VICTORIA UNIVERSITY OF WELLINGTON

Department of Computer Science

PO Box 600
Wellington
New Zealand

Tel: +64 4 471 5328
Fax: +64 4 495 5232
Internet: Tech.Reports@comp.vuw.ac.nz

Comparing a Computing Curriculum with the ACM/IEEE-CS Recommendations

Robert Biddle and Ewan Tempero

Technical Report CS-TR-95/21
December 1995

Abstract

Recently there has been an increased emphasis on monitoring the structure and coverage of tertiary education programmes, and on making the results more widely available. For programmes in the field of computing, one way to achieve this is to compare the curriculum to a known standard. In this paper we describe our experience in comparing our curriculum with the ACM/IEEE-CS recommendations. We describe our method of comparison and interpretation, and the conclusions we drew. We also address some issues that arose in doing the comparison, including the length of the overall programme, and differences in core and elective material.

Publishing Information

This paper appeared in the proceedings of *Software Engineering: Education & Practice (SE:E&P'96)*. Dunedin, New Zealand, 23-27 January 1996.

1 Introduction

Understanding the structure and coverage of tertiary education programmes has always been an important but difficult issue. Traditionally, many institutions monitored their own programmes only informally, and little effort was made to enlighten outsiders. Recently there has been an increased emphasis on monitoring the structure and coverage of tertiary education programmes, and on making the results more widely available.

In the department of Computer Science at Victoria University of Wellington (VUW), we review some aspect of our undergraduate curriculum every year as part of our regular internal self-auditing process. In 1994 we decided to review the overall structure of the undergraduate curriculum. In particular, we decided to do a comparison between the VUW undergraduate computer science curriculum and the reference curriculum developed and published in 1991 by a joint task force of the two largest international computer science academic and professional associations: the ACM, and the IEEE Computer Society. Their report [2, 3] states:

> This report contains curricular recommendations for baccalaureate programs in the discipline of *computing*, which includes programs with the titles "computer science", "computer engineering", "computer science and engineering", and other similar titles. These recommendations provide a uniform basis for curriculum design across all segments of the educational community — schools and colleges of engineering, arts and sciences, and liberal arts. This report is the first comprehensive undergraduate curriculum report to be endorsed by the two major professional societies in the computing discipline — the Association for Computing Machinery and the Computer Society of the IEEE.

This report followed publication of several curricula with various specific aims, and the recommendations of the report are designed so to allow a choice of structure and emphasis while making clear the necessary core elements any undergraduate program should provide. ACM and IEEE-CS intend to develop a new report every ten years.

In this paper we present the results of our comparison. We begin with a description of our department's undergraduate curriculum, and follow with a summary of the ACM/IEEE-CS recommendations. The remaining sections detail how we went about doing the comparison, and what we found. We were especially interested in ideas involving a computer science curriculum with a software engineering emphasis, and discuss this in the final section of the paper.

2 VUW Curriculum

Our undergraduate program is a general computer science curriculum designed to allow students within the VUW and New Zealand degree framework to develop the knowledge necessary to begin a career as a computing professional, or to begin graduate study in computer science. Students may credit computer science courses to most undergraduate degrees, however those that specialise in computer science are typically working toward a BSc or BA degree. Both degrees have the New Zealand expectation of three years duration; accordingly our program is structured for three years.

At the time of our study, neither the BSc nor the BA had prescribed computer science major requirements, and so even for students intending to specialise in computer science (by far most of our students) the program is constrained only by the general BSc requirements, or the BA requirements for a BA majoring in "science subjects". However the Science Faculty had approved the introduction of Majors to the BSc for 1995. The regulations for the Computer Science Major were designed to reflect the usual advice given to BSc students specialising in computer science. Consequently our study generally applies to the current situation.

The particular way our curriculum is structured into courses reflects the three year nature of the degree, and our course design. Our course design has been influenced by previous published curricula, courses taught elsewhere, and staff expertise.

First Year

COMP 130: Computers and Applications
COMP 102: Computer Program Design
COMP 103: Data Structures and Algorithms

At first year we have three half-year courses, each representing approximately one seventh of a first-year programme. The course on "Computers and Applications" is a general service course and not considered part of our main undergraduate program. The other two courses are done in sequence, and form the beginning of our main program by introducing programming, program design, data structures and algorithms. Many students intending to specialise in other subjects include these courses as part of their first year program.

Second Year

COMP 201: System and Program Development
COMP 202: Formal Methods of Computer Science
COMP 203: Computer Organisation

At second year we have three full year courses, each representing approximately one sixth of a second-year programme. These courses each deal with one of the major broad areas of the discipline. All require COMP 102 and COMP 103 as pre-requisites, as well as a mathematics course (*algebra*). The Computer Science major requires all second year courses, plus second year discrete mathematics or an approved alternative. This is what most students did before the Major was in effect.

Third Year

COMP 301: Software Engineering
COMP 302: Database Systems
COMP 303: Design and Analysis of Algorithms
COMP 304: Programming Languages
COMP 305: Operating Systems
COMP 306: Data Communications
COMP 307: Introduction to Artificial Intelligence
COMP 308: Computer Graphics
COMP 389: Software Engineering Project

At third year we have nine half year courses, each of which deals with some more specialised area of the discipline. The courses have combinations of the second year courses as pre-requisites as necessary; some also require mathematics courses. The major requirements specify four third year courses plus an approved fifth course. The previous BSc and BA degree regulations had the effect that many students regarded 4 third year computer science courses (representing approximately half of a third year programme) as constituting an informal "major", and often took more than just 4 courses.

The Software Engineering Project (COMP 389) course was introduced in 1995 as a result of splitting the existing Software Engineering course into "theory" (COMP 301) and "practice" parts. Consequently its content does not change the overall computer science curriculum, although the outcome of this study did impact the COMP 389 contents as discussed below.

3 ACM/IEEE-CS Curriculum

The ACM/IEEE-CS curriculum recommendations outline a set of common requirements for any undergraduate program, but they do not take the form of complete suggested courses. Instead, they consist of a large number of "knowledge units", forming 10 subject areas:

- algorithms and data structures,
- architecture,
- artificial intelligence and robotics,
- databases and information retrieval,
- human-computer communication,
- numerical and symbolic computation,
- operating systems,
- programming languages,
  (including an introduction to programming)
- software methodology and engineering,
- social, ethical, and professional issues.

These can be arranged or augmented in many ways to suit the requirements of many different kinds of degree program. In fact, knowledge units can even be split across courses.

Each knowledge unit is described in the report, together with the minimum amount of lecture time necessary for the unit, recommendations for practical work that should support the lectures, and a list of pre-requisite knowledge units.

In addition to the core requirements, the report specifies some requirements from mathematics and science. It also recommends some "advanced supplementary" knowledge units , which cover core requirements material in greater depth, or include topics not considered part of the core requirements. The report lists 28 such knowledge units, including advanced software engineering, artificial intelligence, computer communication networks, computer graphics, database and information retrieval, and programming language translation.

The report also some interesting things to say about "processes" and "recurring concepts" within curricula; we do not address these issues in this paper.

4 Comparison Procedure

To compare our curriculum with the report's guidelines, we first read through the report and studied the knowledge unit structure. We then read through all the course descriptions of our courses, consulted with course coordinators, and attempted to determine which knowledge units were involved in which courses. We then estimated the number of hours for a knowledge unit covered by the course. We concerned ourselves primarily with lecture hours in our comparison, and will not discuss issues relating to practical lab work in this paper.

In fact, the two of us (Biddle and Tempero) each did this independently, and then met to compare our tables and discuss any differences in what we had recorded. As it turned out, our tables were very similar (identical in many cases) and the differences were easily resolved. There do remain some minor doubts, mostly to do with vague wording describing the knowledge units.

5 Comparison Results

We tabulated our results by spreadsheet as shown below. ACM/IEEE-CS subject areas and knowledge units are represented by rows, VUW courses by columns. Summary rows are given for area subject area of knowledge units. Each cell in the table is a number of lecture hours in that knowledge unit (row) for that VUW course (column). The rows at the top show the total number of lecture hours in each course, and the number of those lecture hours we have identified as coming from the core requirements. The columns at the far right show the number of hours for a particular knowledge unit over all VUW courses, and the minimum number of hours for the knowledge unit recommended in the report. The less-than and greater-than signs shown are an attempt to make the differences more visible, showing discrepancies in steps of 1, 2, 5, and 10 hours.

6 Interpretation

Looking at the columns at the far right of the table, we can see how the total lecture hours per knowledge unit at VUW compare with the minimum recommended in the ACM/IEEE-CS report. For most of the units, 34 of 56, the VUW courses provide equal or greater (sometimes much greater) than the recommended number of lecture hours. Another 12 are only one hour less than the suggested minima, so giving 46 of 56 units where the VUW curriculum is not significantly below the recommendations. It should be remembered that the report is written assuming a four year degree, as we discuss below.

Some courses line up substantially with the knowledge units for a subject area. For example, COMP 203 covers most of the Architecture (AR) unit; COMP 301 covers most of the Software Methodology and Engineering (SE) unit; COMP 303 covers most of the Algorithms and Data Structures (AL) unit; and COMP 305 covers most of the Operating Systems (OS) unit.

Other courses cover knowledge units from various subject areas, for example COMP 202 deals with units from Algorithms and Data Structures (AL), Programming Languages (PL), and Software Methodology and Engineering (SE). Of course this is reasonable and explicitly allowed in the recommendations; the report admits the division of the units into "subject areas" is not the only division reasonable.

There seems to be broad agreement between the VUW program and the report's recommendations. The report includes some sample curricula, and the VUW program resembles sample "D": a general BSc program in computer science. There are a few areas where the report's recommendations and the VUW curriculum do not quite line up, and these are discussed below.

At VUW, Not in ACM/IEEE-CS

Looking at first year and second year courses, we see that the lecture material in the VUW courses is almost all in the recommended core requirements. The exception is in COMP 201, in which half the course deals with program design and development for larger programs, and the other half deals with file organisation.

		VUW Computer Science Courses:	130	102	103	201	202	203	301	302	303	304	305	306	307	308			
		Total lecture hours:	36	36	36	48	48	49	24	22	24	24	26	23	24	24			
		Hours from ACM/IEEE-CS KUs:	27	36	34	23	48	49	24	10	18	24	26	6	9	3			
KU	#	Description															**VUW**		**ACM/IEEE-CS**
Algorithms and Data Structures																			
AL	1	Basic Algs & Data Structures			21												21	>>>	13
AL	2	Abstract Data Types		4	3												7	>>>	2
AL	3	Recursive Algorithms		1			3										4	>	3
AL	4	Complexity Analysis			1						2						3	<	4
AL	5	Complexity Classes									3						3	<	4
AL	6	Sorting & Searching			6						2						8	>>	6
AL	7	Comp & Undecidability					4				2						6		6
AL	8	Prob Solving Strategies									9						9	>>	6
AL	9	Parallel & Distrib Algorithms																<<	3
AL	SUM	Algorithms and Data Structures		5	31		7				18						61	>>>>	47
Architecture																			
AR	1	Digital Logic						8									8	<<	12
AR	2	Digital Systems						7									7	>	6
AR	3	Mach Lev Rep of Data						1									1	<<	3
AR	4	Assembly Lev Mach Org	1					13									14	<	15
AR	5	Mem Sys Org & Arch	1					8									9	<<	13
AR	6	Interfacing & Comm						5									5		5
AR	7	Alternative Archs						5									5		5
AR	SUM	Architecture	2					47									49	<<<<	59
Artificial Intelligence																			
AI	1	AI & Robotics	2												3		5	>>	3
AI	2	Probs, State Sp, Search Strats													6		6		6
AI	SUM	Artificial Intelligence	2												9		11	>>	9
Databases & IR																			
DB	1	Overview, Models, Appls	2							4							6	>>	4
DB	2	Relational Data Model								6							6	>	5
DB	SUM	Databases & IR	2							10							12	>>	9
Human-Computer Communications																			
HU	1	User Interfaces	2						2								4	<	5
HU	2	Computer Graphics														3	3		3
HU	SUM	Human-Computer Communication	2						2							3	7	<	8
Numerical & Symbolic Computing																			
NU	1	Number Rep, Errors, Portability						2									2	<	3
NU	2	Iterative Approx Methods																<<	4
NU	SUM	Numerical & Symbolic Computing						2									2	<<<	7
Operating Systems																			
OS	1	History Evolution, Philosophy	3										2				5	>>	3
OS	2	Tasking & Processes											2				2		2
OS	3	Process Coord and Sync											4				4		4
OS	4	Scheduling & Dispatch											2				2	<	3
OS	5	Phys & Virtual Mem Org											6				6	>>	4
OS	6	Device Management				1							2				3	>	2
OS	7	File Sys & Naming				1							2				3	<	4
OS	8	Security & Protection											2				2	<	3
OS	9	Comm & Networking	3										2	6			11	>>>	3
OS	10	Distrib & Real-Time Systems											2				2	<	3
OS	SUM	Operating Systems	6			2							26	6			40	>>>	31
Programming Languages																			
PL	1	History & Overview of PL										1					1	<	2
PL	2	Virtual Machines		1													1	<	2
PL	3	Rep of Data Types																<<	2
PL	4	Sequence Control		4													4		4
PL	5	Data Control, Sharing, Types		2		2	2										6	>>	4
PL	6	Run-Time Storage Management				2						2					4		4
PL	7	FS Automata & REs					10										10	>>	6
PL	8	CFGs & PDAs					12										12	>>>	4
PL	9	Language Translation Systems					3										3		3
PL	10	PL Semantics					4										4	>>	2
PL	11	Programming Paradigms										21					21	>>>>	10
PL	12	Distrib & Parallel Programming																<<	3
PL	SUM	Programming Languages		7		4	31					24					66	>>>>	46
Intro to a Programming Language																			
PR	1	Intro to a Programming Language		12													12		12
PR	SUM	Intro to a Programming Language		12													12		12
Software Meth & Engineering																			
SE	1	Fundamental Problem Solving		12													12	<<	16
SE	2	Software Dev Process	1		3	1			6								11	>>	8
SE	3	Software Reqs & Specs					6		3								9	>>>	4
SE	4	Software Design & Impl				16			9								25	>>>>	8
SE	5	Verification & Validation					4		4								8		8
SE	SUM	Software Meth & Engineering	1	12	3	17	10		22								65	>>>>	44
Social, Ethical, Prof Issues																			
SP	1	Historical & Social Context	10														10	>>>	3
SP	2	Resp of Comp Professionals																<<	3
SP	3	Risks & Liabilities	2														2	<	3
SP	4	Intellectual Property																<<	2
SP	SUM	Social, Ethical, Prof Issues	12														12	>	11

The program design elements of COMP 201 can be found in the Programming Languages (PL) and Software Methodology and Engineering (SE) units. But the file organisation topics seem to be absent from the core curriculum, and from the advanced supplementary material. The material in COMP 201 covers indexing structures (B trees, etc.), extensible hashing, secondary indexing, sequential file processing, and so on. Related topics are covered in AL1 (but they are clearly the more basic structures dealt with in COMP 103) and in DB 1 and 2 (but they assume the higher level approach used in COMP 302).

We find this omission puzzling. Knowledge of file organisation is necessary in much professional programming, and is necessary for understanding implementation issues in Database Management Systems and Operating Systems. A file organisation course was specified in previous published curricula, and we find it odd to find no mention of it here. We contacted one of the authors of the report to check on this, and were told that it was thought that issues from file organisation could easily be included in various units in the curriculum as necessary [1]. Given the weight that this ACM/IEEE-CS curriculum is likely to carry in the development of computer science programmes, we believe that it is a mistake to not explicitly mention file organisation, since not doing so will likely mean it is not taught.

At third year, we find several VUW courses covering topics in greater detail than specified in the core requirements, or with topics not regarded as core requirements at all. The courses that seem to go beyond the recommendations are COMP 306 (Data Communications), COMP 307 (Introduction to AI), and COMP308 (Computer Graphics). At third year this seems reasonable as we tailor more advanced material in accordance with local expertise. This is also consistent with the report's philosophy regarding the advanced supplementary material that:

> A complete curriculum will include not only the common requirements but also certain additional material. This advanced and supplemental material gives each individual student an opportunity to study the subject areas of the discipline in depth.

In ACM/IEEE-CS, Not at VUW

There do not seem to be any major areas specified in the core requirements that are not dealt with by VUW courses. However there are some smaller areas.

One of the smaller subject areas is "Social, Ethical, and Professional Issues"; it includes four units totalling 11 recommended lecture hours. There is no coverage of any of this material in our main program at all. The only coverage of some of the material is in COMP 130, a course not usually taken by our "major" students. This is perhaps ironic, as it is probably the majoring students who will need the best understanding of these issues.

One of the units of the "Programming Languages" subject area is language translation systems. This is not directly covered in our undergraduate program. COMP 202 presents the material on formal languages and automata that is essential for this unit, but we do not follow it up until our graduate course COMP 431 (Compiler Design).

Another of the smaller subject areas is "Numeric and Symbolic Computation"; it includes two units totalling 7 hours of lecture material. Some of the material on number representation is covered by COMP 203 (Computer Organisation), but none of our courses cover methods of numerical approximation or symbolic algebra. However, there are courses offered by the mathematics department which deal with some aspects of these subjects.

Finally, we should acknowledge that in the Architecture subject area there are a few knowledge units with hours below the recommended minima. However, at VUW the department of Physics provides several courses involving electronics and computing which we feel addresses this weakness. These courses figure in an alternative BSc Major in Electronic and

Computer Systems, which also requires Computer Science courses; it is possible for students to do either programme, or indeed both.

Three Years vs. Four Years

In reading through the ACM/IEEE-CS report, we are constantly reminded that the authors assume a North American 4 year undergraduate degree. This makes comparison of our program with the recommendations difficult, because we are trying to achieve the same standard in the context of the New Zealand 3 year degree. Although not as obvious, it is possibly also the case that the authors assume significantly different year or semester structures; we have a year of two semesters, each of 12 weeks.

To a certain extent this difference is reduced by the importance attached to breadth requirements (requirements to do subjects in other areas) in many North American bachelors degrees. Such requirements are less usual in New Zealand in general, and in particular there are no such requirements in the VUW BSc and BA degrees. However, the difference between 3 and 4 years may be the reason for our program covering a number of knowledge units in slightly fewer hours than recommended.

Our program does include some courses that exceed the core requirements, as explained above. This might suggest that we could possibly abandon some of the advanced material in favour of greater emphasis on some more basic units. However, we feel that our coverage of the basics is sufficiently good that the students will benefit more by the provision of the more advanced courses. Because so many of our graduating students leave VUW directly to begin careers as computing professionals, we feel that we must structure our program within a three year limit to support that transition.

Student Selection of Courses

The report recommends that the "core requirements" are necessary for any degree in our discipline, and we conclude that the intention is that every student should study all core units. As explained above, our students currently have great freedom in course selection. Even with the new BSc major requirements for computer science, the students will be free to choose third year courses as they wish.

If it were regarded as essential that all students fulfill all core requirements, then some of our third year courses should be compulsory. Those courses would be COMP 301 (Software Engineering), COMP 303 (Design and Analysis of Algorithms), COMP 304 (Programming Languages), and COMP 305 (Operating Systems). Some of the other stage 3 courses include some core units as well. Probably the biggest impact of making these courses compulsory would be on COMP 303, which currently attracts a minority of students.

Our current thinking is that the necessary practical workload of third year courses, together with the framework of a three year degree, restrict what we can reasonably insist upon.

Mathematics and Science

The report specifies that, in addition to the discussed core requirements, students should study both mathematics and science courses in their degree programme.

For mathematics, the report recommends a minimum of one half-year be devoted to mathematics. These courses should include discrete mathematics and calculus, and a choice of probability, linear algebra, logic, or advanced discrete mathematics. At VUW, students have been required to do mathematics courses both as part of the BSc major requirements and as pre-requisites for many computer science courses. These requirements are for first year algebra and one other course, and for second year discrete mathematics (or an approved substitution): this is roughly equivalent to one half-year. While the VUW requirements do not insist on calculus, the level of calculus specified in the report is covered by the New Zealand secondary school 7th form syllabus — this is also true for the level of probability and statistics suggested.

For science, the report recommends "all computing curricula should include a component that incorporates material from the physical and life sciences". Again, the report recommends a minimum of one half-year be devoted to these science courses. At VUW students are not currently required to do any physical or life science courses. Sample programmes and staff advice do often recommend such courses, but, as appropriate for the future plans of the individual student, we might also suggest courses in psychology, linguistics, philosophy, management, etc.

7 Comparing Software Engineering Curricula

Software engineering is a major focus of research at VUW, and this has affected our general curriculum. As previously mentioned, at VUW, we have recently "expanded" our software engineering offerings by dividing an existing course into theory and practice courses. These courses are typical in that they focus on the management of large software development projects. However, we regard the software engineering discipline as encompassing more than just this. Our attitude is that software engineering includes all aspects of the software development process, including algorithm development, programming language design, many aspects of database development, and formal methods. This attitude is somewhat reflected in the ACM/IEEE-CS recommendations.

There are five knowledge units in the software methodology and engineering (SE) subject area:

SE1 - Fundamental Problem Solving: Introduction to the basic ideas of algorithmic problem solving and programming, using principles of top-down design, stepwise refinement, and procedural abstraction. Basic control structures, data types, and input/output conventions.

SE2 - Software Development Process: Introduction to models and issues concerned with the development of high quality software. Use of tools and environments that facilitate the design and implementation of large software systems. The role and use of standards.

SE3 - Software Requirements and Specifications: Introduction to the development of formal and informal specifications for defining software systems requirements.

SE4 - Software Design and Implementation: Introduction to the principal paradigms that govern the design and implementation of large software systems.

SE5 - Verification and Validation:
Introduction to methods and techniques for verification and validation of software systems.

There is also an advanced supplementary knowledge unit "Advanced Software Engineering" that is essentially a more in-depth study of many of the topics that appear in the core knowledge units, with particular emphasis on methods and tools.

The ACM/IEEE-CS report gives a sample curriculum ("G") with a software engineering emphasis. It describes 9 semesters' worth of material: (first year) Introduction to Software Engineering, Software Methodology; (second year) Software Development, Programming Paradigms; (third year) Software Systems, Formalisms and Computation, Operating

Systems, and a two-semester project. The "software engineering emphasis" seems to come from the early discussion of aspects of the software lifecycle in the "Software Methodology" course. In fact, the material presented in this sample curriculum is not significantly different from sample "D": a general BSc program in computer science (which also includes provision for electives in software engineering).

A weakness we have recently detected in our programme is insufficient emphasis on software analysis and design. While we do discuss the design process in several courses, we believe we do not provide enough opportunity for students to acquire the necessary skills in this area. We believe there is a similar problem with user-interface design and software reuse — both issues that are now acknowledged as being of great importance to modern software development. Unfortunately we do not get much guidance in this matter from the ACM/IEEE-CS curriculum. While it mentions all three issues, it focuses on what to do, rather than how to do it.

8 Conclusions

We concluded that our VUW undergraduate program in computer science was in good agreement with the recommendations of the report. There are differences, but they are slight. The most notable omission was discussion of social, ethical, and professional issues. This topic has subsequently been added to COMP 389 (Software Engineering Project) as a consequence of our study.

Because our program is structured for a three year degree, rather than the four years assumed, some material is covered in slightly fewer lecture hours than the report recommends. For similar reasons, our students also have more choice in selection of advanced topics than is recommended. Our mathematics requirements are in agreement with the report, but we allow our students more freedom in choosing outside courses, instead of insisting on physical and life sciences.

We found the process of comparing our curriculum with the ACM/IEEE-CS recommendations reasonably straightforward and worthwhile. We now feel on firmer ground when we discuss possible course development and programme restructuring, even if the recommendations themselves do not always provide detailed guidance about how to proceed. Most importantly, whether we discuss our curriculum amongst ourselves, or with those outside the department, we now have a principled and independent basis for discussion.

References

[1] Eugene Spafford. Private communication, 1994.

[2] Allen B. Tucker, editor. *Computing Curricula 1991.* ACM Press, IEEE Computer Society Press, 1991.

[3] Allen B. Tucker and Bruce H. Barnes. Flexible Design: A Summary of Computing Curricula 1991. *IEEE Computer*, 24(11):56–66, November 1991.