is a platform where students and faculty members work together in organizing a conference, and a forum to exchange ideas and promote learning and participation amongst undergraduate students, graduate students and faculty researchers.

After a very successful 3rd annual ICUE, the Steering Committee chaired by **Dr. Sri Krishnan** (Ryerson University) decided to make this a travelling conference in order to benefit students in other regions as well. The steering committee decided that the proposal from University of Windsor IEEE Student Branch was the strongest and hence the planning for the 4th annual ICUE began in December 2005.

The 4th annual ICUE was held in the University of Windsor on May 20th-21st 2005 thanks to the co-operation and support from the University of Windsor's Department of Electrical and Computer Engineering, Ryerson University, IEEE Canadian Foundation, IEEE Region 4, IEEE Toronto Section and IEEE South-eastern Michigan Section. Funding was also received from DaimlerChrysler 3-E Student Activities Fund (Platinum Sponsor), University of Windsor Student Alliance (Platinum Sponsor), JMP Engineering (Gold Sponsor), and the Alumni Association (Silver Sponsor).

The 2-day conference brought together prominent individuals in various fields of engineering and provided students with an opportunity to learn and interact. It consisted of a wireless communication session, automotive session, other technical and non-technical talks by distinguished lecturers, design demonstrations, paper contest, a wine and cheese social and a banquet and awards ceremony.

The wireless communication session had 4 speakers including **Dr. Gerry Chan**, Vice President, Terrestrial Wireless Systems at Communications Research Centre (CRC). The presentations were followed by a panel discussion which allowed the audience to interact with the speakers. The automotive session followed a similar format and included speakers from both industry and academia. Apart from these, there were presentations on various other technical topics such as "Making Mobile Robotics Help People" by **Dr. McIsaac** from University of Western Ontario.

In addition to technical talks, seminars on non-technical topics such as career growth, job search techniques, and stress management were also included. Keynote speaker **Jim Watson**, an IEEE S-PAC National