Int. J. Engng Ed. Vol. 11, Nos 4 and 5, pp. 390–398, 1995
Printed in Great Britain.

0949-149X/91 $3.00+0.00
© 1995 TEMPUS Publications.

Computer Literacy for Non-Majors: Design and Implementation Issues for Depth and Breadth

JERRY WAXMAN
CUNY, Department of Computer Science, Queens College, 65–30 Kissena Blvd, Flushing, NY 11367, USA
THERESA AUSTIN
School of Education, University of Massachusetts, Amherst, MA 01003, USA

Over the past few years, computer literacy courses have become mandatory at many colleges and universities, involving more and more departments across different disciplines. At Queens College, with the support of the National Science Foundation (NSF) and Funds for the Improvement of Post Secondary Education (FIPSE), we have been developing and implementing a new laboratory-based approach to this type of course. Three ideas—teaching students how to learn software; imparting computational ideas to non-computer science majors; and learning in a hands-on-integrated laboratory—form the basis of the new curriculum. This paper outlines our approach—its philosophy, its implementation, and our experience with it over the past two years.

EDUCATIONAL SUMMARY

1. The paper describes new laboratory concepts and curricular design for the instruction of computer sciences to non-majors.
2. The paper describes new equipment useful in providing individual, small group and whole class instruction. It also presents a combination of approaches that help make instruction more meaningful to students, e.g. hands-on instruction, laboratory exercise book with a variety of question types, individual projects to apply the concepts learned to new software, and lecture-demonstrations.
3. The course is conceived as a breadth course for undergraduate non-majors.
4. The idea that students should learn how to approach software on their own and the notion of discerning the underlying metaphors in given software packages are two concepts which promote greater learner autonomy and critical use, rather than blind consumerism.
5. Engineers are called upon to master many different computational environments during their careers. By incorporating the principles in (4) above, the course in computing for engineers would be much more useful than current courses which just teach a specific programming language.
6. A list of references accompanies the text.
7. All concepts have been tested in the classroom. Preliminary experience with this approach has indicated significant improvement over more standard approaches both in terms of the depth and the breadth of the material learned. As the project develops, we are gathering data on how various types of student populations progress in our new curriculum and laboratory. With these data, we expect not only to be able to determine the impact of the curriculum but also 'how much for whom'. The results will allow us to help make our instruction more learner-centered and may develop a framework that can be profitably adopted at other similar institutions.
8. The undergraduate engineering major will benefit from instruction which shows how to use new software by understanding the models which underlie these applications.

INTRODUCTION

IN THIS paper we present the motivation for, and some interim results of, a project aimed at developing a new approach to teaching the computer literacy/software tools course to non-computer science majors. This approach depends both on a laboratory facility, which was funded by the National Science Foundation (NSF) Instrumentation and Laboratory Improvement Program and the development of a new curriculum, which is being sponsored by Funds for the Improvement of Post Secondary Education (FIPSE). We are now at the end of our second year of a three-year project.

Since computational skills are becoming necessary in more and more disciplines, computer literacy courses have become mandatory at many colleges and universities over the past few years.

Given the perceived centrality of the skills being taught, it is surprising that not much public discussion has been focused on determining an appropriate set of topics to be covered. Consequently, the courses that are typically offered suffer some basic limitations.

One problem is that in many of these courses for non-majors, the focus has been on the introduction of low-level computer skills, primarily software applications. However, new software is being introduced at an ever-increasing rate, and any course that concentrates solely on teaching the mastery of specific packages will not provide the skills needed to master future software that students will surely encounter.

Furthermore, since these literacy courses are conceived as 'service' courses, the content rarely involves concepts basic to computing. Before the advent of the microcomputer, non-majors who wished to learn about computing either took the introductory course for computer science majors, or were offered special sections of computer courses which emphasized such topics as algorithm development, programming and system design. As evidenced by a quick scan of the texts used for the non-major courses, these topics have, for the most part, been excluded. These are important topics even for those students who do not intend to major in computer science. Mastery of these concepts is important to the non-computer science student for pragmatic reasons; it allows them to be much more sophisticated and effective users of software tools. In addition, though, an understanding of these ideas has become an important component of a general broad education; computational 'models' are becoming pervasive. The 'tools course', or course that teaches specific computer software, as currently constituted does not provide the necessary level of content generalizeability to help students transfer and apply their understanding to other applications not introduced in the class.

In addition, since students learn more effectively when practice is integrated with theory, it is important that there be a laboratory component tightly coupled to the concepts presented in the course.

The course that we are developing has been designed to deal with the three limitations outlined above. Three ideas—*teaching students how to learn software*; *imparting computational ideas to non-computer science majors*; and *using a hands-on-integrated laboratory component*—formed the basis of the new curriculum that we have developed and are currently implementing.

This paper reports on our experiences with our approach over the past two years. The next section will briefly review the underlying principles upon which our approach is based. For a fuller treatment of the first year's activities, please refer to Waxman [1].

BACKGROUND

Former approaches

The approach that will be described in the following paragraphs developed after a series of trials with other curricular designs. These consisted of lecture formats coupled with specialized sections for the physical and social sciences applications, a small programming component using Basic, then Pascal, and a 'problem-solving' recitation section.

We found that, while many students who completed this course had developed a rudimentary understanding of the issues involved in computation, this understanding was not very deep. The primary cause of this phenomenon was the paucity of meaningful hands-on experience. The course was severely limited by the fact that only 15 out of the given 45 contact hours for the course could be devoted to actual computer instruction. Additionally, it became apparent that, even had they learned more Pascal, their need for *practical* computer skills would not have been met. The students were non-majors who, generally, would have neither the opportunity nor the need to write Pascal programs. Considering the vagaries of the operating system, the general unpreparedness of the students and the small percentage of class time devoted to programming, many students did not profit optimally from this exposure to programming.

After experimenting with a number of different course formats, we began to offer a 'software tools' oriented course similar in content and level to those offered at other institutions around the country. We switched the format of our introductory non-majors course to one presenting 'software tools' on an experimental basis in the fall of 1988.

Although our course provided practical tools for non-majors, as we experienced teaching the course, we realized that there were a number of areas that needed to be addressed to make it more academically sound and at the same time more relevant for the non-major's actual needs. Again, one of these was the issue of 'depth'. A significant number of four-year schools have resisted implementing similar courses precisely because of this problem. They have felt that, as the course is currently conceived, and as is evidenced by the texts currently available for it, it lacks sufficient academic rigor. Consequently, many have either relegated it to their continuing education programs or have refused to offer it at all. It was our view that this approach was mistaken. The course covered important material relevant to the students in their own areas of endeavor and could impact how they approached their jobs once the students joined the workforce. In addition, this course at Queens College is used to fulfil a college breadth requirement in 'scientific methodology and quantitative reasoning'. Rather than throwing the proverbial baby out with the bathwater, we felt that the course should be taught but that it needed to be redesigned and upgraded. The question was, in essence, what could be done about the lack of depth in the course as it was then constituted.

Planning for change

As we began to search for options, we had to keep in mind the particular logistical constraints of our institution. The first had to do with the size of the student population and the pool of available personnel.

Since a large number of students enrol in this course, we did not have sufficient staff to teach the course in small sections. Consequently, it was taught in a large lecture (250–300 students) linked to a small recitation format (about 30 students). The students were required to attend two lectures and one recitation a week. The propose of the recitation class was to clarify and provide practice with the concepts taught in the lecture.

Quite often the instructors for these recitation sections are graduate students in computer science and the quality of their teaching varies widely. Some are excellent and quite professional; others are not quite as good. In addition, since many of our graduate students are foreign born, the quality of their oral communication sometimes leaves much to be desired. This is not a situation unique to us at Queens College. Many colleges and universities find themselves in a similar situation. Due to these contextual features we found that the recitation class was not as effective as it might be.

Another constraint was the availability of space and hardware. Since the college's microcomputer lab is not exclusively set aside for use by the computer science department but is available to the general college population, the recitation sections were offered in regular classrooms with no computational facilities available. This turned out to be a major problem given our special population of the non-computer science majors. Their lack of familiarity with computers and the problems in the lab setting were major impediments to their effectively using the facilities and doing the exercises. Even though all the concepts were fully explained and illustrated on a computer during the lecture, having seen the demonstration was not sufficiently helpful when it came to actually doing the assignments in the laboratory.

Finally, it became increasingly clear that the students needed considerable 'hand holding' while they worked on their lab assignments. In the lecture, they saw demonstrations of the concepts and various features of the packages and in the recitation classes these topics were reinforced. But when they were actually at the machines, many of them were at a loss. Due to budgetary and staff problems, however, we were limited in the number of tutors we could make available during the free laboratory hours.

Our efforts were thus directed towards addressing four areas of concern in an integrated manner: redesigning and upgrading the curriculum; improving the quality of the recitation; upgrading the laboratory facilities; and providing the support that novice learners clearly required.

Changes to curriculum and instruction

In our redesigned course, we introduced programming concepts within the context of the tools that the students were using. In addition, the nature and function of the recitation has been completely transformed. It is no longer merely a review of the material presented in the lecture but rather provides an environment where the students work out specific exercises under the guidance of the recitation instructor. The laboratory facility that we constructed is what make the new arrangements possible. The roles of the recitation instructors have changed from that of lecturer, a role for which they might not currently be well suited, to that of a supportive coach or tutor. They gain by being placed in a situation more closely matched to their skills and, consequently, their students did as well. This reorientation is made possible in large measure by the laboratory facility that we have installed.

In the next section we describe the lab facility. Following that, we will describe the changes that we have made in the curriculum. Finally, we present some of our preliminary findings as to the efficacy of our approach.

THE LABORATORY FACILITIES

A proposal was submitted to the NSF Instrumentation and Laboratory Improvement Program in 1989 for the construction of an instructional laboratory for use by the recitation sections. The lab and its use has been phased in since the fall 1992 semester.

The computer hardware in the laboratory consists of 30 networked 486/50 student workstations, a 486 network server running Novell Advanced Netware, and an instructor's workstation. Each student station is a DEC PC with 4 Mbytes of RAM, an 80 Mbyte hard disk, SVGA monitor and mouse. Five LaserJet III printers and print spooling hardware are shared by the stations, six stations to a printer.

To enhance instruction in the laboratory, a device called the Tech Commander was installed. This operates in parallel with the network to link the instructor's machine to each student's station and to allow the instructor to see and interact with each student's machine from the master console. An interface box at the student's workstation connects the computer to the Tech Commander network. The Commander, manufactured by Tech Electronics of Atlanta Georgia, is an image and keyboard switching system and is completely external to the computer. Because of the Commander's well-designed architecture/features, no modifications needed to be made to the existing hardware nor did the device interface take up any slot on the PC. The instructor can, by entering a code on the Tech Commander control console, view any station's screen and take control of that station's keyboard. The instructors can also broadcast their screen to an individual student's station or to a group of students. Similarly, from the console any student's screen can be broadcast to any group of machines. Another attractive feature is that the Tech Commander supports two-way communications via a headset so that the interaction between an individual student and the instructor can be one-to-one and private without interfering with the rest of the class activities.

A ceiling-mounted RGB high-resolution projector is connected to the Tech Commander control. This can project any workstation screen image onto a large screen in the front of the lab. The obvious benefit of this addition is the ability to facilitate whole class demonstrations and discussion.

Thus, fluid transition between whole group instruction, small group instruction and individual instruction are all accommodated when the Tech commander is used skilfully.

This facility, which has been dedicated to the course, has been a significant impact on the students' ability to absorb and retain material. Because of this, we can spend less time in the recitation sessions teaching the mechanics of the tools and provide more time for actually manipulating the tools for the individuals' own interests and needs. Further discussion of the laboratory facilities is found in [2].

The recitation, however, is only one aspect of the new course. Equally important is the shift in the focus of the lectures that form the core of the curriculum.

THE CURRICULUM

This part of the project has proven to be the most challenging. The curriculum has undergone a number of changes from its original conception in 1988. We will discuss each in turn in the following paragraphs in terms of content, delivery and impact.

Content

From the beginning, our curriculum has had three principal components:

- Learning models of computer software
- Learning computational models
- Learning principles of programming

We set out to teach these concepts through examination of particular details of several selected packages. Many students, for example, will need to know how to use computerized spreadsheets in their own areas, be it economic analysis, accounting or management. All students will need some type of wordprocessor to help in the preparation of their written assignments. Consequently, the following material was selected:

- An operating system command environment (e.g. Microsoft DOS, Windows)
- A wordprocessing system
- A spreadsheet package and model building
- A database package
- A graphics package
- Various utility programs

While we felt obligated to teach specific packages for both pragmatic and pedagogic reasons, we also were obligated to give the students the computer skills that will prepare them for careers into the *next* century. For this reason, in addition to teaching the students *what* the package models and how to work with it, we teach them how to extend it and 'customize' it. We show them how to use these 'software tools' to *construct new tools of their own*. For example, when we teach DOS we teach the students how to code sophisticated batch files, complete with parameter passing. Similarly, when we teach a wordprocessor we do it with an eye to automating and customizing it via macros; the same with the spreadsheet package. In the database portion of the course (we currently use dBASE III Plus), though a help feature is available (ASSIST), we teach the students to work dBASE from the dot prompt. After this they are taught how to *program* in the dBASE environment. This provides a natural extension of the other environments (DOS, wordprocessor and spreadsheet) in which they will have learned to 'program'.

Secondly, we concentrate on the idea that the various software packages that the students are working with have an underlying 'model' that each implements. For example, underlying a particular wordprocessing program like WordPerfect is the 'model' of a blank page of paper, part of a larger 'document' on which the user composes text. One may define various parameters with respect to the 'paper', such as its margins and the orientation of the text on the page. One may also electronically 'search' for text in the 'document' in various ways. Teaching the students to look for the underlying model in the software they use will ultimately make it much easier *for them to master new software on their own* because it teaches them to anticipate the functions of a software package. This ties their previous knowledge of the world, i.e. the model, to a computer application. In other words, this generalized concept helps students to use the knowledge and experiences they already have to discover ways in which the computer program has instantiated the model. This will undoubtedly empower students to confidently explore new software. What we see students being able to do is inductively identify a model, generalize the features that minimally must be present, test out their hypotheses to find out if indeed these features exist, and refine what they originally thought in light of their discovery process. Thus, by both experience and model building, they 'learn' the package. These are the critical thinking skills that will enable them to generalize from the software learned in our course to other courses that they pursue on their own.

We do not view our role in this course as limited to teaching the students 'skills'. A 'computational paradigm' has emerged over the last 40 years in which computer hardware and/or software is used as a metaphor for explaining such diverse phenomena as DNA and 'the mind'. Very simply explained, this paradigm entails using algorithms to define relationships and interactions that can 'model' real-life phenomena. Knowing the strengths and weaknesses of these computational models will help in 'debugging' programs or 'debunking' untenable predictions. If our students are to participate intelligently in the emerging culture, they need to have a clear understanding of the extension of metaphors based on computational concepts. This is truly possibly only by having hands-on experience with computers and an understanding of the paradigms that are used.

By introducing the concept of 'models' as the focus, this course distinguishes itself from the other 'applications courses' offered to non-majors around the USA. From a pragmatic point of view, learning the packages from this perspective simply gives the students *much more power to generalize appropriately*. They carry out operations that they would not have otherwise been able to carry out; they can solve problems that they would not otherwise have been able to solve. No matter in which discipline the students find themselves, their contribution surely will be that much greater.

INSTRUCTION

We have found by experience that maximal learning takes place when the lectures were strongly coupled to a hands-on multimedia lab experience. As we implement this philosophy, our approach has become more intensely learner-centered than in previous years. As we develop the curriculum, we struggle with designing a variety of appropriate activities to be carried out by learner.

Learners are presented with a global picture of the concepts, then work on discrete exercises that form this picture. It is our belief that working from both top-down concepts and bottom-up gives students both an understanding of the whole and its relation to the parts. This is accomplished through a guided discovery of the topics in the course. Thus, students will be enabled, once the course is completed, to explore on their own in a systematic and productive way. Our efforts for the past two years have been to determine just how much exploration and just how much guidance is needed for a variety of learning styles.

Aside from a commercially available text detailing the use of the packages, the course material consists of lecture notes and a laboratory manual. The lecture notes provide a printed version of the overhead transparencies used in the lecture. The lab manual presents a set of 28 'experiments' or guided explorations that enable the students to gain experience and confidence in mastering new software environments. The laboratory notes have been twice revised and will be extensively reworked this summer.

In the following paragraphs, we discuss how the use of the hardware and evaluation of our efforts have facilitated more effective instruction.

EVALUATION AND ASSESSMENT

Year 1

Laboratory facilities and logistics. Although the laboratory facility for the recitation sections was to have been completed for use during the fall 1991 semester, severe financial constraints at the college caused us to defer its implementation to the summer of 1992 for use during the fall 1992 semester. While this set back our schedule by two semesters, it did have some positive consequences. As a result of the delay, we were able to take advantage of technological and price breakthroughs that were unavailable when funding was initially granted for the purchase of the hardware.

In the interim, the existing facilities were strained by the increased numbers of students enrolling into the course and the budgetary hardships inflicted by a cut in state financial support. These factors lead to plans to make a more effective use of the FIPSE-funded student aides and the NSF-funded laboratory. Originally we intended to use the new laboratory strictly as a classroom for the recitation sections. However, by confronting our problem with staffing, we realized that by utilizing the facility as an 'open laboratory' during those times when classes were not scheduled, we could make use of its unique architecture to multiply the effectiveness of the student aides.

During the first year the student aides were deployed in the college PC laboratory and circulated among four rooms to help students on demand. At times, students were not even aware that these aides were available, since they were stationed in a different part of the facility. Assigning the student aides to the new laboratory and making it an 'open' facility for the students to work on their assignments did help solve this problem.

Curriculum design. Because of the time lag in equipping the laboratory, in the first year of implementation, the experimental curriculum was implemented without the equipment. This meant that students attended a lecture and recitation sections and still worked in the computer laboratory outside of class. While not the optimum conditions for introducing the new curriculum, it did allow us to evaluate the initial impact of the new curriculum independently from the impact of the laboratory to modify those parts of the curriculum that did not depend on the laboratory, and test for the effects of those changes separately.

During the first grant year, in collaboration with the recitation instructors, we mapped out a lecture-by-lecture syllabus delineating the topics to be covered in each class and recitation session.

Impact. Ongoing evaluation of the curriculum is carried out by collecting three sources of data: classroom observations, student questionnaires and student achievement on examinations. While classroom observations by the lecturer, recitation instructors and laboratory aides provided qualitative data on the effect of the new curriculum from the instructors' perspective, the student questionnaires and exam scores completed the picture with the students' perspective at two times each semester—the beginning and end. A number of the classes were taped (both video and audio) and the tapes were reviewed for lesson's depth, breadth and clarity of presentation.

The questionnaires that we designed provide us with important information as to the characteristics of our student population, their background, reasons for taking the course, prior exposure to computers, and their reaction to the course and its contents. These were distributed, collected and coded twice during each of the two semesters in the first year.

Finally, the student performance on similar examinations from semester-to-semester and year-to-year provide us with evidence of levels of mastery of the topics in the course.

Though a fuller analysis of these data is scheduled for this summer, the information that we have gleaned from the first year significantly influenced the structure of the second year's course. We found, for example, that, as hypothesized, the course has had a positive impact on the willingness of students to take additional computationally oriented courses *as a result* of taking this course. Of those students for whom this course represented their first introduction to computers, a full 25% expressed interest in additional math/computer science courses directly as a result of this course. We also found, somewhat to our surprise, that for those students for whom this course was their first introduction to computers, the female students expressed slightly greater confidence in the material learned than did their male counterparts. We say 'somewhat to our surprise' since the prevailing wisdom has it that at college level, women tend to be less confident about their mathematical ability than men.

Another important result of our ongoing evaluation was the detection of a problem both in performance and in student confidence in the DOS segment of the course. As a consequence of the evaluations in fall 1991, we found that the first topic covered, DOS, was causing significant difficulty for the students. This became evident to us via the multiple measures of exam scores and student self-reports. The scores on both the laboratory and final examinations for this topic were the lowest among all the software topics. In addition, the self-evaluations of level of confidence in understanding the material were the lowest for DOS on surveys administered both at mid and end semester. These findings were confirmed by the lecturer while observing some of the students taking laboratory examinations in the spring 1992 semester, and by noting how the students tried to solve the exam problems on the computer. Through these observations it was verified that the students' mastery of DOS was, in fact, very shaky.

We hypothesized that there were three probable factors contributing to this situation. The first factor seemed to be that DOS was the first 'sofware' the students became familiar with, and for many the first experience on the computer. It was possible that their lack of performance was due to the newness of the environment and the strangeness of the concepts. Secondly, DOS does not represent a particular 'application' as is the case with a spreadsheet or a wordprocessor. Rather, it gives them the tools for controlling various aspects of the PC environment—such activities as copying files and creating directories and subdirectories. The problem seemed to be that since the students have no experience with 'files' that have been created by other programs, they have little intuition about how (or why) these files are to be copied. Thirdly, it appeared that it is not only the abstractness of the DOS environment that was causing the problems; it was the 'doing' of the commands as well. Since the students had not yet internalized a model of DOS, they did not see the overall picture of how commands were used and interrelated. They had significantly more difficulty figuring out what to do with DOS than in other software environments.

We decided to divide the DOS material into more manageable units; we presented the preliminary topics near the beginning of the course and moved the more advanced topics, including directories and batch files, after the coverage of WordPerfect. With this change, we noticed a significant improvement in the student's understanding. This will be detailed below.

In addition, we added an additional required recitation (now supervised laboratory) hour to the course. This was very positively received by the students and resulted in notable improvement in student performance.

Year 2

The laboratory facilities and logistics. The laboratory was operational during the first month of the fall semester at the beginning of the second year. We had 30 networked 486 computers with six students to a printer. The only setback was that the Tech Commander unit, which would have allowed the recitation instructor access to the student station from the instructor's console, was not operational until the middle of the semester. This was caused by scheduling problems of the various craft unions on campus. We compensated for this, however, by having the instructors move from station to station, and having the students in the laboratory work together in informal 'cooperative teams' to help each other out.

Problems that we experienced during this second year due to a shortage of student tutors was partially alleviated by additional funding (both internal and external sources which are discussed later). The problem of crowding in the general PC laboratory was eased by the fact that we added an additional instructor-led recitation hour, and many students were able to complete a significant amount of the work there and hence spent less time in the general laboratory.

Curriculum design. While the course remains as a lecture/recitation format, two structural changes have been made: a sequencing of the DOS lessons in the fall 1992 and spring 1993 semester curricula, and an increase in the number of laboratory hours. In addition, a complete manual of exercises was created to be used in the laboratory recitation session. Student performance was monitored for change.

In the fall 1992 semester, DOS was covered in four chapters in consecutive laboratory sessions, three through six. The intent was to determine whether or not having explicit exercises with the concepts of DOS would significantly affect student performance. While we found some improvement—the students' laboratory grades went from an average of 78 to 81—their self-confidence levels did not change.

Though virtually all available textbooks, including our current manual, cover the DOS material all at once, both our experience and observation indicated that this approach has been only slightly helpful for the students. For this reason, during the second semester of this second year, we attempted further modifications by dividing the presentation of DOS topics. The more elementary topics were introduced at the beginning of the course and the more advanced material was presented after the students had experience with a software package. With the new sequence of introducing basic DOS concepts first, such as the idea of an operating system, disk and file commands and more advanced DOS topics such as directories and batch files after WordPerfect, we found a significant increase in the students' DOS laboratory scores. Whereas in the spring 1992 semester the average was 78 on the DOS laboratory exam, in spring 1993 the average was 85. Thus, it seems that when less of the abstract material was presented at the beginning, the students had time to develop their internal model in the context of a more 'concrete' program such as a wordprocessor.

Another significant issue, that of getting over the initial 'doing' hurdle, was partially alleviated during the fall 1992 semester when the laboratory became operational. Since the recitation instructors were able to offer immediate 'hands-on help', students were able to master the pure 'technique' with greater ease. In additon, we discovered, not much to our surprise, that many of the poorer students were spending as little as one hour per week working on laboratory assignments. This amount of time was truly insufficient to complete all the exercises upon which they would be graded. this was our first red flag that perhaps not all students were actually completing the exercises; rather some were copying the material.

Impact. To respond to the inadequate amount of time, we were able to add an additional hour of required recitation (laboratory) time to the course. This has had a significant impact on both the students' impression of their mastery of the material and their actual performance. During the spring 1992 semester (the second semester of the first year), when the recitations were in a regular classroom, there was no instructor-supervised laboratory. The average grade for the first practical (DOS and WordPerfect) was 79. During the spring 1993 semester, the first semester in which the laboratory was available for the full course, with two recitation sessions per week, the average on that same exam was 85. Both semesters' exam assessed the same material and were administered in the same fashion.

In trying to explain these results, we noted the changes that have occurred from spring 1991 to spring 1993, in which a manual was introduced and extra weekly laboratory session was included. The additional laboratory session provided students with more hands-on-computer time working under supervision instead of the past practice of allowing students to use this same time to work independently. Since each laboratory examination dealt with material that was explicitly covered in a recitation instructor-lead class, the increase in score is not attributable to more time being spent on the same work. We suspect that even though, during fall 1992, the students had been expected to complete the laboratory work from the manual on their own, many did not in fact do so. In the spring 1993 semester, with the obligatory additional laboratory contact hour, the students were in effect forced to spend an additional hour, which helped them assimilate the concepts and solidify their understanding of the underlying model.

By the end of the first grant year, the course material consisted of a published student manual containing about 175 pages of transparencies that are used in the lecture, and handouts on assignments and projects prepared by the recitation instructors. Based on the syllabus we have outlined in the fall of 1991, work was begun in spring 1992 on the design of laboratory manuals which were to be used in the recitation classes for fall 1992. This was carried out by Dr Waxman in collaboration with one of the recitation instructors and with students who had completed the course. A preliminary version of some of the materials was completed by the end of May 1992 and was used during a special section of the course (to be more fully described below which was taught by Dr Waxman in June 1992.

During the summer semester of 1992, we evaluated the effectiveness of the preliminary laboratory material. Before the beginning of the fall semester, the manual was reworked and greatly expanded into a 150-page laboratory manual with 28 different labortory assignments. The model that was used was that of a laboratory manual in one of the sciences such as biology or chemistry. An initial section described the laboratory objectives and the rest of the laboratory guided the student through a series of 'explorations' of the software and its capabilities. It would start with simple problems and gradually involve the student in more complex problems. The manual did not tell the students what to expect. It was deliberately designed to be open-ended.

The manual was used in the fall 1992 semester and, in a somewhat modified version, during spring 1993. The students' assignments in the manual were graded three times during each semester: once for DOS and WordPerfect, once for Lotus 1-2-3 and, once for dBASE. The recitation instructors reported that the stronger students responded quite favourably to the desing of the manual. However, the weaker ones were put off by its open-endedness. Consequently, we plan to revise the manual and partition each laboratory into two sections. The first will directly present the elementary material. Students will be told what to do and what outcomes they should expect. The second section will introduce the more advanced material utilizing the exploratory model but providing more 'scaffolding' to help the weaker student.

In order to collect the student data, we began an automated data collection process. The protocol has been programmed in Foxpro (a dBASE dialect) and the students fill in the questinnaire online. This allows much more time for us to code the open-ended student responses and do data analysis. The questionnaire has now been revised and distributed for the fourth semester.

ADDITIONAL INFLUENCES

Two outside factors have also influenced our project's development. There are the increase in support staff and the extension of the curriculum to service a distinct population.

Increased student support

Although it is not easy to assess the extent to which availability of additional individualized help has nfluenced the students' learning, we believe that it must be acknowledged. While FIPSE is providing the salaries of a number of laboratory assistants for this tutoring, we sought other funds to increase the available outside-of-class help and to broaden the pool of student advisors to include undergraduates and particularly, minority students. This was made possible because Queens College was recently awarded a Ford Foundation Multicultural Development Award. The faculty was invited to present proposals to compete for funding from this award in the form of semester-long support for an appropriate project. At that time, a request for a grant to augment the FIPSE funds was submitted. These new funds would be used for hiring laboratory assistants from a specially selected group of undergraduates to serve as interns. Fortunately, we were awarded this grant and were able to fund four additional undergraduate student tutors.

Currently, the interns write accounts of the types of student problems that surface while doing the laboratory assignments. They have received orientation and subsequent training to compile their field notes. These data will be used to analyze patterns of student strengths and weaknesses with the course materials. In reviewing these data, the interns discuss alternative ways of handlng the difficulties that students have in carrying out the assignments. Hence they are being prepared to take on supportive instructional roles while they learn about different student learning styles.

Expansion of the reach of the curriculum

Queens College has an increasing number of older students returning to complete degrees, and has a special progrtam, Adult Continuing Education (ACE) to accommodate them. One of the courses offered int his program is a software 'tools' course. Last June, the Principal Investigator of the grant taught this course to see if the curriculum, as it had evolved thus far, could meet the needs of this group as well as those of more traditional undergraduates. Nationally, more and more older adults are returning to the campus. It was hoped that teaching ACE course would enable the PI to gain some perspective on the unique needs of this group. This course was offered in the summer preceding the second year of the project.

We found that while we could maintain the same level of material with the older students, the amount of both lecture and laboratory time required for them to learn it increased by about one-third. This group is an excellent one to work with from a perspective of curriculum development. Due to their maturity and rich life experiences, they are serious and articulate. They are quite vocal in expressing, in no uncertain terms, what is working in the course and what is not. The course was very successful. About 30 students registered for it. Their response was so positive that when the ACE program offered it again this summer, 100 students registered during the first afternoon. Registration has been closed at slightly over 100.

We have continued to teach the course during the summer following the second year of the project, utilizing the curricular materials developed over the year. The additional feedback on the utility of these materials will permit us to compare the results from two years of teaching this special population.

CONCLUSION

We have described a new approach to teaching computation to non-computer science majors. It is based on the ideas of teching students how to master new software on their own, that of imparting computational ideas to non-computer science majors, and that of incorporating a hands-on-integrated laboratory component with adequate support for whole class and individual instruction. Preliminary experience with this approach has indicated significant improvement over more standard approaches both in terms of the depth and the breadth of the material learned. As the project develops, we are gathering data on how various types of student populations progress inour new curriculum and laboratory. With these data, we expect not only to be able to determine the impact of the curriculum but also 'how much and for whom'. The results will allow us to help make our instruction more learner-centered and may develop a framework that can be profitably adopted at other similar institutions.

Acknowledgements—The authors would like to gratefully acknowledge the generous support of the National Science Foundation—Instructional Laboratory Instrumentation Program, grant no. USE 905-1014 and the Fund for the Improvement of Post Secondary Education, United States Department of Education, no. P116-B10157, whose funding made possible the work reported on in this paper.

REFERENCES

1. J. Waxman, Model-based software instruction. *Technical Report 91-007*, Queens College, Department of Computer Science (1991).
2. J. Waxman, An enriched laboratory-based non-majors introduction to computers. *Creativity: Proceedings of the Annual Conference of the American Society for Engineering Education* (1992).