AC 2008-277: CUSTOMER BASED COURSE DEVELOPMENT – CREATING A FIRST YEAR PROGRAMMING COURSE FOR ENGINEERS AND SCIENTISTS

Patrick Jarvis, University of St. Thomas

Patrick L. Jarvis received his J.D. in Law and Ph.D. in Computer Science both at the University of Minnesota in Minneapolis. He has broad industry and consulting experience in the design and development of procedural and object-oriented systems, relational database systems, peer-to-peer and client-server systems, as well as the management of high technology employees. His law practice focuses on arbitration and mediation of high technology disputes. He joined the Computer and Information Sciences faculty of the University of St. Thomas in 1992. Recently he has taught object-oriented programming and design, database design, and data structures.

Jeff Jalkio, University of St. Thomas

Jeff Jalkio received his PhD in Electrical Engineering from the University of Minnesota and worked for thirteen years in industry in the fields of optical sensor design and process control. In 1984, he co-founded CyberOptics Corporation, where he led engineering efforts as Vice President of Research. In 1997 he returned to academia, joining the engineering faculty of the University of St. Thomas where he teaches courses in digital electronics, computing, electromagnetic fields, controls, and design.

Marty Johnston, University of St. Thomas

Marty Johnston received his Ph.D. in Physics from the University of California, Riverside working in atomic physics. After serving as a Post-Doctorial Researcher at the University of Nebraska he came to the University of St. Thomas in 1995 to initiate an undergraduate research program in physics. When he is not in the laboratory, Marty can be found teaching classical physics, electricity and magnetism, theoretical mechanics or experimental methods.

Christopher Greene, University of St. Thomas

Chris Greene received his Ph.D. in Electrical Engineering from the Massachusetts Institute of Technology (MIT) and proceeded to a 25 year career in industry. At Honeywell, he did research on adaptive control and navigation systems before becoming Program Manager for several large aerospace programs. At Horton and Nexen, he was responsible for the development of industrial control products. In 2002, Dr. Greene joined the engineering department at the University of St. Thomas where he currently teaches classes in signals and systems, controls and digital design.

Mari Heltne, University of St. Thomas

Mari Heltne received her Ph.D. in Management Information Systems from the University of Arizona, working in the areas of expert systems and group technologies. After serving on the faculty and administration of Luther College in Decorah, Iowa, for 24 years, she became the assistant dean of the Honors College at the University of Arizona. In 2002, she joined the faculty at the University of St. Thomas, where she now is chair of the Department of Computer and Information Sciences, teaching courses in Systems Analysis and Design. Her research interest in group systems continues, and she has recently worked collaboratively with a faculty member in Journalism on ethical issues in web based technologies.

Customer Based Course Developments – Creating a First Year Programming Course for Engineers and Scientists

Abstract

One of the difficulties encountered in developing an introductory programming course is that it is often expected to serve many competing purposes. An introductory course might be intended to convey fundamental concepts in computing or fundamental computing constructs such as looping and conditional execution. The course might provide practical knowledge such as the syntax of a particular programming language or the use of a computational package commonly used in a specific application area. The course might provide an opportunity to develop cognitive abilities in incoming students such as general problem solving skills and good study habits. Finally, first year courses are often expected to include socialization aspects to improve student retention rates and aid in the transition from high school to college. Often different departments requiring such an introductory course may have differing needs, further reducing the likelihood of a successful outcome. With all these disparate objectives, it is not surprising that such introductory courses often fail to meet the expectations of the instructor, the students, or the departments requiring the course

In this paper, we discuss a process used to successfully develop such a course. We discuss how the need for the new course was originally identified as a result of program assessments and how the requirements for the course were developed cooperatively between computer science faculty and faculty in physics and engineering. We discuss how funding was obtained for course development and present results from the first semester of course offerings.

Background

Ten years ago, our institution created a common introductory programming course for all students who needed programming as part of their major field. Prior to this time, several introductory programming courses were offered, aimed at students majoring in business, computer science, and the physical sciences. This diversity of introductory courses, each with its own unique focus and using a different programming language, led to difficulties when students continued on in advanced courses. The new common introductory course, taught using the Java language, sought to alleviate these difficulties by providing a common language and a common set of learning outcomes for all students. Unfortunately, by 2004, assessment data from follow-on courses in engineering indicated that the course was not meeting all of its intended objectives. This data was supported by student surveys on program objectives and anecdotal information from faculty in physics, computer science, mathematics, and engineering. Discussions between faculty in computer science, physics, and engineering over the following year led to the conclusions that (1) a new introductory programming course was needed to meet the needs of engineering and the physical sciences and that (2) traditional curriculum review and informal communication between departments was insufficient for a timely and successful redesign.

Requirements Analysis

The first step in the design of a new course was to understand the requirements for this course and how the current course fell short of meeting these requirements. Through discussions with

faculty from physics and engineering, it was determined that students taking the existing course were unable to apply what they had learned regarding programming to either learning new programming languages, or using Java to solve disciplinary problems in follow-on classes. Furthermore, students did not understand why they needed to take the programming class or how it fit into their curriculum. It was determined that the underlying objective of the course was to provide the students with an understanding of how to develop an algorithm to solve a problem rather than to program in a particular language.

Course Design

In the year leading up to the first offering of the new course, representatives from computer science, physics, and engineering met several times to develop a design for the new course. The course would be offered to first semester freshmen as part of their socialization into college life and as an introduction to the reasoning skills required by engineering and the sciences. The course structure would include lectures on Tuesdays and Thursdays, programming labs on Mondays and Wednesdays, and open labs on Fridays for students to work on assignments Examples from engineering disciplines would be used to motivate the techniques taught. As textbooks, we chose to use Brian W. Kernighan and Dennis M. Ritchie's, **The C Programming Language** (Prentice Hall, 1988. ISBN 0-13-110362-8) and Danital T. Kaplan's, **Introduction to Scientific Computation and Programming**, (Thomsons Brooks/Cole, 2004. ISBN 0-534-38913-9). A programming language, C, and an application package, Matlab, would be used in parallel so that students could see that underlying concepts were language independent and develop a sense of the difference between language syntax and fundamental control structures. The actual first offering of the course was quite different.

The lecture/lab structure was too rigid and was abandoned early on. The material was presented in a "just in time" fashion so when a lecture was needed, it was needed *now,* despite the day of the week. A similar situation developed for lab time. If the students were having difficulty with a concept, more lab time was often the answer and presenting new material through a lecture was not. This more fluid approach worked well and neither affected our learning objectives nor reduced the amount of material covered in the course, although it did require some additional and unanticipated planning.

We were not able to present C and Matlab in parallel. Although we have not given up on the idea, we have not been able to reduce the complexity of this approach to a point that gives us sufficient confidence that the students' learning would not suffer. The best we were able to do during the semester was a small amount of alternation between the topics such as, develop this solution in C, know develop the solution in Matlab.

Observations on the Initial Offering of the Course

Part I - Socialization and Reasoning

The students were placed into groups of four or five and assigned reasoning problems. The goal was to build a logical path of reasoning that led from a set of facts to a conclusion about some missing facts. For example, the students were asked how an automobile uses gasoline and were asked to list everything they knew about a car – nothing happens when it is turned off; when it is running it makes noise, vibrates, gives off heat, produces gases, etc. As the facts were

accumulated, the groups were asked to give an answer to the question of gasoline use that was consistent with their list of facts. What the instructor was trying to do here was to show the students that they already are problem solvers and that problem solving is a reasoning process using logic. Meeting in groups forced them to socialize a bit and get more comfortable in the classroom. Another process that was begun and continued through the semester was that of having a student present the solution to the class as a whole. This was done to help get them comfortable speaking in front of groups and presenting a topic, albeit short, in a clear and understandable way. Before each student would present, the instructor would give them advice such as "face the audience", "speak clearly and not too quickly", "use the board to illustrate your answer", etc.

Some problems were designed to give the students a chance to offer alternative solutions. For example: "*Assume that you live in a city. When you take a shower, it is easy to see that water is provided to your house under pressure. How does this happen? Can you suggest an alternative way to provide water to your house under pressure? What are the tradeoffs between the two methods*?" This allowed discussion of water towers and their locations, pumps, windmills, cisterns, etc. The goal here was to reawaken their creative abilities. As the instructor told them: "*You were all creative when you were kids. Every time you sat down and were handed a crayon, you drew something interesting. It's time to get that creativity back.*"

A third set of problems was designed to show tradeoffs in design solutions and that problems are solved for a reason; they aren't solved in a vacuum. For example, they were given this assignment:

a.) Why do public restrooms have hand drying devices? That is, what is the purpose of hand drying devices in a restroom? (It is not sufficient to answer "to dry hands")?
b.) Give an example of a purely (or mostly) electrical solution to the hand drying problem and an example of a purely (or mostly) mechanical solution.
c.) For each solution, list the direct costs of that solution (costs that are directly attributable to having it perform its function of drying hands).
d.) For each solution, list some indirect costs (costs that are incurred, but not directly attributable to having it perform its function of drying hands).
e.) For each solution, name one or more events that could cause the solution to no longer be desirable.
f.) For each solution, list some possible unexpected outcomes of using that solution. An unexpected outcome is an event that occurs but was not anticipated when the solution choice was made. Not all unexpected outcomes are bad.
g.) Using the above information, write a recommendation for using one or the other of the solutions keeping in mind the purpose you listed in part *a*.

Another goal in this set of problems was to make them aware that almost every solution has side effects – some good, some bad, some quite unexpected.

<u>Part II Algorithms</u>

Many problems, including computer based problems, require solutions that are organized into steps. Having solved some general problems, the idea of a series of steps as a solution to a problem was introduced. To illustrate how algorithms can be sometimes difficult to specify even

when the solution is well-known, a pair of students was selected and placed back-to-back. One gave the other instructions on how to tie a shoe. The instructions had to be doable and unambiguous. As the students quickly learned, even when the solution to a problem is well understood, it is sometimes difficult to express the solution in elementary steps.

The class then built on the idea of the kind of steps needed in an algorithm by writing instructions for an imaginary S robot. The S robot was defined to be able to pick up one playing card from a deck and hold it in its hand. It could make some determination about the card such as its suit or value and it could either put the card back on the top of the deck or discard it to the floor. The students were again placed in groups, given some playing cards, and assigned problems to solve of the nature: *count the number of cards in the deck*. New instructions where added to the robot's repertoire as new problems required them (e.g., count the number of face cards in the deck) but the basic pick up from the top, put back on the top was not changed. It was in this manner that counters, conditionals, and loops were introduced. The syntax of the commands was agreed upon by the students and the only requirements were that the commands must be elementary and non-ambiguous. The students were presented with problems that could not be solved using a single S robot and asked to find the minimum number of such machines that were required to solve the problem. Then Q robots were introduced – pick from front of deck, put down at end of deck. More complicated problems were assigned and the groups presented a variety of solutions. Not all of these solutions worked and this allowed the presentation of the ideas of an infinite loop or an undesirable situation such as the robot attempting to take a card from a deck that has no cards remaining in it.

Problems were posed that required solution by S machines alone or Q machines alone or S and Q machines together. Since Q machine solutions often required the counting of the cards in the deck to avoid infinite loop problems, the idea of a function call was introduced as a shorthand device for writing the loop(s) necessary to count the cards.

Algorithm specification for S and Q problems was originally done using pseudo-code but as the students progressed through the problems, flow charts were introduced as an alternative method of specification. This was done to prepare students for programming problems and to help them see that the same algorithm could be specified in a variety of ways.

Part III - C Programming and Algorithms

Once algorithms had been introduced, it was time to begin writing programs. It was hoped the students would see the similarity between commands for the S and Q robots and statements in a C program. The class started with "Hello World". C syntax requirements presented an opportunity to discuss context free and context sensitive grammars (such as human languages). Once variables had been introduced, the instructor spent some time talking about the five parts of the computer and what was really happening inside the machine when the program was being run. The functions of the ALU and the organization of memory provided a convenient detour into base two representation and arithmetic. Conditionals and loops were introduced and flowchart representations of the algorithms were presented. The instructor stressed repeatedly over the course of the semester that the algorithm is developed first and the program is simply a translation of the algorithm into statements of whatever programming language you are required to use.

Since the course focused on "concepts over constructs", only one method of doing something was introduced at this point in the semester. The "if" and "while" were used exclusively until arrays were introduced. It was only then that the "for" loop was studied. The students were encouraged to only write code they understood – "*write plain, vanilla code*". Program efficiency was down played and program clarity was emphasized. Functions were introduced early in the semester but pointers were discussed in detail only when arrays were introduced. Sample programs would be followed using values drawn into a simulated memory so that students could more easily see how pointers were used and how the values changed.

It was not intended that this part of the semester train the students in C. C was always placed in context with other programming languages and important programming concepts were emphasized: for example, actual and formal parameters and step-wise refinement.

C is not a particularly friendly language to beginning programmers and it was some time before input could be introduced since it required the address (&) operator and a discussion of pointers. Single dimension arrays were introduced and the students wrote functions for searching, finding maximum and minimum, etc. This concluded the first part of C programming.

Programming is a matter of practice, so many labs were given in which the student needed to write functions to implement small functionality. One lab required the students to develop a function that would test whether a number passed to it was prime or not prime. This was then developed into a small program that would generate the k_{th} prime. Timing was added and this gave the students a feel for how many computations were being done by this algorithm. Ever few days over a several week period the class returned to this program and talked about how the algorithm might be improved. Each change was timed. After several of these sessions the improved algorithm produced a result in less than three seconds that had required over 4.5 hours by the original algorithm.

Part IV – Matlab

By this time, the students (at least those who were still trying) were comfortable with developing algorithms and converting them into C statements. Some were great at it and it was often the ones who had never programmed before. Matlab was a complete departure from what they had been doing.

As in the programming section, students first had to copy the instructor's code as they were introduced to Matlab. But they were quickly required to start figuring Matlab functionality out for themselves by "playing" with the statements. Most books on Matlab that have the word "engineer" in the title are not very helpful for freshmen engineers. The problems in the book assume a knowledge of engineering that freshmen simple don't have. While the book was useful as a reference and explanation of command syntax and function, it was necessary to develop labs and assignments whose problems were familiar to them but required the Matlab tools that would be useful to them as engineers.

Matlab is quite different than C and writing Matlab functions, while they have the flavor of C, are more unstructured. For example, functions in Matlab can return more than one value.

Functions, scripts, and *ad hoc* commands were covered. Function writing was emphasized but the danger was that the students would simply translate C solutions line by line into Matlab. To ensure that the students were exposed to the Matlab's different approach to problem solving, the instructor designed a problem with constraints that would require them to investigate Matlab more carefully. The students had earlier finished a C assignment that implemented the dice game called Craps and then augmented it to simulate the playing of *n* games. They were now given the same assignment in Matlab that included user input with error checking, required producing graphs, and had the restriction that no conditional statements ("if" or "if"-like) could be used. This required them to both explore the Matlab functionality and also think of alternative ways to test variables.

While Matlab had some important concepts, it was mostly a vocational exercise and an open-book skills test was given to evaluate their understanding of the package.

Part V – Return to C

The final part of the semester was spent introducing C file input and *structs*. The sequential file processing algorithm was introduced as well as file and record handling. This also provided an opportunity to talk about hard disk organization and CDs.

Summary

At the end of the semester, the faculty team that initially designed the course met again to review the initial offering and to determine what improvements might be made in future offerings of the course. While we can't judge the ultimate success of the new course until we can observe the students' performance in follow-on classes, we can make some observations about what went well and what went poorly during this initial offering.

What Went Poorly

1. Some students simply copied their assignments from others. It is easy to delude yourself that you really understand material but until you try to do the assignment, you really don't know. Some students succumbed to the temptation of having others do their work and learned little. When exams were given, it was too late. About 15% of the class had either late semester withdrawals or below C grades. The instructor met individually with each student and went over his or her midterm exam to explain the areas of weakness and told them specific things they could do to improve. Students had the choice of dropping their midterm exam and using the final exam as their only exam points. Of the seven people who were failing at midterm, three later dropped the course, three failed the course, and one received a D-.

2. All the students can read but not all choose to do so. Assignments with detailed specifications frequently were more than one page long. Many students never got beyond the first page.

3. More graded homework is necessary. Freshmen cannot be relied upon to be responsible enough to do assignments if they are not required to hand them in. For some students, an assignment is only an assignment if it is turned in and graded. Ungraded labs where frequently looked upon as optional and were often never completed and, in some cases, never started. This despite the fact that the lecture following the lab was based on the work done in the lab.

Exhortations that the labs must be finished or the lecture would make little sense were ignored by many. First semester freshmen bring with them to college whatever high school experience they had in the areas of independent work, discipline, and responsibility. These three must be well-developed in a student in order to succeed in college level engineering and science courses. Our students came from a variety of schools from small farm town high schools to prestigious college preparatory schools. The perceived quality of their high school seemed to have little influence on the student's willingness to complete ungraded labs. Five graded assignments were made during the semester. Future offerings of this course should require a minimum of one per week.

4. The idea that people teach themselves was foreign, troubling, and annoying to some of the students. They resented being asked to "find out how this works". Education at the high school level did not seem to require them to learn independently. Increasingly, colleges are fighting the voyeur student: one who prefers observation over participation. For this type of student, group work is the least effective teaching learning environment. They do little or nothing in the group and learn little or nothing in the process. These students want to be told how to do something with little desire to discover how to do it themselves.

5. We do not currently have an effective peer mentoring program for this course. We need a place where students can get help outside of class and normal office hours.

What Went Well

The socialization aspect of the course was successful. Students developed confidence in presenting their ideas to classmates and by the end of the semester most had developed the study skills needed for success in their college classes. Many freshmen in the course commented to their advisors that this was the hardest course they had ever taken but that they had really enjoyed it.

Many of the students left the class with a good beginner's command of problem solving and algorithm development. Some became quite good at programming in C.

Once the more fluid approach to the scheduling of lectures and labs had been started, the weekly course structure worked well. In particular, the open Friday lab was a great success. Students who needed help could use this time to get help from the instructor. In addition this time was used by many students as an informal class day. Student groups would come to the lab and work together on the material, often answering one another's questions.

It appears that we achieved all of our goals with the exception of teaching Matlab and C concurrently. We look forward to tracking student performance in future semesters to see if our new course has actually overcome the deficiencies of the course it replaced.