Informatics in Education, 2004, Vol. 3, No. 1, 31–42

What Computing Curricula is Needed: A Case at the University of Latvia

Juris BORZOVS
University of Latvia and Riga Information Technology Institute
RITI, Kuldigas 45b, LV-1083 Riga, Latvia
e-mail: juris.borzovs@riti.lv

Received: December 2003

Abstract. Computer science undergraduate (bachelor) curriculum of the University of Latvia was developed in late 80th based on curricula of several US universities while keeping strong practical setting. The very core of the curriculum comprises Software Engineering lectures and related information system development course project in the second year. Thus every undergraduate (even theoretical computer science major) is prepared to start professional career of computer programmer by the second study year. It is amazing to realize how well this rather old curriculum conforms to the new ACM/IEEE Computing Curricula 2001.

Key words: computing curricula, ACM, IEEE, University Latvia.

1. Why American Computing Curricula?

By no means ACM/IEEE Computing Curricula 2001 to date is the only model to design specific curriculas at particular universities. European countries have their long-lasting traditional education systems that sometimes differ significantly. However, The Bologna Declaration on the European space for higher education signed by European ministers of education (19 June 1999) implicitly confess that European education system lags behind that of the United States and needs to be improved. The Lisbon European Council of 23 and 24 March, 2000 set the European Union a major strategic goal "to become the most competitive and dynamic knowledge-based economy in the world". As one of consequences, the Career Space consortium was established to develop guidelines for new ICT curricula (Career Space). The Guidelines appeared to be very much in line with the ACM/IEEE Computing Curricula 2001. However, it is not very surprising provided that American computing education is the strongest in the world and The Bologna Declaration calls for American-style grade system.

2. New Benchmark

The Computing Curricula 2001 (CC2001) project is a joint undertaking of the Computer Society of the Institute for Electrical and Electronic Engineers (IEEE-CS) and the Associ-

ation for Computing Machinery (ACM) to develop curricular guidelines for undergraduate programs in computing. The report continues a long tradition of recommendations for academic programs in computing related fields dating back to 1965 (ACM Curriculum Committee on Computer Science, 1965).

The complete CC2001 report will eventually consist of several volumes containing separate recommendations for other computing disciplines, including computer engineering, software engineering, and information systems.

Highlights of the report we are considering (Computing Curricula, 2001) include the following:

- *The Computer Science (CS) body of knowledge.* The authors of the CC2001 report have identified a body of knowledge appropriate to undergraduate computer science programs. Drawing on the structure of earlier curriculum reports, they have arranged that body of knowledge hierarchically, subdividing the field into areas, which are then broken down further into units and individual topics.
- *The CS undergraduate core.* From the 132 units in the body of knowledge, they have selected 64 that represent core material, accounting for approximately 280 hours of instruction. They defined the core as the set of units for which there is a broad consensus that the material is essential to an undergraduate degree in computer science.
- *Learning objectives.* For each of the units in the body of knowledge, the authors of CC2001 report have developed a set of learning objectives designed to promote assessment of student achievement. These learning objectives appear as part of the detailed description of the body of knowledge. In addition to the individual learning objectives, the report outlines a more general set of objectives that all computer science graduates should be able to meet.
- *Course descriptions* contains detailed course descriptions for 47 courses that are part of the various curriculum models. In addition, over 80 additional advanced courses have been identified that would be appropriate for undergraduate programs.

Computer science undergraduate (bachelor) curriculum of the University of Latvia (UL) was developed in the late 80-ties (Computer Science Bachelor Study Programme, 1998), before well-known *Computing Curricula 1991* (Tucker *et al.*, 1991) was released, and was strongly based on existing CS curricula of few North American universities. Acknowledging a leading position of the USA and her universities in information and communication technologies (ICT) industry and education it was and still is a rational decision. Now, after almost 15 years of successful execution of the UL Curriculum it is time to review the Curriculum against the new benchmark.

3. Changes in the Computer Science Discipline

Technical advances over the past decade has increased the importance of many curricular topics, such as the following:

- the World Wide Web and its applications,

- networking technologies, particularly those based on TCP/IP,
- graphics and multimedia,
- embedded systems,
- relational databases,
- interoperability,
- object-oriented programming,
- the use of sophisticated application programmer interfaces (APIs),
- human-computer interaction,
- software safety,
- security and cryptography,
- application domains.

As these topics increase in prominence, it is tempting to include them as undergraduate requirements. Unfortunately, the restrictions of most degree programs make it difficult to add new topics without taking others away. It is often impossible to cover new areas without reducing the amount of time devoted to more traditional topics whose importance has arguably faded with time. The CC2001 Task Force has therefore sought to reduce the required level of coverage in most areas so as to make room for new areas.

Computing education is also affected by changes in the cultural and sociological context in which it occurs. The following changes, for example, have all had an influence on the nature of the educational process:

- changes in pedagogy enabled by new technologies;
- the dramatic growth of computing throughout the world;
- the growing economic influence of computing technology;
- greater acceptance of computer science as an academic discipline;
- broadening of the discipline.

4. Principles

The CC2001 Task Force has articulated the following principles to guide their work:

1. *Computing is a broad field that extends well beyond the boundaries of computer science.*
2. *Computer science draws its foundations from a wide variety of disciplines.*
   Undergraduate study of computer science requires students to utilize concepts from many different fields. All computer science students must learn to integrate theory and practice, to recognize the importance of abstraction, and to appreciate the value of good engineering design.
3. *The rapid evolution of computer science requires an ongoing review of the corresponding curriculum.*

4. *Development of a computer science curriculum must be sensitive to changes in technology, new developments in pedagogy, and the importance of lifelong learning.*
5. *CC2001 must go beyond knowledge units to offer significant guidance in terms of individual course design.*
6. *CC2001 should seek to identify the fundamental skills and knowledge that all computing students must possess.* Despite the enormous breadth of computer science, there are nonetheless concepts and skills that are common to computing as a whole. CC2001 must attempt to define the common themes of the discipline and make sure that all undergraduate programs include this material.
7. *The required body of knowledge must be made as small as possible.*
8. *CC2001 must strive to be international in scope.*
9. *The development of CC2001 must be broadly based.*
10. *CC2001 must include professional practice as an integral component of the undergraduate curriculum.* These practices encompass a wide range of activites including management, ethics and values, written and oral communication, working as part of a team, and remaining current in a rapidly changing discipline.
11. *CC2001 must include discussions of strategies and tactics for implementation along with high-level recommendations.*

5. Structure of the Body of Knowledge

The CS body of knowledge is organized hierarchically into three levels. The highest level of the hierarchy is the **area**, which represents a particular disciplinary subfield. Each area is identified by a two-letter abbreviation, such as OS for *operating systems* or PL for *programming languages*. The areas are broken down into smaller divisions called **units**, which represent individual thematic modules within an area. Each unit is identified by adding a numeric suffix to the area name; as an example, OS3 is a unit on *concurrency*. Each unit is further subdivided into a set of **topics**, which are the lowest level of the hierarchy.

5.1. *Core and Elective Units*

One of goals in proposing curricular recommendations was to keep the required component of the body of knowledge as small as possible. To implement this principle, the CC2001 Task Force has defined a minimal **core** consisting of those units for which there is a broad consensus that the corresponding material is essential to anyone obtaining an undergraduate degree in this field. Units that are taught as part of an undergraduate program but which fall outside the core are considered to be **elective**.

In discussing the CC2001 recommendations during their development, the authors have found that it helps to emphasize the following points:

- *The core refers to those units required of all students in all computer science degree programs*. Several topics that are important in the education of many students are not included in the core. This lack of inclusion in the core does not imply a negative judgment about the value, importance, or relevance of those topics. Rather, it simply means that there was not a broad consensus that the topic should be required of *every* student in *every* computer science degree program.
- *The core is not a complete curriculum*. Because the core is defined as minimal, it does not, by itself, constitute a complete undergraduate curriculum.
- *The core must be supplemented by additional material*. Every undergraduate program must include additional elective topics from the body of knowledge. The CC2001 report does not define what those topics must be, as this additional work can and should vary based on institutional mission, the areas of concentration offered by a given institution, and individual student choice.
- *Core units are not necessarily those taken in a set of introductory courses early in the undergraduate curriculum*. Although many of the units defined as core are indeed introductory, there are also some core units that clearly must be covered only after students have developed significant background in the field. For example, the task force believes that all students must develop a significant application as some point during their undergraduate program. The material that is essential to successful management of projects at this scale is therefore part of the core, since it is required of all students. At the same time, the project course experience is very likely to come toward the end of a student's undergraduate program. Similarly, introductory courses may include elective units alongside the coverage of core material. The designation *core* simply means *required* and says nothing about the level of the course in which it appears.

5.2. *Assessing the Time Required to Cover a Unit*

To give readers a sense of the time required to cover a particular unit, the CC2001 report was to define a metric that establishes a standard of measurement. Choosing such a metric has proven difficult, because no standard measure is recognized throughout the world.

For consistency with the earlier curriculum reports, the task force has chosen to express time in **hours**, corresponding to the in-class time required to present the material in a traditional lecture-oriented format. To dispel any potential confusion, however, it is important to underscore the following observations about the use of lecture hours as a measure:

- *The task force does not seek to endorse the lecture format*. Even though they have used a metric with its roots in a classical, lecture-oriented form, the task force believes that there are other styles – particularly given recent improvements in educational technology – that can be at least as effective. For some of these styles, the notion of *hours* may be difficult to apply. Even so, the time specifications should at least serve as a comparative measure, in the sense that a 5-hour unit will presumably take roughly five times as much time to cover as a 1-hour unit, independent of the teaching style.

- *The hours specified do not include time spent outside of class*. The time assigned to a unit does not include the instructor's preparation time or the time students spend outside of class. As a general guideline, the amount of out-of-class work is approximately three times the in-class time. Thus, a unit that is listed as requiring 3 hours will typically entail a total of 12 hours (3 in class and 9 outside).
- *The hours listed for a unit represent a minumum level of coverage*. The time measurements that have assigned for each unit should be interpreted as the *minimum* amount of time necessary to enable a student to achieve the learning objectives for that unit. It is always appropriate to spend more time on a unit than the mandated minimum.

5.3. *Packaging Units into Courses*

The structure and format of courses vary significantly from institution to institution and from country to country. Even within the United States, some colleges and universities use a semester system while others follow a shorter quarter system. Under either system, there can be differences in the number of weeks in a semester, the number of lectures in a week, and the number of minutes in a lecture.

The authors assumes that a **course** meets three times a week over the course of a 15-week semester and that the individual class meetings run somewhere between 50 minutes and an hour. This schedule is typical for a 3-credit semester course in the United States. Given that some of the available time will be taken up with examinations and other activities, we have assumed that 40 hours of lecture are available over the semester. In addition, students are expected to devote three hours of time outside of class for each in-class hour, which means that the total time that each student is expected to invest 160 hours in each course. Other countries use different metrics for expressing the expected level of work.

6. Summary of the CS Body of Knowledge

A summary of the body of knowledge – showing the areas, units, which units are core, and the minimum time required for each – appears as Table 1. Asterisks denote compulsory units in the UL Curriculum, plus sign – electives.

Table1

Computer science body of knowledge with core topics underlined

DS. Discrete Structures (43 core hours)	**HC. Human-Computer Interaction (8 core hours)**
∗DS1. Functions, relations, and sets (6)	∗HC1. Foundations of human-computer interaction (6)
∗DS2. Basic logic (10)	∗HC2. Building a simple graphical user interface (2)

∗DS3. Proof techniques (12)	+HC3. Human-centered software evaluation
∗DS4. Basics of counting (5)	+HC4. Human-centered software development
∗DS5. Graphs and trees (4)	+HC5. Graphical user-interface design
∗DS6. Discrete probability (6)	+HC6. Graphical user-interface programming
	+HC7. HCI aspects of multimedia systems
	+HC8. HCI aspects of collaboration and communication

PF. Programming Fundamentals (38 core hours)	**GV. Graphics and Visual Computing (3 core hours)**
∗PF1. Fundamental programming constructs (9)	∗GV1. Fundamental techniques in graphics (2)
∗PF2. Algorithms and problem-solving (6)	+GV2. Graphic systems (1)
∗PF3. Fundamental data structures (14)	+GV3. Graphic communication
∗PF4. Recursion (5)	+GV4. Geometric modeling
+PF5. Event-driven programming (4)	+GV5. Basic rendering
	GV6. Advanced rendering
	GV7. Advanced techniques
	+GV8. Computer animation
	+GV9. Visualization
	+GV10. Virtual reality
	GV11. Computer vision

AL. Algorithms and Complexity (31 core hours)	**IS. Intelligent Systems (10 core hours)**
∗AL1. Basic algorithmic analysis (4)	+IS1. Fundamental issues in intelligent systems (1)
∗AL2. Algorithmic strategies (6)	+IS2. Search and constraint satisfaction (5)
∗AL3. Fundamental computing algorithms (12)	+IS3. Knowledge representation and reasoning (4)
+AL4. Distributed algorithms (3)	+IS4. Advanced search
+AL5. Basic computability (6)	+IS5. Advanced knowledge representation and reasoning
+AL6. The complexity classes P and NP	+IS6. Agents
∗AL7. Automata theory	+IS7. Natural language processing
∗AL8. Advanced algorithmic analysis	+IS8. Machine learning and neural networks
+AL9. Cryptographic algorithms	+IS9. AI planning systems
∗AL10. Geometric algorithms	IS10. Robotics
∗AL11. Parallel algorithms	

AR. Architecture and Organization (36 core hours)	**IM. Information Management (10 core hours)**
∗AR1. Digital logic and digital systems (6)	∗IM1. Information models and systems (3)
∗AR2. Machine level representation of data (3)	∗IM2. Database systems (3)
∗AR3. Assembly level machine organization (9)	∗IM3. Data modeling (4)
∗AR4. Memory system organization and architecture (5)	∗IM4. Relational databases
∗AR5. Interfacing and communication (3)	∗IM5. Database query languages
AR6. Functional organization (7)	∗IM6. Relational database design
AR7. Multiprocessing and alternative architectures (3)	+IM7. Transaction processing
AR8. Performance enhancements	IM8. Distributed databases
+AR9. Architecture for networks and distributed systems	∗IM9. Physical database design
	+IM10. Data mining
	+IM11. Information storage and retrieval
	+IM12. Hypertext and hypermedia
	IM13. Multimedia information and systems

IM14. Digital libraries

OS. Operating Systems (18 core hours)	**SP. Social and Professional Issues (16 core hours)**
∗OS1. Overview of operating systems (2)	∗SP1. History of computing (1)
∗OS2. Operating system principles (2)	+SP2. Social context of computing (3)
∗OS3. Concurrency (6)	∗SP3. Methods and tools of analysis (2)
∗OS4. Scheduling and dispatch (3)	+SP4. Professional and ethical responsibilities (3)
∗OS5. Memory management (5)	+SP5. Risks and liabilities of computer-based systems (2)
OS6. Device management	∗SP6. Intellectual property (3)
∗OS7. Security and protection	∗SP7. Privacy and civil liberties (2)
∗OS8. File systems	+SP8. Computer crime
+OS9. Real-time and embedded systems	∗SP9. Economic issues in computing
OS10. Fault tolerance	∗SP10. Philosophical frameworks
∗OS11. System performance evaluation	
OS12. Scripting	

NC. Net-Centric Computing (15 core hours)	**SE. Software Engineering (31 core hours)**
∗NC1. Introduction to net-centric computing (2)	∗SE1. Software design (8)
∗NC2. Communication and networking (7)	SE2. Using APIs (5)
∗NC3. Network security (3)	∗SE3. Software tools and environments (3)
∗NC4. The web as an example of client-server computing (3)	∗SE4. Software processes (2)
∗NC5. Building web applications	∗SE5. Software requirements and specifications (4)
∗NC6. Network management	∗SE6. Software validation (3)
NC7. Compression and decompression	∗SE7. Software evolution (3)
NC8. Multimedia data technologies	∗SE8. Software project management (3)
NC9. Wireless and mobile computing	+SE9. Component-based computing
	∗SE10. Formal methods
	SE11. Software reliability
	SE12. Specialized systems development

PL. Programming Languages (21 core hours)	**CN. Computational Science (no core hours)**
∗PL1. Overview of programming languages (2)	+CN1. Numerical analysis
∗PL2. Virtual machines (1)	CN2. Operations research
∗PL3. Introduction to language translation (2)	+CN3. Modeling and simulation
∗PL4. Declarations and types (3)	CN4. High-performance computing
∗PL5. Abstraction mechanisms (3)	
∗PL6. Object-oriented programming (10)	
+PL7. Functional programming	
+PL8. Language translation systems	
PL9. Type systems	
∗PL10. Programming language semantics	
PL11. Programming language design	

Note: *The numbers in parentheses represent the minimum number of hours required to cover this material in a lecture format. It is always appropriate to include more.*

7. Characteristics of CS Graduates

7.1. *Cognitive Capabilities and Skills Relating to Computer Science*

- *Knowledge and understanding*. Demonstrate knowledge and understanding of essential facts, concepts, principles, and theories relating to computer science and software applications.
- *Modeling*. Use such knowledge and understanding in the modeling and design of computer-based systems in a way that demonstrates comprehension of the tradeoff involved in design choices.
- *Requirements*. Identify and analyze criteria and specifications appropriate to specific problems, and plan strategies for their solution.
- *Critical evaluation and testing*. Analyze the extent to which a computer-based system meets the criteria defined for its current use and future development.
- *Methods and tools*. Deploy appropriate theory, practices, and tools for the specification, design, implementation, and evaluation of computer-based systems.
- *Professional responsibility*. Recognize and be guided by the social, professional, and ethical issues involved in the use of computer technology.

7.2. *Practical Capabilities and Skills Relating to Computer Science*

- *Design and implementation*. Specify, design, and implement computer-based systems.
- *Evaluation*. Evaluate systems in terms of general quality attributes and possible tradeoffs presented within the given problem.
- *Information management*. Apply the principles of effective information management, information organization, and information-retrieval skills to information of various kinds, including text, images, sound, and video.
- *Human-computer interaction*. Apply the principles of human-computer interaction to the evaluation and construction of a wide range of materials including user interfaces, web pages, and multimedia systems.
- *Risk assessment*. Identify any risks or safety aspects that may be involved in the operation of computing equipment within a given context.
- *Tools*. Deploy effectively the tools used for the construction and documentation of software, with particular emphasis on understanding the whole process involved in using computers to solve practical problems.
- *Operation*. Operate computing equipment and software systems effectively.

7.3. *Additional Transferable Skills*

- *Communication*. Make succinct presentations to a range of audiences about technical problems and their solutions.

- *Teamwork*. Be able to work effectively as a member of a development team.
- *Numeracy*. Understand and explain the quantitative dimensions of a problem.
- *Self management*. Manage one's own learning and development, including time management and organizational skills.
- *Professional development*. Keep abreast of current developments in the discipline to continue one's own professional development.

8. Standards for Achievement

8.1. *Threshold Standard Representing the Minimum Level*

- Demonstrate a requisite understanding of the main body of knowledge and theories of computer science.
- Understand and apply essential concepts, principles, and practices in the context of well-defined scenarios, showing judgment in the selection and application of tools and techniques.
- Produce work involving problem identification, analysis, design, and development of a software system, along with appropriate documentation. The work must show some problem-solving and evaluation skills drawing on some supporting evidence and demonstrate a requisite understanding of and appreciation for quality.
- Demonstrate the ability to work as an individual under guidance and as a team member.
- Identify appropriate practices within a professional, legal, and ethical framework.
- Appreciate the need for continuing professional development.
- Discuss applications based upon the body of knowledge.

8.2. *Modal Standard Representing the Average Level*

- Demonstrate a sound understanding of the main areas of the body of knowledge and the theories of computer science, with an ability to exercise critical judgment across a range of issues.
- Critically analyze and apply a range of concepts, principles, and practices of the subject in the context of loosely specified problems, showing effective judgment in the selection and use of tools and techniques.
- Produce work involving problem identification, analysis, design, and development of a software system, along with appropriate documentation. The work must show a range of problem solving and evaluation skills, draw upon supporting evidence, and demonstrate a good understanding of the need for quality.
- Demonstrate the ability to work as an individual with minimum guidance and as either a leader or member of a team.
- Follow appropriate practices within a professional, legal, and ethical framework.

- Identify mechanisms for continuing professional development and life-long learning.
- Explain a wide range of applications based upon the body of knowledge.

9. Benchmarking of the UL Curriculum

Significant time and space is needed to precisely map the UL Curriculum to the CC2001. That's why we simply marked units in Table 1 and listed below major issues only. To summarise shortly, the both curriculas have very much in common and are based on the same principles.

9.1. *Common Aspects*

Both curriculas expect 4-year studies and eventual two-year college as first stage of study. In 2001, the UL established a two-and-a half year college curriculum (according to the government regulations it is called 'first level higher professional education') "Computer Programming". Four semesters are almost identical in bachelor and professional studies, the fifth semester of college comprises 20 weeks of practical internship in industry and final qualification work in programming. Computer programming graduates are eligible to continue education in the third year of bachelor programme upon personal application only.

Programming, Software Engineering, Project courses and Capstone project make the core, and other elements of body of knowledge supports them.

Discrete mathematics is integrated into the introductory curriculum.

Mathematical rigor and scientific method are emphasised.

Familiarity with applications, communications skills, working in teams are encouraged.

9.2. *Differing Aspects*

However, there are also several significant differences.

Computing across curricula, i.e., introductory and special courses for non-majors are not integrated into the UL Curriculum and are not taught by the staff of UL Computer Science Department. For historical reason, the UL Curriculum still contains much more mathematics (2–3 times more) than it is required by CC2001. Part of CC2001 core topics are taught as electives. Working in teams is encouraged but not yet compulsory.

It would be reasonable to reconsider a content of the UL Curriculum in depth to learn out whether it is or is not reasonable to make respective changes, e.g., to make some electives compulsory.

References

www.career-space.com.

ACM Curriculum Committee on Computer Science (1965). An undergraduate program in computer science – preliminary recommendations. *Communications of the ACM*, **8**(9), 543–552.

Computing Curricula 2001: Computer Science, Final Report. The Joint Task Force on Computing Curricula of IEEE Computer Society and Association for Computing Machinery, December 15, 2001, 240 pp.

Tucker, A.B., B.H. Barnes, R.M. Aiken, K. Barker, K.B. Bruce, J.T. Cain, S.E. Conry, G.L. Engel, R.G. Epstein, D.K. Lidtke, M.C. Mulder, J.B. Rogers, E.H. Spafford and A.J. Turner (1991). *Computing Curricula'91*. Association for Computing Machinery and the Computer Society of the Institute of Electrical and Electronics Engineers.

Computer Science Bachelor Study Programme. University of Latvia, accredited in 1998.

J. Borzovs has dr. habilitus degree in computer science from University of Latvia (1999). Currently he is a director(CEO) of Riga Information Technology Institute, deputy director general in Quality Affairs at a/s DATI, assoc. prof. at University of Latvia and Riga Technical University, member of Latvian Council of Higher Education, chairman of ICTE Professional Education Council, member of Experts Commission for Professional Education and Employment at Latvian Confederation of Employers. His interests lies in quality management systems, information technology, software engineering, software testing, standards, intellectual property rights, ICT terminology, audit and consulting. Since 1999 he is active also in occupational standards and computing curricula development at University of Latvia and other Latvian institutions of higher education. J. Borzovs has published over 150 papers.

Informatikos mokymo programa: Latvijos universiteto patirtis

Juris BORZOVS

Latvijos universitete informatikos mokymo programa (kvalifikaciniam bakalauro laipsniui įgyti) pradėta taikyti devintojo dešimtmečio pabaigoje. Remtasi keleto JAV universitetų informatikos mokymo programomis, stipriai pabrėžiant praktinių įgūdžių svarbą. Mokymo programos branduolį sudaro paskaitos programinės įrangos projektavimo klausimais bei kursas apie informacinių sistemų plėtrą. Tokiu būdu kiekvienas antro kurso studentas (net ir kompiuterių mokslo teoretikas), jau gali pradėti profesionalaus programuotojo karjerą. Galima pabrėžti, kad ši pakankamai sena mokymo programa visiškai atitinka naujas, 2001 metų ACM ir IEEE kompiuterių mokymo programos nuostatas.