WHAT TO INCLUDE IN COURSES

R.W. Hamming

Rapporteur: Ms. D.M. Bowman

The discussion began with an introduction from Professor Hamming in which he related his own experience in curriculum design and pointed out those factors which he considered most important. Thence, the discussion proceeded with questions and comments from various members of the audience.

Professor Hamming initiated the discussion from the chair by speaking about his experience in curriculum design. He felt qualified to comment on this matter because as a representative from industry he had been invited on many occasions to assist in university curriculum design. Recently as a member of an ACM curriculum committee, he had prepared a small document which contained some rules that he considered most important to observe in curriculum design. The first one of these rules was that 'the curriculum should be cumulative'. Professor Hamming explained that by 'cumulative', he meant that the courses should build one on top of another. In defence of this remark, he pointed out that anyone who takes Calculus discovers they relearn their earlier mathematics while taking the Calculus. Further, he commented that what one genuinely wants to teach in a course should be used in subsequent courses. He felt that in order for man to truly master a subject, he must use it day after day after day. The second rule concerned being able to measure what a 'computing science' is. Professor Hamming cited from a sign on his office wall: 'Without measurement it is difficult to have a science'. He felt that curriculum has to cover both theory and practice, and that computing science should entail a fair amount of probability and statistics. A third consideration was that the university has to prepare the student for the distant future and at the same time prepare him for a job tomorrow. Lastly, computing people should look

beyond designing a curriculum for themselves, as they have to look-out for the needs of the rest of the university. At this point, Professor Hamming remarked that he did not know what should go into the courses, but simply that he had pointed out some of the conditions to be met in the design of these courses. As chairman of the discussion he invited the audience to start with what they thought ought to go into the discussion.

Dr. Lavington began by enquiring whether the current discussion was addressing the design of a non-specialist course or a specialist computer science course. Professor Hamming responded that since Professor Naur had discussed course design for the general student fairly extensively, he felt that the present discussion might be directed toward computer science majors. However, Dr. Lavington, Professor Randell, and Professor Naur agreed that the discussion should also address the non-specialists such as medicine or social science.

In response to this, Dr. Scoins described a problem which he noted in relation to the service courses which the Computing Laboratory offered to the other departments of the University. Essentially, the problem concerned whether or not to teach non-specialists before, or after, they had learned some of their own subject. If the course was offered during the first year, the parent department was unable to supply examples that their students would be familiar with, and so the Computing Laboratory had to use examples from elementary mathematical functions. This did not prove satisfactory because it had little relation to do with what engineering students were doing, for example. The alternative that was tried was to teach the students in their second year after they had learned some engineering in the hope that they would benefit more from being able to do engineering examples. However, the difficulty encountered in doing this was that the students were too impetuous and attempted to get the computer to solve their 'more complex' problems. They would not take the time to learn sufficient computing skills. Dr. Scoins summarised that this problem of when

to teach a non-specialist course and how to construct meaningful examples was a serious difficulty and that in lieu of a solution some departments were oscillating between first year and second year and back again.

From the chair, Professor Hamming offered his solution to the first problem of extracting from the parent department what sort of examples they wanted their students to be able to do on the computer. His approach to this situation was to choose someone of "intelligence" from the parent department, call him up and go and see him, and talk with him many times until he discloses enough about what he wants his students to do.

Professor Michaelson suggested that Dr. Scoins might recollect that some systems are only stable when they oscillate and so one should just let them oscillate.

At this point, Professor Aspinall offered his comments as an outsider from a department of electrical engineering. He suggested that although an engineer usually receives only a course in FORTRAN, BASIC, or ALGOL, that it would hardly solve his problems, and that 'it might help if a computer science department was involved in what you design'. He emphasised that in the coming years, engineers are going to want a course in computing about the design of systems. He thought that the developing subject of systems theory may well become as important to engineers as control theory had become over the last two decades.

Professor Hamming, from the chair, related his personal experience on the subject of systems engineering. About 1960, he was invited to go to Stanford each winter quarter and teach a course in systems engineering. At that time he did not see how to teach the material except by anecdote and case history. However recently, a friend of Professor Hamming, who is a master expert in systems engineering, has written ten essays which in the opinion of Professor Hamming would serve as a basis for a course in systems

engineering. Professor Hamming continued by saying that systems engineering is very very hard to teach, and that one can tell a systems engineer from other engineers not by what he says, but by what he does. He commented that it is hard to get the person to see the light of the problem and make value judgements, and that universities in the United States have been 'sneaking out of that for years' by teaching science rather than engineering. Thus we have got a very real problem.

Dr. Lavington agreed, and added that although computing science is partially at fault, often the difficulty is 'tied up with politics'. He observed that for political reasons his department is only allowed a very small time slot in which to teach any computer topic to non-specialists. Further, he criticised that in a thirty-hour systems course for non-specialists such as Professor Aspinall wanted, there would be no solid material to pin comments on, and so the course would degenerate into 'talky, talky' sessions that would not be taken seriously by the inexperienced student who perhaps has never programmed the machine.

At this point, Mr. Voysey commented that over the years, from listening to people talk particularly of simulation examples in topics of programming examples, there seems to be an idea that finding examples in topics of programming and simulation teaching should somehow or other be easy. He added that he was totally taken aback by this, because as an outsider he would expect it to be terribly difficult. Since learning 'by example' and 'by experience' is very important for a number of classes of people, he would expect an immense effort to be made in providing examples which are not too complicated but still have meaning to a wide range of people in their discipline. 'It is a bit like having a library and expecting people to use it by knowing nothing about classification or never using an index' (Voysey). Mr. Voysey continued by suggesting that university curriculum design should involve more of the experience of students outside the university as to what they actually use and what is their

present experience. He added that curricula and examples seemed to be placed as curricula first and then examples within the curricula - 'Somehow it is a secondary thing which we will somehow or other do later'. He concluded by saying that he would have thought that the idea of examples being secondary is the inverse of what it is supposed to be. Professor Dijkstra interjected 'Do you mean to say there is no point in showing an example unless you can think what it is an example of?' In response to this, Mr. Voysey commented that more stress should be placed on examples in teaching and that the examples should be sorted out very carefully so that they build on each other. Further, he added as an example that when simulation was first used, there were a number of people in a number of disciplines who found it quite a mental gymnastic to stand outside a simulation and have to read into it whatever was required.

At this point, Dr. P. Lauer wanted an opinion on a technique which he had used for teaching students as follows: with his class, he would select a programming package, take it apart, and teach the programming required to understand that package. In the process of taking the package apart, the class would consider such questions as: 'Does this package really model what you are trying to model?' 'How much can you trust this package to give you the information that you want?' Dr. Lauer added that for purposes of system design one could use this technique very nicely to do the job properly.

Professor Hamming, from the chair, agreed that if there is a regular package in use in the field, that to pick it apart and show the pieces is a very nice idea.

Professor Ercoli then began to comment from his number of experiences at trying to teach what a computer does and what computer science is, to various kinds of people. He began by stating that one cannot speak of the 'uneducated' individual in general without making distinctions. He added that it is one thing to explain what a computer does or can do to a scientist or engineer who is trained in induction, deduction, using axioms, and checking proofs, and

another to explain this to a medical doctor, for example, who has a completely different way of thought. Professor Ercoli found that very often students of law were highly talented in the field of logic and that they could make long chains of analogies, provided the pattern was that of a tree. However, these same people could not follow loops in the logic and so would be lost at the introduction of iteration and recursion. Further, Professor Ercoli supported a point that Ms. Goldberg had made in one of her lectures that assignment statements (everything which is connected with functions of numbers) are difficult. He explained that assignment statements create a difficulty because certain categories of people are not able to abstract, and therefore, for them, the concept of a variable is very difficult. Although these people can use a pocket calculator to add three plus two and get five, they cannot understand a computer program which does not just add two constants, but adds any two numbers and perhaps adds n numbers together. Again, he stressed that the approach one must use in teaching has to be quite different for different groups of people and the adding of n numbers is something which medical doctors and lawyers cannot grasp; whereas, it is extremely familiar to chemists, engineers, and mathematicians. Further, Professor Ercoli proposed that if you want to teach someone something, you must know how their mind works. In defence of this point, he related the example of an electrical engineer who had used analogue computers and who needed to be reminded to use n times the same adder to add n numbers instead of n adders. Lastly, Professor Ercoli observed that there were many very able people who could do extraordinary things and yet they found the concepts in computing science difficult to grasp. He questioned whether it was a problem of motivation or how their minds work.

Professor Hamming agreed that Professor Ercoli had made a very good point. Once as a Professor of Civil Engineering he had a similar experience teaching civil and electrical engineers. The electrical engineers who were used to a high degree of abstraction

could add n numbers, neither specifying what n was nor what the numbers were, but the civil engineers found that kind of thing quite awkward. Professor Hamming concluded that our general assumption that everyone can abstract is wrong, and reminded the audience that the agony of learning what 'x' was in algebra was not overcome in one afternoon.

Further to this, Professor Ercoli explained that he had two children with, of course, the same home background and he would assume their genetic background is more or less the same. In spite of this, one child abstracts markedly more easily than the other, and he did not know whether to attribute this to motivation or what?

Professor Hamming, from the chair, responded to this by stating that you must approach a person with things he already knows in order to teach him something. From his own experience, Professor Hamming was able to teach iteration, loops, and index registers, very easily to a chemist by explaining it in terms of a zone melt. But, he admitted, for the case of a lawyer, he did not see how to explain recursion, and that this was a very real problem.

At this point, Professor Brooker commented from his experience of teaching first year social science students in a class of about one hundred. Although he thought that he had made every effort to choose very simple problems, at the end of the course of about fifteen lectures (including five on simple FORTRAN), about thirty of the class were competent. Looking back, he observed that even the students who were struggling preferred BASIC to FORTRAN because they had difficulty grasping the difference between integer and real types. He had felt that perhaps the students were not motivated and so thereafter, he confined his efforts to more motivated students, such as economists who use statistics.

From the chair, Professor Hamming commented further from his own experience of teaching the non-specialist. The case he described

was that of teaching the secretaries at his office to do text editing. These people were high school graduates, they were certainly not mathematically competent, but they were used to working with text. The technique used to teach these people originated in Mexico, and is called 'each one teach one'. The philosophy used is that one secretary was shown what to do and then the other secretaries showed each other what to do starting from the first secretary. Professor Hamming emphasised that sooner or later these secretaries were doing relatively complex tasks such as 'search all the text from here for all occurrences of such and such, find them, and then substitute for all these occurrences something else like that'. He added that he was very impressed with the rate of success of this method.

In reaction to this, Professor Wells wondered: 'How many of the people in this room learned their computing by the "each one teach one" principle. Certainly, that is the way I learned.' Professor Hamming admitted that he had learned to teach himself, and upon enquiry found that most of the audience had as well. He agreed that it seemed to be one of the more effective methods.

Dr. Lavington then questioned whether this method could be applied to students in the same fashion as employers apply it to their secretaries. He was concerned (in agreement with Professor Brooker) about the motivation of the students. He found that a voluntary class of part-time graduate students of mixed background had no trouble at all because they had signed on at their own free will. He felt that related to the motivation was the attitude of the parent department. If the parent department regarded the course as a chore and an unnecessary waste of time then the students did not do very well. Professor Hamming retorted to this: 'Well my experience with students at Princeton is that the students are usually more aware of the desirability of computing than the faculty in their own departments'.

On that point of motivation, Dr. P. Lauer then added that he has made a practice of telling his students a few things to help motivate them. For example, he might tell them: 'If you took this course outside the university it would cost you £300', or 'Any job you take, no matter what you do, if you say you have got this computing experience, then you can get £10 or £20 more added to your salary'. Dr. Lauer felt that this was not a bad motivation.

Professor Hamming then pointed out that there seemed to be general agreement that it was hard to teach and coordinate a group of people and that a person with a problem learns better from his peer group. So he suggested that perhaps people should be encouraged to work in groups and grapple with the problems by the 'each one teach one' method. In computing science, he added, that one could organise this by assigning two or three routines to a group and let them split up the problem amongst themselves.

On the topic of finding suitable examples, Professor Capriz, then stated that if he were teaching mathematics or graph theory to engineers, physicists, or chemists doing research now, that he would find the kind of specialised mathematics that they use too difficult. For students, they would have to have a wider background in their own science before they could grapple with advanced examples. As a compromise Professor Capriz suggested that since in the teaching of structural engineering the classical examples are simple ones from many years ago, that computing science should do likewise and use examples from the simple speed calculations that engineers did on the computer 20 years ago, rather than select complex and difficult examples from the present activities of the engineer on the computer.

On the question of motivation, Professor Naur then spoke from his experience teaching computer science students. He felt that in order to achieve motivation it was necessary to use the 'shock' treatment of 'just throwing them into the deep end' as follows: for first year students, after just a few weeks of learning the

elements of a programming language, they are assigned a big project. Professor Naur felt that this pressure to write, debug, and document a specific problem was absolutely vital to motivate the students to consult their books and learn by their experience. He added that the questions of content of a course and of form of a course are intertwined completely because certain things can be taught only by certain techniques. 'Who would ever learn to play the piano by reading a book?'. Finally he added that in the case of computing that there is no substitute for the humdrum experience of getting into contact with the computer, writing a program, and finding and correcting the errors, in order to acquire an understanding which one cannot get otherwise.

At this point, Dr. Ogden objected that so far the discussion had been about the teaching of people who were at some stage in their careers going to use the computer. He felt that the wider issues of what else people should know about computers and the impact of computers on society were being avoided. He questioned what should people know about things like: how computer systems are constructed, what is a small computer system, what is the difference between a general purpose and a special purpose dedicated computer, and what is the esoteric concept of real-time.

In response to this, Professor Hamming mentioned that his secretaries who do text editing know very little else about the computer except what they are supposed to do and what will happen when they do certain things. He stated that the secretaries do not know how things are stored or even what the machine looks like. 'But do they need to know these things? I do not know.'

Dr. Holt then added apologetically that he was bored by the whole discussion and that he thought that it was an enormous pity that such a group of really interesting people could not do better. Dr. Holt offered that he had made a list* which he was sure would

* See Appendix

be controversial, of what in his opinion, anyone who goes through higher education ought to know about computers. He emphasised that there ought to be some kind of division of the classes of material to see what the relations are in a priority to help discern what the general person should know as opposed to the computer professional.

In response to this Professor Hamming, from the chair, asked 'What does the educated person need to know about the telephone system except how to look up a number and dial?' Dr. Holt responded that he thought this was an excellent example because he thought it is very bad that the average person in our society has never stopped to think about the telephone system. He believed that the average person would be enormously well served to have the telephone system raised to his level of consciousness so that perhaps he might think there is something wrong with it. Professor Hamming retorted that the greatest fault of the telephone system was that it was geared to get your voice to a geographical location and not to a person. However, he commented that he did not see why this should be discussed in an undergraduate course or why the educated person needs to know this. Professor Michaelson added that there are lots of things like that a person should know, but that one cannot expect to pack them into an undergraduate course. Professor Hamming continued by saying that perhaps there was no minimal amount people need to know about computers, but that people wanted to know how to use them to get what they wanted regardless of which pulses do what and where. Finally, Professor Michaelson stated that most people were not going to use the computer, but that they needed to know about the politics of their use.

Mr. Tully then added that his own suspicion was that many people want to know, at least psychologically, how computers work. He added that he agreed with Dr. Holt's list of items that no course should omit. He emphasised that he felt how much one teaches both specialists and non-specialists is absolutely critical. At

this point, Mr. Tully stated that he would be very happy to teach from Dr. Holt's list, but that he was at a loss for material about theories of systems. Dr. Holt agreed that it was very difficult and that he did not know of any material that is satisfactory for that purpose.

Professor Ercoli commented that he had once heard Professor Hamming give the advice of 'write a book' at a Computer Education conference in the United States. Professor Hamming responded that since he had been discouraged at what the educated individual ought to know about computers without becoming a computer expert and since at that time few machines were available for 'hands-on' experience, he had decided to write a book for a course with 'no hands on'. He recommended that anyone with strong feelings should write a book, and warned that when writing a book that one should keep in mind how much of their material will be relevant a long time later and how much is just covering minute technique at the present moment.

As time was running short, Professor Hamming felt it appropriate to close the discussion.

Appendix: Dr. Holt's List

Items not to be omitted in learning about computers.

1. System Concepts.
2. Experience program writing and building.
3. The Structure of the Computer business.
4. What is the state of the art.
5. Concepts of computer Use

   Common and technical,
   e.g. calculator, giant brain, robot problem solver, communications medium controller, etc.

6. Social Issues

Concepts	Practice	Regulation
Person/computer	Security	Legislation
Communication and	Obsolescence	Professional ethics
on what it depends	Reliability	
	Tracing responsibility	

8. System design practice

e.g. Computer-aided conversation
Postal service
Information service to a community of users
A Market

9. On the Relation of resource Structure and problem definition

10. On the limits of systems thinking

11. Practice in Representation