# A Computing Course for All Freshmen Engineering Students (Half Architecture, Half Programming)

**Yale N. Patt, Kevin J. Compton**
**Dept. of Electrical Engineering & Computer Science**
**University of Michigan, Ann Arbor**

Abstract

The EECS Department at Michigan has recently changed its first required course for its own majors from a high level language programming course to one that spends the first half of the semester studying the underlying hardware structure. This course is also being selected as the course of choice by many non-computer engineering majors with good results. This paper describes the course, our rationale, and some data as it relates to non-majors.

## 1. Introduction/Rationale.

The Electrical Engineering and Computer Science Department at the University of Michigan, Ann Arbor, has recently instituted a major change in its requirements for our undergraduate majors. We feel strongly that the conventional model practiced throughout the country of introducing majors to computing via a high level language programming course is flawed. Our answer to whether it is better to start with Pascal, C, or C++ is "none of the above." Some professors in other engineering disciplines may say, "Right! Fortran."

Wrong! No programming language. For our own majors, we felt it was important to present computing from the bottom up, that is, to first introduce the basic logic structures, then the basic computer, and only after the student understands what is going on underneath do we begin to teach him/her to program, in our case, in C. What we didn't expect is the resounding acceptance by a number of other engineering students. Statements from some mechanical engineers, for example, have pointed to the automobile, and the large number of features that are microprocessor controlled. "We have a pretty good idea how to deal with combustion," they argue, "but what the future automotive engineers are going to have to know is how to make appropriate use of microprocessors in the automobile, and that means understanding how the microprocessor works, not simply how to program it."

The course, EECS 100 Introduction to Computing, was developed primarily by the two authors, although it has benefited greatly from input from (first) other members of the Computer Science and Engineering curriculum committee, in particular Ann Ford and David Kieras, and (more recently) from the TAs who have suffered through the first three iterations, in particular, PhD student Sanjay Patel. Details of the course are described in a companion paper in this conference, "Introduction to Computing: The Correct (Bottom-up) Approach"[1]. In this paper we want to emphasize the details that are particularly relevant to the non-computer engineer.

2. Relevant Points.

Memorization

Since the course builds from the bottom up, we have found less memorization of rules is required than in traditional programming courses. Students more nearly feel that the rules make sense since by the time a topic is taught, the student has an awareness of how that topic is implemented at the levels below it. Engineering students like to understand at a deeper level how things work, rather than be required to do as they are told, cookbook fashion. They feel that they are on stronger footing when they can tie things to what they have already learned. This approach is good preparation for later courses in design, where understanding of fundamental underpinnings is essential to making the required engineering tradeoffs.

Cutting through protective layers

In today's real world, engineers are likely to use computers in systems where they have to interact at the device or pin level, for example, in systems where the computer is being used to sample data from some metering device such as a weather meter or feedback control system. It is no longer the case that non-computer engineers interact with the computer solely by simply programming it in FORTRAN or some other high level language. Consequently, the high level programming language course, where the compiler protects the student from everything ugly underneath, does not serve most students well. EECS 100 attempts to deal with the device registers on their own terms.

Students work in groups

Graded material consists of two mid-terms, a final exam, six programming assignments (one in machine language, two in assembly language, and three in C), and several problem sets. 65% of the grade is based on the three in-class written exams. Students are encouraged to study in groups, and in fact, only one copy of a problem set need be turned in, with all members of a study group signing off on it. Students are also encouraged to discuss the programming assignments and work out together the initial attack of their algorithms. However, in the case of programming assignments, students are expected to do their detailed algorithm design and coding on their own.

The rationale for our approach to grading is our awareness that students can learn better in groups than if they are on their own, so we encourage them to do so. However, this group atmosphere can encourage some students to let others do their work. Therefore, we insist that the actual programs are their own work, and that the bulk of their grade comes from work where they are clearly on their own. They are encouraged to learn from each other but must demonstrate that they have mastered the material themselves.

This approach is also good preparation for the engineering world they will join after graduation, where most tough problems are addressed by a team, rather than by an individual.

Analysis of Algorithms

The final component of the course is an introduction to Analysis of Algorithms, which is a particular interest of ours. All engineering is about tradeoffs, and we consider it important to point out time/space tradeoffs as early in the curriculum as possible. In this course, the tradeoffs deal explicitly with time and space as they relate to computer algorithms, but we submit that the lessons learned here are not lost when translated to other engineering disciplines.

3. Student Reaction

Enrollment in the course was 130 students, Fall semester 1995, 165 students, Spring semester 1996, 260 students, Fall 1996, and 425 students, Spring 1997. It was made a required course for electrical engineering and computer science students Fall semester, 1996. At the end of Fall 1996, we asked students to fill out a questionaire. Although we are still working with the raw data, several inferences can already be drawn.

EECS 100 as a proseltyzing tool

Table 1 shows the number of students who changed their mind as to their intended major as a result of taking EECS 100. There has been some concern that if freshmen were allowed to take EECS 100 before being exposed to other engineering disciplines, they would all switch to computer engineering. The data suggests that this is not the case. Most students were not swayed one way or the other by the brilliant teaching performance in EECS 100. Of those who were, 30 students (of the 247 total who responded to the question) switched out of computer science and engineering, and 17 switched in. EECS incurred a net loss of 13 majors.

**Table 1. Intended Majors**

| | | **AFTER EECS 100** | | |
|---|---|---|---|---|
| | | **CS** | **CE** | **Other** |
| **BEFORE EECS 100** | **CS** | 35 | x | 16 |
| | **CE** | x | 67 | 14 |
| | **Other** | 7 | 10 | 98 |

Performance as a function of intended major

Table 2 shows the grades earned by each of three groups of majors, computer science, computer engineering, and non-computing majors. The data suggests that non-computing majors can succeed in this course.

**Table 2. Grade Distribution as a function of Intended Majors**

| | **A+,A,A-** | **B+,B,B-** | **Below B-** | **TOTAL** |
|---|---|---|---|---|
| **CS** | 7 | 12 | 22 | 41 |
| **CE** | 22 | 20 | 32 | 74 |
| **Other** | 20 | 28 | 60 | 108 |

Satisfaction with EECS 100, as opposed to a programming course

Table 3 shows the responses to two questions, broken down according to whether the student at the end of the course intended to major in CS or CE or some other major. The student was asked whether he/she was happy or unhappy that he/she took EECS 100, as opposed to an introductory programming course as is done at most universities. At Michigan, engineering freshmen have

available to them introductory courses in C and in Fortran. Second, the student was asked, if he/she had it to do all over again, whether he/she would take EECS 100 or a course in C. The results show fairly convincingly that the students were happy they took EECS 100 and would do so again, even with benefit of hindsight.

Enrollment in EECS 100 Winter 1997 (Spring semester to everyone else) seems to bear this out. More than 400 are enrolled. There is no substitute for word-of-mouth when it comes to marketing a course among undergraduates. Students in all engineering majors seem to be voting with their feet as far as EECS 100 is concerned.

**Table 3. Satisfaction with EECS 100**

| | CS/CE | Other | TOTAL |
|---|---|---|---|
| **Happy** | 99 | 71 | 170 |
| **Unhappy** | 17 | 11 | 28 |
| **100** | 88 | 63 | 151 |
| **C** | 28 | 19 | 46 |

University of Michigan Course Evaluation Results

The Center for Learning and Teaching at the University of Michigan administers anonymous course evaluations by all students in all courses each semester. Two data points from the Fall 96 evaluations are interesting. Students were presented with a set of statements, along with 5 boxes, in which they were to respond to whether they strongly agree with the statement (5), agree, are neutral, disagree, or strongly disagree (1). The responses were then averaged for all students in the course.

Statement 1: "I learned a great deal from this course." Students in EECS 100 responded as follows:

Strongly agree: 78
Agree: 53
Neutral 12
Disagree: 7
Strongly disagree: 2

The computed average is: 4.30.

The university also reports the following summary information for all courses evaluated:

75% of all courses answered this question with score above 3.92. 50% of all courses answered this question with score above 4.18. 25% of all courses answered this question with score above 4.58.

Statement 2: "The amount of work required was appropriate for the credits received." The real meaning of this statement (near as I can tell, after eight years at Michigan) is: "The amount of work required was not too much for the credits received." Students in EECS 100 responded as follows:

Strongly agree: 26
Agree: 53
Neutral 23
Disagree: 24
Strongly disagree: 31

The computed average is: 3.12.

The university also reports the following summary information for this question:

75% of all courses answered this question with score above 3.63. 50% of all courses answered this question with score above 4.00. 25% of all courses answered this question with score above 4.21.

Taken together, the two questions suggest that students felt they learned a lot, but were worked far too hard, given the number of credits received. That coupled with the results of our survey that 86% were happy they took EECS 100, gives us some cause for gratification.

4. Final thoughts.

Because we strongly believed that the bottom-up approach is the correct way to introduce our own students to computer science and computer engineering, we developed EECS 100. What we didn't pay attention to is the fact that the bottom-up approach, which carries with it understanding at all levels, is the correct approach for all engineering majors. Engineering is the Science of Tradeoffs, and one can't make tradeoffs if one doesn't know the underlying principles at work. All the anecdotal feedback we have received thus far supports the view that the bottom-up approach is the preferred mechanism for introducing computing. We are particularly delighted that the anecdotal data supports the view that non-computer engineering majors can do as well and benefit as greatly from this approach as our own undergraduate majors.

However, we wish to point out that the evaluation of EECS 100 is just beginning. We have the raw data collected at the end of Fall 1996, and we expect to generate more in subsequent semesters. We will analyze it better as we better understand what questions to ask.

Further information on the course can be obtained from the web page for the course (http://www.engin.umich.edu/class/eecs100). This is a working web page that currently enrolled students regularly access for information about EECS 100. A textbook is in preparation and should be available at least in manuscript form by Fall semester 1997. A prelimary draft is in use this semester. Finally, we welcome and encourage dialogue on the points raised in this paper. We can be reached at patt@eecs.umich.edu and kjc@eecs.umich.edu.

Acknowledgement

EECS 100 is succeeding because a lot of people have put a lot of positive energy into it. We have benefited from the commitment of two outstanding graduate students, Sanjay Patel and Matt Postiff. Sanjay came back to Michigan to pursue a PhD in computer architecture, but insisted that he could not **not** be involved with this new course. The result is that the course is much better for his insights into the material and into what students have problems learning. Matt Postiff has written all iterations of the LC-2 Simulator, and has had to suffer countless hours of interaction with us, in large part because both us will not be satisfied until we get it "right"! The result is that the LC-2 Simulator is a very important positive component of the course. We have been fortunate to have very dedicated TAs, including Dave Cybulski, Peter Kim, Rob Chappell, Paul Racunas, Vinodha Ramasamy, Dave Armstrong, Aaron Wagner and Steve Maciejewski, and we are grateful for that. Finally, the support and insights of our colleagues, in particular, Dave Kieras and Ann Ford, has added a great deal to the course.

Reference

1. Yale N. Patt and Kevin J. Compton, “Introduction to Computing -- The Correct (Bottom-up) Approach,” these proceedings.