The Software Studio: A Transitional Course for Those Entering the Field of Information Science

James J. Alpigini, John S. Mullin
Penn State Great Valley School of Graduate Professional Studies

Abstract

The Master of Science in Information Science degree program at the Penn State Great Valley School of Graduate Professional Studies attracts students from a diverse set of backgrounds. For those candidates with non-technical degrees, a need has been identified for a transitional course, namely a software studio which goes beyond traditional professional development offerings. This paper discusses the design of and the experience gained in conducting the software studio. The software studio is designed to enable a student with a non-technical background to make the transition to the study of information science at a graduate level. Since its introduction in the spring of 1999, this course has been offered during each spring, summer, and fall semester. As this is a relatively new course, it is undergoing continuous incremental improvements based on in-class experience, student feedback and changes in the industry.

I. Introduction

As information technology becomes increasingly important to daily life, the demand for professionals with formal education in software engineering and information systems continues to rise. In response to this need, the popularity of Information Science, (IS) programs like the one offered at the Penn State Great Valley (PSGV), School of Graduate Professional Studies continues to increase.

The Master of Science in Information Science (MSIS) is a relatively new and exciting degree program at the Penn State Great Valley School of Graduate Professional Studies. The MSIS program offers a synergistic blend of computer engineering, software engineering and management courses, which emphasize a balance of information science and management theories and thus enable the development of a students technical competence, leadership skills, and business expertise.

While many students interested in this degree program have the academic potential to succeed, often their undergraduate background and work experience has not prepared them for studies in such a technical discipline. It is necessary then, to provide these students with a course that bridges this gap in their background, enabling them to succeed in the MSIS program and eventually the industry as a whole.

In order to design a course that aids in this transition, it is necessary to have a basic understanding of the backgrounds that these students possess. The focus of this paper is the adult graduate student with a non-technical undergraduate degree. Typical candidates for the MSIS degree program have from 5 to 20 years of experience in a professional environment. While many of these candidates have had some exposure to information systems, their goal is to either expand their current capabilities and responsibilities or to transition their careers into an information systems area.

At PSGV it has been observed that people desiring to make this type of career transition have often relied on the continuing education. The primary goal of these continuing education courses is to augment the skill set of students already involved in the industry. While these professional development courses are invaluable to people in the field, they do not provide the needed depth of understanding for people making the transition to the information science field. To address this problem, PSGV has developed a 3 credit, graduate software studio.

Upon successful complete of this course, students will have the necessary skill set to overcome the problems encountered in information science and software engineering. To achieve this goal, they must gain an understanding of a basic software design method, as well as, an ability to write programs using a modern programming language. This goal also requires the studio itself to adapt. At the conclusion of each semester, student input is solicited and reviewed, and the presented lecture material is modified as needed to further improve the course.

This paper is organized as follows. First, a description of the decisions made during the design of this course and the rationale used to make them is presented. Next, an overview of the course material is presented, noting the obstacles faced by the students and the contributions that their diversity in background brings to the course. The paper then concludes with a discussion of the lessons learned and future plans for this course.

II. Course Design Decisions

Assuming that the students who would take this course have no background in computer or information science, two objectives presented themselves immediately. At the conclusion of this course, these students should be familiar with a basic design methodology and they should be able to apply this methodology to design and implementation of programs using a “modern” computer language. In addition to identifying the objectives for this course, it was important to recognize the limits of the material that could be presented. While a very high level discussion of the organization of a computing systems would be necessary, detailed discussions of computer architectures and operating systems are beyond the scope of the material taught in this course.

Although obvious to many professionals, it is important to state the reason for integrating the use of methodology and programming language skills. People, even those with knowledge of design processes, often launch into the coding of a solution before they have a thorough understanding of the problem. This results from the misconceptions that software is free or that the incremental cost of corrections appears to be low. As is the case with any design problem, the cost of corrections increases significantly as a design moves from conception to delivery and production use. By integrating the use of a design methodology with the instruction of a computer language, this course encourages students to develop good design habits from the outset, rather than hoping they unlearn bad habits in the future.

While a number of models for the design and life cycle of software systems are typically taught in an introductory software engineering course, this course focuses on a single basic design methodology. The first decision made was the selection of the design methodology that would be incorporated in throughout the course. The Waterfall Design Methodology was selected with the rationale that it serves a basis for the Incremental, Spiral, and RAD methodologies. Furthermore, an attempt to introduce a more complex methodology would obscure the importance of the basic steps that are clearly defined in the Waterfall method.

Selection of the programming language to was the next choice made. Visual Basic, PASCAL, ADA, C / C++, and Java were among the languages under consideration. The criteria for the selection of the language included support for "modern" constructs such as those needed for object oriented design, commercial acceptance, availability of text books and supplemental materials, and availability of compilers for on-campus and home use. The selection of the language did not imply that one language was superior to the others in an overall sense, only that it met the criteria and suited the needs of the program of study to a greater degree. Using this basis for the decision, the C subset of C++ was selected as the language to be taught. While some might argue a more apt introduction would have been an object oriented language such as the full C++ implementation, an understanding of basic structural coding is necessary to support method development for an object oriented language. It should be noted that some of the failings of C were considered to be of equal importance to its strengths. In addition to the criteria already stated, the influence of C in defining other languages, e.g., Java and Perl, was also noted. Having selected the C subset of C++, it was also recognized that an emphasis on a "good" coding style would be necessary.

Many of the courses taught at PSUGV are presented in a 7 week format, allowing students to take 2 courses in succession during a single semester. However, the studio was scheduled as a 14 week course in order to allow students more time to assimilate the material.

The text assigned for this course includes a copy of a commercially available compiler, allowing students to work at home, as well as, on campus. This has the additional benefit of providing a consistent environment for program development among the students taking the class.

Because this is a transitional course, the emphasis was placed on a hands-on approach rather than theoretical presentation. During the 14 weeks of course, there is one class period each week. These class periods are divided into a lecture and lab session. Ideally the lab session is at least half of the period.

Although a midterm and final examination are included in the grading scheme, as a hands-on course, the grading relied more heavily on the successful completion of design and programming assignments. Five programming assignments of increasing difficulty are required, followed by two design assignments. In the design assignments, students are required to prepare a design report that indicates how they utilized the Waterfall design methodology to implement a solution to the problem described in the assignment.

Students are given 1 week to complete the first program, a "Hello World" variant. This first program is not graded. Beginning with programming assignment 2, students are asked to work with a partner. The partners are given 2 weeks to complete programming assignments 2 through 5. For the design projects the partners are given 4 weeks from assignment to completion. While a shared report is expected for the first design report, the report for the final project must be completed on an individual basis.

III. Course Overview

In this section a chronological overview of the course is presented, as shown in Table 1. While this is not an exhaustive review, the major hurtles that students encounter are described with approaches used to help the students overcome them.

Table 1: Course Outline

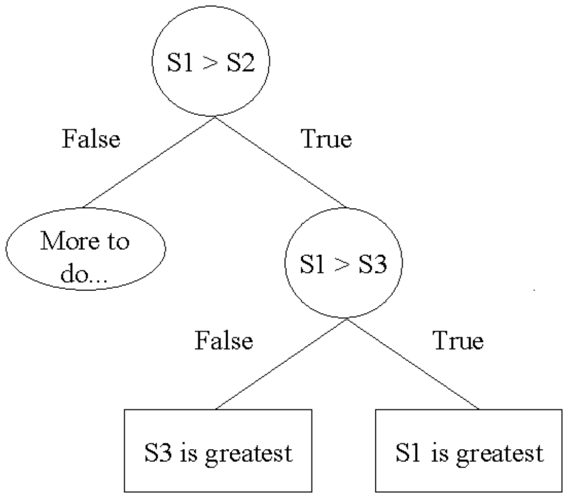
Lecture	Topics
1	Introduction: Methodology Overview, Programming Environment, & Hello World
2	Basic C syntax: data types for numbers and arithmetic operators
3	Boolean logic and the use of the "if" statement
4	Looping constructs: "while" and "do..while", Using the Waterfall Methodology
5	The "switch" statement, File IO
6	Review of topics to date
7	An introduction to functions, Variable addresses
8	Automatic memory, variable scope, and addresses as parameters to functions
9	Dynamic memory and C structures
10	Using dynamic memory : An introduction to linked list - adding elements, Waterfall: Requirements
11	Linked list continued - deleting elements, Waterfall: Analysis and design
12	Project implementation issues, Parameter passing revisited, Waterfall: Test

After stating the goals of the course, presenting the syllabus, and reviewing the grading policy, the first lecture begins with a basic description of a computer system and its operating environment. At this point the Waterfall design methodology, which will be pursued throughout the course, is introduced. While a detailed presentation of the requirements, analysis, design, implementation, and test stages of the design method is not given at this point, the overview of the methodology provides a firm foundation for future discussions. Once this has been done, the students are introduced to C and the programming environment. During this portion of the lecture, students are asked to identify programs that they have used. The compilation process is then described and a step by step or cookbook description of their first programming assignment is presented. A slight variation to the "Hello World" program, typically included in programming courses, is their first assignment. In this variation, the students are required to have the program prompt them for their name and then print "Hello <their name>", so that they are immediately exposed to both input and output operations. The primary purpose of this assignment is to introduce the programming environment, allowing the students to clear the first hurtle: program entry, compilation, and execution.

The next two lectures emphasize is the basic grammar of C. These lectures include the basic data types and mathematical operations, as well as, a discussion of keyboard and monitor IO. The programming assignments for these lectures are fairly simple, e.g., read 3 numbers from the keyboard and calculate the sum. With the introduction of Boolean logic, the use of the "if" statement, and statement blocks in the 3rd lecture, students are asked to request 3 string values from the user and determine which one is the greatest alphabetically. As may be expected, this leads to the next hurtle for the students to overcome. The issue here is not the syntax of C, rather it is the logic needed to solve the problem. To tackle this issue, the students are returned to the

Waterfall methodology. While the requirements seem clear, it becomes apparent that further analysis is needed before pursuing the solution to this problem. At this point the students are asked to consider the states of the data within in the program. As an aid to solving this problem, the use of decision tree is introduced. After partially completing the decision tree as shown in figure 1, the code necessary to implement the completed portion of the decision tree is discussed. Use of statement blocks for the true and false conditions of the "if" often becomes a second obstacle in the completion of this program. In the class period following this assignment, a review of the use of comments and indenting styles, first discussed with the introduction the "if" statement, is often sufficient to overcome the problems encountered. For those groups still struggling, individual assistance is provided during the lab portion of the class period.

Figure 1: A partial implementation of a decision tree for finding the string with the greatest value, given strings: S1, S2, and S3.



The definition and use of arrays, the while, and do..while statements are covered in lecture four. As an introduction to the corresponding assignment, the focus returns to the use of the Waterfall model. The initial problem statement is that the program must request the temperatures for a week from the user, determine the average, the high and the low temperature, and the day on which the high and low occurred. Before moving to the analysis stage the students are asked if they are satisfied with the statement of requirements. Typically, the students erroneously indicate that they are in fact satisfied. In one case a student with significant marketing experience, indicated that she was not happy with requirement detail. After she was asked to hold her questions, the remaining students indicated that they thought requirements were sufficient. During the analysis stage that followed, the student with the marketing experience was invited to ask her fellow students her questions. At this point the students were asked 'what types of data they think are necessary?' and 'how many of each will be needed?'. Obviously, they did not have enough information.

In order to reinforce the need for adequate requirements, a scenario where a developer does a "seat of the pants estimate for completion" of this problem based on inadequate information is described. In this scenario the developer's initial estimate of 3 weeks was reported to his boss. After the discovery of the problem during analysis, the developer contacts the customer for a follow-up discussion of the requirements. The customer tells the developer that a meeting can be scheduled 2 weeks from now. Based on this scenario, a discussion of the importance of a complete set of requirements and the iterative nature of the design process ensues. Following this discussion, the usage of a context diagram and state diagrams within the analysis stage of the Waterfall methodology is introduced. The transition to the design stage, using the knowledge gained during analysis, is then demonstrated. In the design stage a structural decomposition diagram is used to specify the organization of the program that will be constructed during implementation. At this point in the course, there has been no discussion of the use of functions other than main(..). A skeletal solution is presented for the implementation of this program. This skeleton provides a framework that groups the statements used to perform different portions of the functionality together using comment blocks to delineate the sections. Use of the Waterfall method concludes with a discussion of the testing phase. During this discussion the distinction between verification and validation is emphasized. A spreadsheet or tabular format is defined for use in documenting the tests they will perform at a system level.

Lecture five covers the usage of switch statements and file IO. Beginning with the first assignment, the students have been using the gets(..) and printf(..) statements to perform simple IO. The use of files is described as the process of opening files, performing an input or output operation using fgets(..) or fprintf(..), and then closing the file. The existence of other file IO functions is mentioned at this point, however, the students are encouraged to "stick to the basics". Although this was expected to be a difficult topic, experience has demonstrated the transition to file IO to be easy for the students who rely on the simple fgets and fprintf functions.

During the next class period, all material to date is reviewed.

Functions are introduced in the lecture seven. In C, parameters are passed by value and a single value can be returned. As the lecture begins, the use of various functions already encountered is reviewed. Simple examples are then presented to demonstrate parameter passing. During the discussion of this topic, the use of the position of parameters to align the arguments passed from the calling routine to the function is emphasized. A functional implementation of the program assigned in lecture 4, where the students were required to find the high and low temperatures and the day on which they occurred, is used to demonstrate the use of functions. Within this demonstration, the use of the structural decomposition developed in the design stage of the Waterfall model is highlighted.

A question is then posed concerning the limitations on functions within C, where all arguments are passed by value and only one value can be returned. Given the material covered thus far, it would be impossible to construct a function that could determine more than one value. Returning to the temperature example, a function could not be written to determine the range of temperatures, i.e., the high and low. At this point, the use of addresses is introduced as a means of passing the location of a variable to a function, allowing the value stored in the location to be updated. The use of an entry in a card catalog and the corresponding book on the shelf in a library as an analogy for the addresses and their de-referenced value has proven to be extremely useful in conveying the use of addresses within C.

In the class period following the introduction of functions, the concepts of automatic memory and variable scope are discussed. The use of addresses as parameters to a function are then examined in greater detail. At this point the treatment of array variables without their indices is discussed. The algorithm for the bubble sort and an implementation for its use with an array of strings is then developed in class.

At the conclusion of lecture eight, the students are given their first design assignment. In this assignment, they are to present the user with a menu that will allow the user to read a list of up to 10 names from a file, sort the names, print the names to the screen, and read a file containing a second list of names reporting any names that appeared in the first list. In addition to the program, the students are required to submit a report describing the activities and information gained during each stage of the Waterfall method.

Mechanisms for dynamic memory allocation and the construction of record or structure variables in C, are presented during the lecture nine. In lecture ten, the construction of a linked list is used to illustrate the use of dynamic memory and structures. At this point the final design project is assigned. For this project the students are required to construct a system to manage a mailing list. A menu interface is to be provided, allowing the user to add labels, delete labels, print them to the screen, store them to a file, or reload them from a file. Although the program is to be developed in conjunction with their partner, each student is expected to submit a report detailing the actions taken and information gained during their use of each step in the Waterfall method to design this system.

In the remaining class periods of the course, implementation issues for the linked list are discussed during the first half of the lecture portion of class period. During the balance of the lecture period, the information for each section of the report is discussed in detail. Particular attention is devoted to testing objectives, approaches, and limitations. Within the test report format, the students are expected to have columns for the test step goals, inputs, expected results, actual results, and actions taken to correct problems as shown in Table 2.

Table 2: A test report example, where Window's Explorer is the unit under test.

Goal	**Contol (input)**	**Output**	**Predicted Results**	**Action**
Test Ability to start the program	Click on Start, navigate to Programs->Window's Explorer	*As expected*	A list of directories on the left and files on the right.	*None Required*

IV. Lessons Learned

As of this writing, 6 sections of studio have been run, with approximately 30 students in each section. Even with this limited experience a number of lessons have been learned through observation and student feedback.

First and foremost, this is a course where no question is "too dumb", but some may be too detailed. For example, one student asked about the information contained within the FILE structure created when a file is opened. For this type of detailed question, it is best to defer the answer to the lab session following the lecture portion of the class. This allows interested students to have their answer without confusing those who may be struggling with basic concepts.

Although only one analogy, the card catalog and books, was mentioned in the last section, physical examples should be used to clarify concepts wherever possible. When workable analogies have been discovered, they have been incorporated into the lecture material for the future.

As the difficulty of the program assignments increases, many students attempt to code the entire solution before compiling for the first time. In simple terms, they attempt to eat the whole pie in one bite. When they do attempt to compile the program, they are then often overwhelmed by the number of errors reported. To resolve this problem, the students are instructed to make a copy of their source. They can then remove portions of the code until they get down to a reasonable subset that they can correct. They are then told to gradually add the additional functionality back in, making corrections as they go.

When this course was first offered, each student was required to complete all assignments individually. During the lab portion of the class period, ad hoc groups formed to discuss approaches to the completing the assignments. Discussions with the students who participated in these ad hoc groups revealed a better understanding of both design methodology and programming concepts. A partner or buddy system for the completion of programming assignments has now been formalized. Although this course is intended as a transitional course, students with limited programming experience have attended. In implementing the partner approach, care must be taken to ensure that both partners participate. To this end the partners are encouraged to alternate programming and testing roles.

V. CONCLUSION

Due to the fact that the studio is a relatively new course, there is obviously room for refinement. Given the nature of the software engineering and the constant changes in technology, the expectation is that this course will be constantly evolving.

The best measure of success for the studio is the success of students in the courses they take after the studio. While the data at this point is limited, the indications are that the studio has been able to provide the necessary foundation for subsequent course work. As more information becomes available refinements will be made in the studio to better ensure the success of the students making the transition into this field.

JAMES J. ALPIGINI
James J. Alpigini is an Assistant Professor of Systems Engineering at the Penn State Great Valley School of Graduate Professional Studies. In addition to research, he teaches in the areas of computer architecture, computer security, numerical analysis and mechatronics. He received a B.E.E. degree from Villanova University in 1982, a M.Eng.E.S. degree from the Pennsylvania State University in 1993 and a Ph.D. from the Engineering Faculty at the University of Wales, Swansea in 1999.

JOHN S. MULLIN
John S. Mullin is a Lecturer in Software Engineering at the Penn State Great Valley School of Graduate Professional Studies. In addition to the Software Studio, John teaches in the areas of project management and E-Commerce. He received a B.E.E. degree from Villanova University in 1980, a M.Eng.E.S. degree from the Pennsylvania State University in 1998.