THE SOFTWARE CURRICULUM AT WARSAW UNIVERSITY INSTITUTE OF INFORMATICS

W.M. Turski

Rapporteur: Dr. T. Anderson

I would like to explain how we approached, designed, implemented and failed in continuing to implement a course of software education at Warsaw University.

First, consider the overall structure of the university, which caters for some 30,000 students, slightly over a third being part-time students. The university is composed of faculties, such as law, physics and history. Each faculty in turn is composed of 'directions'. The faculty of mathematics and (theoretical) mechanics contains four directions and one of these, known as Informatics, covers computer science. Of 1,200 full-time students in the faculty, 450 take the informatics direction.

The first degree awarded is a Masters degree which used to be obtained after five years study, but now, as a result of economic stringency, is awarded after $4\frac{1}{2}$ years. All informatics students spend two years taking a common base course and then select one of three specialisations: theoretical studies, numerical analysis, and software and systems applications. I will describe the last of these and the common base. A syllabus for this combination is summarised in the following table.

Subjects

	I	II	III	IV	V
GENERAL 780					
Philosophy, Economics Political science, Psychology Pedagogics, Foreign lanugage	spread over years I,II,III				
MATHEMATICS 1080					
Logic, Set theory	60+60				
Linear algebra, Geometry	90+90				
Analysis	120+120	120+120			
Numerical methods		60+60			
Differential equations		30+30			
Probability			60+60		
INFORMATICS 930					
Introduction	30+30				
Programming	45+45	60+90	+60	+120	+60
Computers and systems		60	30		
Mathematical foundations			30+30	30+30	
Dydactics of informatics			30+60	15+75	
SPECIAL TOPICS 1050					
Programming languages			60+60		
Formal languages			30+30		
Operating systems			60+60		
Compiling techniques			30+30	30+30	
DP systems				60+60	
Digital simulation				30+30	
Electives				90+90	60+60
Seminars			60	60	30

Syllabus: Common base, software and systems applications

Entries in the table represent hours of directly supervised study in lectures and exercise classes. A value preceded by a + denotes supervised class work. The entire course occupies a total of 3,840 hours.

Logic and set theory are taught as a single course for historical reasons. The large (50% of the mathematics component) course on mathematical analysis deals mainly with continuous mathematics from the calculus onwards. Students specialising in software receive but a single course in numerical methods. The programming course hours include time spent on projects organised in a similar way to a physics laboratory course to form a programming laboratory. Hardware topics are covered in the course on computers and systems. In the mathematical foundations course we present the elements of automata theory and other models of computability. The course on dydactics is included with the aim that all our graduates should be able to teach.

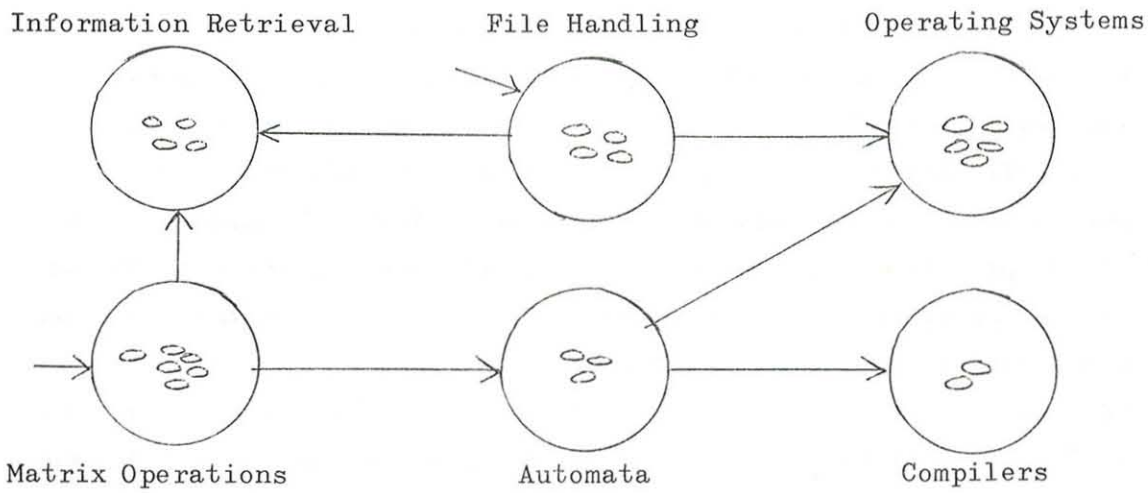
Courses in the special topics component are only attended by the students specialising in software. All are obligatory, but, in one course the student selects three highly specialised topics from a set of 12 based on current research interests of the staff. The research seminars are attended by small groups of students whose topics for their Masters theses are in a common area.

Almost a third of the work load in the above syllabus is mathematical, despite the view held by many of my colleagues, myself included, that the study of mathematics is incidental to software specialists. It would be very nice if fundamental notions such as completeness and closure, which are directly relevant to system design, could be taught within the framework of software subjects, but our own field has not yet progressed far enough to provide sufficient examples as are needed to yield the required insight - mere definitions are certainly inadequate. Proofs of program correctness are an essential ingredient of the thinking process of the programmer, but an appreciation of proof-oriented techniques is best developed in the mathematical disciplines and then imported into computer subjects. Furthermore, mathematics is a marvellous way of teaching the vital effort-saving techniques embodied in the use of formal notation for conciseness of expression and to avoid sloppiness. Finally, teaching adaptability is an aspiration of the entire

university, and we believe this has been achieved to a greater extent in the mathematics faculty than in the remainder of the university. Since informatics is a part of the mathematics faculty we can draw on a high standard of mathematicians to teach these courses and are still able to bias the slant of the teaching so that it fulfils its role in educating computing science undergraduates.

Most of our graduates find employment in the software industry, where they appear to be in great demand. Throughout the course the students are required to develop experience in programming, and they graduate as programmers of four years standing. Proficiency in four very different languages and familiarity with at least two distinct environments is insisted upon (for example Algol60, Fortran, Pascal, Cobol, Cybernet, OS/370, George). An advantage of not possessing a large computer installation of our own is that we do not have, or pass on, any bias towards a particular system. We have some small computers, and easy access to a number of large machines.

The most important factor contributing to the student's practical exposure is the programming laboratory which consists of about 20 problem 'nests'. The nests are partially ordered, and each contains a number of problems. A student can select any problem from within a nest, but must pass through each one in a sequence constrained by the partial ordering.



A part of the network of 'nests' indicating some constraints.

An advantage of this system for the staff is that each nest has a supervisor who is a specialist in the appropriate area, and thus can discharge this 'teaching load' without the necessity of supervising class-work on subjects in which he is not expert.

Students are sent out to acquire experience in software houses either for six weeks full-time at the end of the third year, or for 12 weeks half-time during the fourth year. Most prefer the latter choice. At this stage in their training, the students are competent programmers. Projects are assigned to a sequence of groups of students, one group taking over when the previous group have completed their period of industrial experience. This teaches the importance of documentation and the need for staged design and implementation. The second group makes a report which includes comment on the input received from their predecessors. The first group is not graded until the second group have completed their stage. Professor Randell asked if this process was unending. Professor Turski indicated that it must seem so to the groups, since projects usually ran for about two years. Mr. Laver asked how the scheme was viewed by the employers. Professor Turski explained that the employer was asked to regard the continuous flux of groups of four or five students as if they were a permanent group whose performance could be approximately that of two people. The extra work for the employer is compensated by the opportunity he receives to market himself to future graduates.

A further contact with industry is that many M.Sc. thesis topics originate in proposals from the software houses. The student who selects such a topic receives dual supervision, both academic and industrial, and his industrial supervisor sits on the examination board. Often this approach leads to subsequent employment, but sometimes provides a warning to avoid a particular firm (or a particular graduate). Typical example topics are

(i) A mini FORTRAN cross compiler running on a RIAD 30 producing code for a mini computer. (Direct programming.)

(ii) A model of a special purpose operating system (message switcher) for a mini computer, running on an IBM/370 machine but calibrated to yield exact timing data. (Model design.)

(iii) A study of the effectiveness of the tape handling routines at a savings bank. (Study of an existing system.)

I now wish to identify a number of reasons for what amounts to a gradual disintegration of the academic programme I have described. These may be grouped under the general heading of 'fatigue'.

A great deal of effort has been put into the preparation of many new courses, but although these have been very successful, the effort involved is not appreciated by the university. Similarly, a lack of suitable teaching aids has necessitated the production of 'home-made' aids, an activity which, of course, receives no recognition at all. The resulting slow academic progress of the teaching staff (especially junior) has led to dissatisfaction and a rapid waning of enthusiasm. Interaction with industry, while excellent for the students, has led to staff operating as unpaid consultants, something which is very difficult to prevent or place on a rational basis. In some instances, heavy industrial involvement has led to problems of identity for academic staff. Student interest in the course is high, but many feel unable to complete the heavy workload and as a result receive no degree whatsoever. A high rate of attrition brings pressure from the university administration - wastage of funds - and from the students - wastage of effort. The reduction of the course length by half a year was achieved by simply deleting the last semester. Obviously the syllabus should now be redesigned - just when, for the first time, stability had been reached with a full run through of students. Finally there is a lack of sufficient facilities for experimental purposes.

I may have painted too black a picture, but my expectation is that the programme will be fossilised, in that although it will be maintained our future plans are now unlikely to be implemented.

Discussion

Professor Page began by nostalgically reflecting on his own undergraduate mathematics background but suggested that many of the reasons advanced in support of teaching so much mathematics were the same kind of reasons as those advanced for teaching latin. Although these reasons do have some validity, the removal of latin from the curriculum left time for other subjects which, it is hoped, achieve the same results and are themselves of some use. Similarly, would not a shift from continuous to discrete mathematics be advantageous in the context of what is essentially a discrete discipline. Professor Turski "Probably, yes. But I had so many years of latin". It was agreed that a further advantage of a high proportion of mathematics in the course is that someone else does the teaching. Professor McKeeman asked what was known about students who specialised in theoretical studies and then went into industry. It was pointed out by Professor Turski that this rarely happens. Numbers taking that specialisation are low in any case and most go on to other universities. An industrial research laboratory would be an acceptable alternative. Mr. Laver enquired as to the overall attrition rate for students and wondered if it would not be preferable to produce more, if slightly less capable, graduates than just a few of very high standard. Professor Turski expressed his personal opinion that mediocracy would be the worst thing that could happen. In the first year almost 50% of the students drop out, some switching to mathematics. (A principal cause is the first exposure of the students to programming exercises set by Professor Turski!) In the second year a further 25% of the remainder are unsuccessful. However, years after the first can be repeated (almost indefinitely). Professor Griffiths felt that not enough emphasis was placed on applications. Professor Turski replied that in addition to the elective course there

was also the course on D.P. systems. He conceded, however, that engineering and science applications were only covered by the numerical analysis specialisation. Professor Reynolds asked if the universities produced the majority or only the elite of software personnel. Professor Turski quantified the production of software personnel by universities as being around 300 per annum, while other higher educational establishments produced roughly 1200 per annum. However, nearly half of Poland's computer manufacturer's programmers are products of the universities.