WYGIWIGY Graduates:
What You Get Is What Institutions Give You

Rick Duley
North Perth, Western Australia
rickduley@gmail.com

Abstract

Each university has its own entry standards, its own curricular priorities and its own graduate profile expectations. Faced with the conundrum of cramming a quart of understanding into a pint pot of curricular time, Software Engineering program designers have created individual solutions and vastly differing programs. Consequently SE graduation parchments have become as meaningful as Sam Goldwyn's verbal agreements. They are not worth the paper they are printed on because no-one knows what they mean! This paper elucidates the problem, explains its origin and postulates paths to solution.

1 Introduction

'University-based Software Engineering Education' is an oxymoron. This is not because SE faculty don't try, it's because the term *'Software Engineering Education'* covers such a wide spectrum of knowledge and understanding as to defy succinct definition. Little wonder that programs from different universities, which graduate students with identically named degrees, vary in content, in breadth and in depth. Even identically named courses within different programs vary in reflection of the interests, priorities and even personalities of the people who write them Unfortunately, two separate undergraduate programs may each produce graduates bearing parchments emblazoned *'Bachelor of Software Engineering'*, *'Bachelor of Engineering (Software Engineering)'*, or *'Bachelor of Science (Software Engineering)* or some such, while the programs they represent differ widely.

This article looks at these variations in the Australian context and asks, *"What do these parchments actually tell a potential employer about the person about to be employed?"* For any two universities the answers would have to be different. This article looks at that problem and postulates some solutions.

2 Educating Software Engineers

Commentators have, for decades, noted an *'increasing number'* of students, employers, faculty, business executives, management specialists, public officials and taxpayers dissatisfied with the product of our universities. It has been pointed out that a university diploma is, in many cases, a receipt for attending classes, that curricula cover too much material giving students little chance of mastering essentials Little wonder that Ron Oliver should observe a decline in the quality of commercially available software despite what we have continued to learn about the software development process over the past three decades [14]. Oliver states forcefully that Computer Science graduates are not properly educated or trained for software or systems engineering.

2.1 Shortcomings

David Parnas has claimed that many top industry researchers and implementors are reluctant to hire CS graduates because their education has not prepared them for the work they actually do (we well might soul-search as to whether today's SE graduate fares better). Graduates get jobs, he wrote more than a decade ago, because their industry is booming while managers doubt the usefulness of their hirees' education [15]. Parnas saw the shortfall as a lack of solid grounding in fundamental science and mathematics rooted in the historical development of the discipline. Whatever the shortfall, the product of our undergraduate programs hasn't gained much respect in industry over the last decade as Mike Feldman points out, referring to a potential employer of graduates and writing *"He was unlikely to hire any of our graduates; he was not looking for 'engineers'; he needed a few 'brilliant hackers'"* [5].

2.1.1 Expectations in the Non-technical Area

Tom Hilburn insists that the dissatisfaction employers express about potential employees does not relate to a lack of technical skills or scientific preparation but rather to a lack of people and process skills [7]. Hilburn cites an inability to communicate effectively, insufficient experience and preparation for teamwork, poor individual work management and a lack of understanding of organisational structures and business practices as major shortfalls. He proposes a new approach to SE education centred on people, process and technology and it is worth noting that his SE curriculum model included eight *'General Education'* electives taken from *"History, Literature, Art, Music, and so on"*. However much we may laud the determination to produce a rounded, *'renaissance man'*, graduate we must also consider the added demand on course time — and it doesn't end with liberal arts.

Since Computing Curricula '91, ethics and the implications of the use of software and technology have required consideration in curriculum design. Considering SE specifically, the recently endorsed Code of Ethics can, and sometimes does, form the basis of an entire course on its own. This theme is itself complicated by the globalisation of the industry. Different regional cultures have differing ethical perspectives and this must be taken into account by designers of software liable to be used in a transcultural context.

To compensate for these shortcomings, one could continue adding subjects to a SE curriculum model and each subject may have its own importance, significance and right of inclusion but there simply isn't time to cover everything in an undergraduate program. Consequentially, undergraduate programs are focussed in ways which reflect the interests and priorities of the people who design them — a uniqueness and individuality which can be viewed as a benefit or a pitfall. Certainly there is benefit in variety, but this author takes the view that this variety should be based on some similarity in the manner in which races and colours are based on the similarity of humanity. Variety for the sake of variety, or for perceived competitive advantage, I see as a pitfall.

3 Program Outlines

Six of the Australian Universities offering Software Engineering programs accredited by the Institution of Engineers , Australia, (IEAust) grant baccalaureates entitled Bachelor of Engineering (Software Engineering). Comparison of their published course outlines shows 65 courses included in the six programs (see Table 1).

Table 1 : Course Topics

Advanced and Parallel Computer Systems
Advanced Object Oriented Concepts
Advanced Software Engineering Topics
Advanced Topics in Distributed Systems
Business Data Management
Computer Communication
Computer Design
Computer Engineering
Computer Structures
Computer Systems and Industrial Electronics
Computer Vision and Image Processing
Data Communication and Networking
Data Modelling
Database Systems
Design and Analysis of Algorithms
Digital Networks and Systems
Digital Signal Processing
Electrical Engineering
Electronics
Embedded Software Design
Engineering Design
Engineering Law
Engineering Methods
Engineering Practice
English for Technical Communication
Ethics and Professional Issues in Computing
Foundations of Computer Science
Functional Programming
Graphics and Computation
Interactions of Society and Technology
Interactive System Design
Introduction to Information Systems Development
Introduction to Programming Environments
Knowledge Representation and Reasoning
Linear Systems
Logic and Computation
Microprocessor Techniques
Network Theory and Analysis
Operating Systems
Physics
Principles of Statistics
Probability Theory and Random Processes
Programming Language Design Concepts
Programming Logic and Microcontrollers
Project Management
Project Team Development
Real Time Systems
Requirements Engineering
Software Architecture
Software Engineering Workshop
Software Internationalisation
Software Modelling and Analysis
Software Quality Principles
Software Reliability and Testing
Software Security
Software System Specification
Software Technology
Systems Programming and Design
Theory of Computation
Verification and Validation

(Because of space restrictions the courses Engineering Management, Mathematics, Programming, Software Engineering and Software Engineering (Project) were deleted from Table 1) It was not possible to enter into detailed comparative discussions with the people who actually teach the courses but efforts were made to compare the advertised content of the courses, not just the names. In the first instance, the courses offered in the first university on the list were entered into the diagram in chronological order. Courses from the second university were entered next, those common to the first university in their appropriate places and the rest in chronological order — and so on for the rest of the universities studied. Since this was not intended to be a competition between universities, the resultant data was converted to a diagram (Figure 2) which simply shows the number of semesters dedicated to individual topics by each university. Figure 1 gives the colour code for the diagram.

Courses from the first four semesters (rows 1-19 in Figure 2) at the first university are commonly supported by the other universities, most notably Mathematics (row 4) and Software Engineering (row 12). This commonality of basics is lost, however, over the following years. Another indicator of this lack of commonality is that less than half the courses offered overall are offered as core by the university represented in the left-hand column.

When we sort the chronological data into neutral alphabetical order (Figure 3) the scatter effect becomes even more apparent. Randomicity is exaggerated by the variation in core size between different programs — note that the program represented in the left-hand column of Figure 2 has twice the number of core courses defined as does the program represented in the right-hand column. Each of the six universities studied devoted either two or three courses to Mathematics but this was the only topic common to all programs. Software Engineering was represented by 1,2,2,4,3 & 0 courses, Projects Management scored 1,0,1,2,1 & 0, Design and Analysis of Algorithms scored 1, 0, 2, 0, 1 & 1. Even greater randomicity in program content would ensue as students choose their elective subjects. It is quite obvious that there is little

One Course	
Two Courses	
Three Courses	
Four Courses	
Six Courses	

Figure 1 : Colour Key

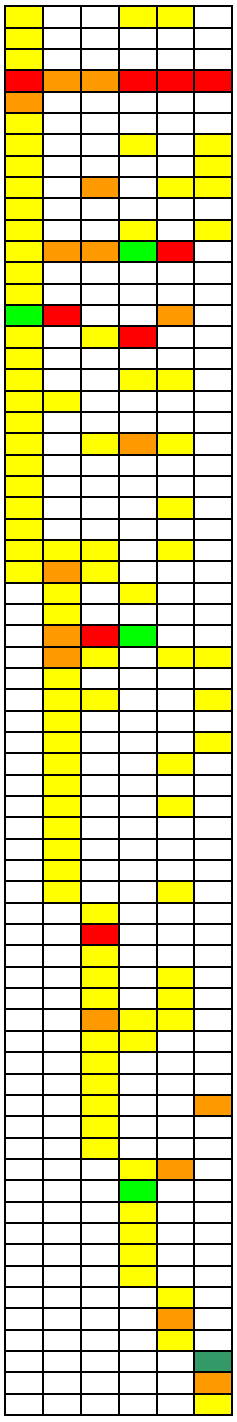

Figure 2 : Chronological

commonality or equivalence of study leading to graduation between the six programs despite the fact that the qualification received on completion has the same title in each case.

3.1 Itemising Anomalies

One university dedicated one course in each semester for the first three years to workshops on the technicalities and required skills which are progressively put into practice as a Software Engineering project — by contrast the Software Engineering Project courses in the other five universities scored 4,3,0,0 & 2. That was a surprise — two of the universities have a Software Engineering Program which does not advertise the inclusion of a Software Engineering Project. Presumably the practical experience to which students in IEAust accredited programs are required to be exposed as *"Exposure to Professional Engineering Practice"* [8] takes care of this aspect of the graduate's education at these two universities. Software Engineering educators, however, must ask how these variations in course content can exist and potential employers must be confused as to what knowledge and skills hirees from differing universities will bring with them.

It should be noted that one could, almost certainly, carry out a similar comparative exercise for any degree and find similar results. That is not the point. The point is that Software Engineering, as a new discipline, is fighting to establish its identity and the variation in education described above will not help in that fight. Resolution of these issues must be a priority goal as Software Engineering matures as a discipline.

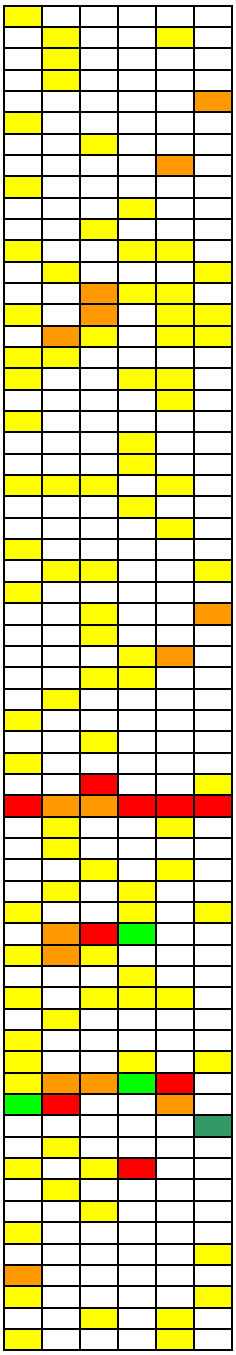

Figure 3 : Alphabetical

3.2 Goals and Priorities

Owen Astrachan pointed out that most students taking economics will not become economists nor most students taking mathematics mathematicians — similarly most students taking Computer Science will not become Computer Scientists [1] but it seems reasonable to assume that most students studying Software Engineering will pursue a career in the profession. If potential employers are to be able to make a fair assessment of what to expect of hirees, and if the general public are to gain a consistent perception of what a Software Engineer actually is, we are faced with a requirement for greater consistency between programs than that demonstrated above. One thing we need is some core of topics which all graduates will have studied and some means of guaranteeing that all graduates have achieved at least some standard minimum level in those topics.

In the absence, at the time of writing, of a Software Engineering Volume of Computing Curricula 2001 the Computer Science Volume provides us with some insight to the

matter of Core Units. This is justifiable because many of the people who currently style themselves *'Software Engineers'* graduated from CS programs. This volume makes the point that the core is not a complete curriculum but merely some units that all students must take — and at any time in the program not necessarily as an introductory unit — and which will be supplemented by additional material. Pedagogy Focus Group Three (*The Computing Core*) was charged with developing curricular models which included *'a short list of courses'* numbering *'four to five beyond the introductory year of study'*. This would indicate a list of 12 to 15 courses as a core, but Software Engineering Education Knowledge (SEEK) document [19] lists 218 *'essential'* topics. Once again, the very breadth of the field in which we work would appear to have struck down the best of intentions if this many topics are to be dealt with in any depth.

3.3 Core — What core?

What, then, is the core? What topics must be covered? What topics cannot be omitted? This argument perennially generates more heat than light and the product of differing opinions demonstrated by Figure 3 is ongoing even with the prestigious Carnegie Mellon University's Software Engineering Institute instituting a curriculum project in 1985. SWEBOK, the Pedagogy Focus Area Group of the Software Engineering Curriculum section of CC2001 and the IEEE-CS/ACM Working Group on Software Engineering Education and Training (to name a few teams) continue the work. In contrast, Mary Shaw argues that the necessary curricular improvements do not require separate courses (from Computer Science) leave alone separate curricula [17].

Christof Ebert suggests that software engineering education needs a flavour of real life which might be provided by business-school-like case studies [4] while Steve McConnell is adamant that software development should be treated as engineering [10]; Ann Sobel advocates introducing formal methods early and extensively applying them throughout the core curriculum [20] while Goodrich and Tamassia argue that sophisticated mathematical arguments are unnecessary and key ideas can be presented visually [6]; Terry Shepard and others propose that more testing should be taught [18] which conflicts with Edsger Dijkstra's 1972 assertion that the more effective programmers don't waste time debugging — they don't introduce the bugs to start with [3]. This litany of conflicting opinions on what is essential, what should be core, can be extended virtually indefinitely.

Confusing this issue is the matter of student expectation as Long and others point out:

> *"Some students fail to see why we are not teaching them exactly what (they think) employers want them to know: traditional C++ programming. Especially those few who have already programmed in C++ tend to baulk at having to be more rigorous about, e.g. behavioural specifications. So one challenge is to make sure students realise that the real CS1/CS2 objective is to lay the foundation for a 35 year computing career, not to providing training for their next co-op."* [9, p.256]

Even this objective deserves expansion. University should not only lay the foundation for a career but, standing as it does in arguably the most formative years of a young adult's life, it should also lay the foundations for the rest of the student's life. As Cardinal John Henry Newman put it:

> *"If a practical end must be assigned to a University course, I say that it is that of training good members of society. Its art is the art of social life, and its end is fitness for the world."* [13, p.143]

3.4 Graduateness

Referring to Engineering Degree (not Software Engineering Degree) programs, Brisk and Parr state:

> *"...the aims of accreditation of today's Engineering degree programs should be very much to ensure that all stakeholders are comfortable that the graduates are suitably equipped for the roles they will take on after university."* [2, p.1]

We will return to the subject of roles in the next section but here consider the problem of the comfort of all stakeholders. From the ongoing debate referred to above we must infer that neither graduates nor employers can be completely satisfied under current practices simply because no program can *teach* all there is to know and no graduate can *know* all there is to know. Graduateness simply adds to the learning load.

In the UK, the Higher Education Quality Council's Quality Enhancement Group stimulated considerable discussion on this matter defining *'graduateness'* as the essential attributes of a graduate. Their report quotes an earlier report which concluded that, while there is a consensus that there is a set of core qualities, there is no agreement as to what they are or how they might be recognised. Across the whole spectrum of higher educational study, the earlier report suggests, these qualities might include:

> *"...the ability to write in grammatically-acceptable and correctly spelt English (or Welsh), a certain level of numeracy, a range of general knowledge, a basic familiarity with Information Technology, and so on."* [16, Section 14]

This seems to imply an obligation of Higher Education to society in general in line with Cardinal Newman's opinion and it finds echo on the other side of the Atlantic where Mike Feldman suggests that graduate programs should include a broad education in liberal studies giving an understanding of the societal implications of the work being performed [5]. Adding the demands of graduateness to the overload of specific topics described above inexorably forces institutions to pick and choose for themselves and creates the situation we face in which supposedly comparable undergraduate qualifications have, in truth, little in common. What an employer gains in a graduate depends entirely in the individual priorities and interests of the faculty who define the programs the hiree studied. Rephrasing — what the employer gets is simply what the local institution delivers. The question is whether or not this is good enough.

4 WYGIWIGY: What You Get Is What Institutions Give You

Given the demonstrated impossibility of presenting all things to all students, program designers may find relief in applying the *'divide and conquer'* strategy by more precisely defining and specifying the learning they seek to pass on and the target audience to whom they wish to pass it.

4.1 Know the Topic

CC2001(CS) makes the point that Computer Science is an important support for other disciplines stating that every university has the responsibility to offer the subject to all students (p.68). This may not equally apply to Software Engineering if we are to make the assumption in the section 'Goals and Priorities' that Software Engineering students are likely to stay within the discipline. Software Engineering programs might in that case justify some uniqueness and distinctive emphasis. Courses, indeed programs, tailored to specialist Software Engineers would be inclined to befuddle the Dip. Ed. or Nursing student whose needs are for more general, more applied, computing. Proponents of Software Engineering must continue to strive for individual, specialised programs concentrating on the creation of software even at the risk of reducing coverage

of some of the deeper and more specialised areas of Computer Science such as hardware and operating systems and precluding participation by students taking other majors.

4.2 Know the Student

CC2001(CS) also notes that students vary substantially in their level of preparation (p.74) so, for precision in program design, designers must find ways to be more precise (and perhaps prescriptive) in their concept of the group to whom they speak. Engineering Schools traditionally demand higher levels of secondary qualification than the more liberal colleges. If Software Engineers are to be the elite of software developers it may be reasonable to make similar high demands on the student intake. Then again, just as the generic term *'Engineering'* covers a range of sub-disciplines (civil, electrical. electronic, mechanical, chemical etc.) it may be reasonable to stream software engineering students after they have completed the core studies.

4.3 Streaming — the *'Need to Know'* Principle

Brisk and Parr, as quoted in section 'Graduateness', may have provided us with the solution to our curricular conundrum when they write of *'roles'*. Given the breadth and scope of the discipline itself, sub-specification of the group *'Software Engineering students'* could be effective — sub-specification on the basis of the roles the students expect to play in industry. Some students might see themselves in a Project Management role (concentrating on process) and would like to go deeper into subjects such as Capability Maturity, Personal and Team Software processes, Risk Management, Metrics etc. Other students may be more interested in Software System Engineering, looking at problem definition, solution and package design — they might be suited by a program concentrating on algorithms, complexity, mathematics and formal proofs. Other students may wish to concentrate on code construction and testing and be suited by a program concentrating on programming fundamentals, programming languages etc. Nothing for nothing, however, and this concept of streaming creates a new problem.

Considering that many institutions are struggling to get even one Software Engineering program up and running, it must be unreasonable to expect that they should, in effect, create several tailored to differing student aims and ambitions. We can, however, slash this Gordian Knot.

4.4 Collegiate Collusion

In a world where the terms *'internationalisation'*, *'globalisation'* and *'rationalisation'* are freely bandied around, it may be asked whether universities, famously individualistic, can remain aloof from these processes for much longer. Government funding for tertiary education might well be endangered if universities continue to duplicate programs, facilities and bureaucracies on campuses perhaps only a stone's throw apart — especially if those programs continue to produce unsatisfied graduates and unsatisfied potential employers. Co-operation and collaboration between tertiary institutions under which Software Engineering Schools could specialise in separate streams would give the student, the discipline and the potential employer considerable advantages. Students from one university could enrol in, and be credited for passing, courses at another. Current emphasis on the use of the Internet and on Distance Education make this a viable option even at a campus remote from collegiate neighbours. At the same time, the load on individual program designers would be relieved by the reduction in material to be covered as a result of specialisation within the field.

Since 2000, the International Software Engineering University Consortium (ISEUC — pronounced "I see, you see") has been moving to take advantage of the variety of courses available through distance education world-wide [12]. This must be a move in the right

direction but progress could be accelerated by active collaboration between neighbour universities instead of enforced reliance on *de facto* variations in curricula.

4.5 Introductory Curriculum Definition

Prerequisite to this collusion is an agreement on common ground from which any university can confidently build its specialist courses and offer them extramurally. As progress continues to be made towards the finalisation of the Software Engineering Volume of Computing Curriculum 2001, the author has been pressing for the delineation of a small subset of SEEK as a basic *'core'*. This core would form the basis of an Introductory Software Engineering Program (ISEP) which **ALL** SE undergraduates should be required to master prior to graduation. Under this scenario, a university could be sure that an extramural student could handle its specialist courses and a potential employer would have the assurance that the potential employee, regardless of *alma mater*, would have a firm grounding in some basics.

5 Only the Beginning

Specialisation is presented as a possible solution to the problem of the academic breadth of Software Engineering. This problem is fundamental to Software Engineering Education and it worsens by the semester, so Software Engineering Educationalists — indeed Tertiary Educationalists in general — do well to remember that they only provide the starting impetus to a graduate's career.

Mike McCracken tells of panic when starting his first job as an electrical engineer. As his initial panic subsided he realised his education wasn't worthless — it was just an entrée into engineering [11]. This, I believe, is a key insight. No graduate is professionally complete. Pursuit of completeness, as we have shown, produces confusion. Continuing the chase means the potential employers, the end users of the tertiary institutions' product (graduates), have to put up with what they are given. Specialisation, clarification and collaboration may produce a better outcome.

Above all, Software Engineering Educators must come to terms with the fact that the core of courses common to undergraduate programs must be reduced. There is a constant tension between academics with differing specialties and priorities as to what is essential and this results in far more being included in the core than can be adequately covered in an undergraduate program of four years. In the SEEK document no less than 218 units are designated as *'Essential'*. No program could do that many units justice and educators and potential employers must accept that. Just as trades apprentices become *'Journeymen'* on the completion of their apprenticeships and go on to learn the finer points of their trade by practical experience before becoming fully-fledged *'Tradesmen'*, so must we expect graduates to grow and develop as they move out into the profession. This has always been, and will remain, the province of the employer and employers must continue to be prepared to shoulder that responsibility.

References

[1] O. Astrachan, Education Goals and Priorities *ACM Computing Surveys*, 28(4es); 1996

[2] M.L. Brisk and P.J. Parr, New approaches to engineering course accreditation, Proceedings of the 4th Baltic Seminar on Engineering Education pp. 1-4, (2000). UNESCO International Centre for Engineering Education. Melbourne, VIC.

[3] E.W. Dijkstra, The Humble Programmer *Communications of the ACM*, 15(10); Oct, 1972 - pp. 859-866,

[4] C. Ebert, The Road to Maturity: Navigating between craft and science *IEEE Software*, Nov,

1997 - pp. 77-82,

[5] Feldman, Michael B., Professor of Computer Science, George Washington University, Washington DC (USA): Inspiring our Undergraduate Student's Aspirations [Web Page] (2001) URL http://www.seas.gwu.edu/~mfeldman/papers/aspirations.html - [Accessed: June 10, 2002].

[6] M.T. Goodrich and R. Tamassia, Teaching the Analysis of Algorithms with Visual Proofs, Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education pp. 207-211, (1998). ACM. New York, NY (USA).

[7] T.B. Hilburn, Software Engineering Education: A modest proposal *IEEE Software*, Jul, 2000 - pp. 44-48,

[8] A. Institution of Engineers, Manual for the Accreditation of Professional Engineering Programs Oct 7, 1999. Institution of Engineers, Australia. Canberra, ACT: URL - http://www.ieaust.org.au.

[9] T.J. Long, B.W. Weide, P. Bucci, D.S. Gibson, J. Hollingsworth, M. Sitaraman, and S. Edwards, Providing Intellectual Focus to CS1/CS2 *Proceedings of ACM/SIGCSE 98*, 1998 - pp. 252-256,

[10] S. McConnell. *Code Complete*, Redmond, WA (USA): Microsoft Press, 1993.

[11] W.M. McCracken, SE Education: What Academia can do *IEEE Software*, 14(6); 1997 - pp. 27,29,

[12] K.L. Modesitt, D. Bagert, and L. Werth, Academic Software Engineering: What Is and What Could Be? Results of the First Annual Survey for International SE Programs, Proceedings of the International Conference on Software Engineering (2001).: URL - http://www.engin.umd.umich.edu/CIS/ISEUC/ISEUCpage.html.

[13] J.H.C. Newman. *The Idea of a University*, Notre Dame, IN (USA): University of Notre Dame Press, 1982.

[14] Oliver, S. Ron, Sucky Software at NASA [Web Page] (2000) URL http://www.caresscorp.com/Oliver.Academy/scky.sftwr.at.NASA.htm - [Accessed: December 19, 2000].

[15] D.L. Parnas, Education for Computing Professionals *IEEE Computer*, vol. 23, pp. 17-22, 1990.

[16] Quality Enhancement Group, The Higher Education Quality Council (UK): What are Graduates? Clarifying the Attributes of 'Graduateness' [Web Page] (95) URL http://www.lgu.ac.uk/deliberations/graduates/starter.html - [Accessed: September 28, 1999].

[17] M. Shaw, Software Engineering Education: A Roadmap, Proceedings of the conference on the Future of Software Engineering (International Conference on Software Engineering) pp. 371-380, (2000). ACM. New York, NY (USA).

[18] T. Shepard, M. Lamb, and D. Kelly, More Testing Should Be Taught *Communications of the ACM*, 44(6); Jun, 2001 - pp. 103-108,

[19] Sobel, A. E. K., Editor, Miami University, OH (USA): Computing Curricula - Software Engineering Volume: First Draft of the Software Engineering Education Knowledge (SEEK) [Web Page] (2002) URL http://sites.computer.org/ccse/artifacts/FirstDraft.pdf - [Accessed: September 18, 2002].

[20] A.E.K. Sobel, Empirical Results of a Software Engineering Curriculum Incorporating Formal Methods, Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education pp. 157-161, (2000). ACM. New York, NY (USA).