(2)

Broad Agency Announcement Number 91-18

Final Rpt. MDA 972-92-J-1035

A Practical Software Engineering Curriculum: A Pilot Study

Arun P. Gupta

Weibin Zhu

Kimberly R. Brown

Calvin Ezell

Division of Computer and Applied Sciences

Georgia Southwestern College

Americus, Georgia 31709-4693

Phone: (912) 928-1393

Email: gupta@gswaixn3.gsw.peachnet.edu

A Practical Software Engineering Curriculum: A Pilot Study

ABSTRACT

This project identifies a problem in Software Engineering instruction and proposes some approaches to overcome it. It is advocated that a software engineer must have at least one formal course in Software Engineering. Further, a course containing merely the concepts is not sufficient; additional course(s) complementing the formal course are required. A streamlined sequence of courses was developed that would help achieve the goals of Software Engineering. This report presents the course sequence and the observations for the project. It is hoped that other institutions can benefit from this study.

I. INTRODUCTION

The software crisis is very real and has been very well documented. Many books and articles related to Software Engineering (e.g., [Boe83], [Boo87a], [Fai85]) report that software costs have surpassed hardware costs and are steadily increasing. Though significant contributions have resulted from research in this field, their impact has not yet been fully achieved. Some of this can be attributed to the Software Engineering education at small institutions and, to some extent, at large institutions as well. Of the 1011 US institutes and colleges offering the Computer Science curriculum (as listed in [CBB91] and by electronic mail and USENET survey), 523 offer only the Bachelors as the highest degree in Computer Science. Unfortunately, as listed in [CBB91], of the 523 institutions, only 187 have formal course(s) in Software Engineering. This indicates that a significant portion of the graduates have no formal Software Engineering course/training. It has been agreed that Software Engineering education is obtained only through "hands-on" experience, implying that one course is not sufficient for education in Software Engineering; follow-up courses are required. To this effect some adjustment in teaching Software

Engineering is proposed. This paper presents the results of the study. This research is targeted at small, teaching institutions because they constitute the majority of the existing US institutions offering the Bachelor of Science curriculum in Computer Science.

In this paper a streamlined sequence of courses is presented; it is expected to benefit students desiring to become Software Engineering professionals. Attempts are also made to (1) study the feasibility of establishing such a curriculum, (2) identify problems encountered during the research, and (3) give suggestions to other institutions who wish to establish a similar curriculum. In Section II the status of Software Engineering instruction at Georgia Southwestern College is presented. Section III provides the rationale for this research. First, the problem is described and then, some approaches to overcome it are presented. Section IV identifies the sequence of course developed to provide the formal training and the follow-up courses. Section V gives the experiences during this pilot study and also some suggestions for other institutions thinking of embarking on such a project. Finally, Section VI summarizes the experiences and presents conclusions.

II. BACKGROUND

Software development costs have rapidly increased with respect to hardware costs. Software Engineering text books can be easily found which give figures indicating that software costs have surpassed hardware costs. Thus, it is important that Software Engineering be an integral part of any Computer Science curriculum. Although significant contributions have resulted from research in the field of Software Engineering, their potential impact has not yet been fully realized. It is believed that the major problem lies in Software Engineering education; students are often taught only the principles of Software Engineering and do not get hands-on training on practicing these principles. What is required is software engineers who know the principles and also have first hand experience in how they affect software systems.

The main objective of this research was to develop a course sequence for teaching Software Engineering that provides students with the principles and practices of Software Engineering. To be called a software engineer one must have at least one formal course in the discipline. Further, a course containing merely the foundations of Software Engineering is not sufficient; courses complementing the basic course are required. To this effect, a three quarter course sequence for students aspiring to become software engineers is studied. The first course covers the principles of Software Engineering followed by two courses in which students use these principles. By actually working with the principles and learning from the results, future software engineers will have a strong foundation. GSW has adopted the Ada programming language for the later two courses since it provides features well suited for Software Engineering ([Sam86]).

The remainder of this section presents the status of Software Engineering instruction at Georgia Southwestern College (GSW). GSW is a small Liberal Arts College with an enrollment of slightly over 2500 students (it is a small institution). The Division of *Computer and Applied Sciences* was established in 1984 and offers, among others, a Bachelor of Science degree in Computer Science. In the Fall 1991 Quarter the Division had over 230 students of which over 40 were pursuing a B.S. in Computer Science. Currently, there are almost 200 students of which about 25 are enrolled in the B.S. in Computer Science program.

Computing facilities available to students in the Division are an IBM 9375 minicomputer running the VM operating system and on the network within the Division. The Division also has five IBM PS/2 file servers (Intel 80386 and 80486 based) each of which runs the IBM AIX-1.2/1.0 operating system. Each of these servers is connected to several Intel 80286 and 80386 based microcomputers via an Ethernet network. Within this environment students have access to the system resources from any microcomputer by connecting to any one of the five file servers. The stand-alone microcomputer based environment

provides students with capabilities to perform basic tasks like printing, spread-sheet processing, and word processing whereas the IBM AIX based environment provides them with enhanced capabilities. Recently, a new node was set up to run under the SCO OpenDesktop System Release 3.2 from Santa Cruz Operation, Inc. It also serves as the host for the Ada compiler from Alsys, Inc. So far this node is used only for the Ada-based Software Engineering courses.

Since 1990 the B.S. program in Computer Science has seen significant changes. Some deficiencies in the curriculum were identified using [ACM91]. Accordingly, a few new courses were added and some existing courses were modified. A significant change was the addition of a course in Software Engineering in the Summer Quarter of 1991. This course was offered for the first time in the Fall Quarter of 1991 and used C or Pascal as the programming language for small projects. Other courses added included one in Discrete Structures and two entitled Software Engineering with Ada (I and II).

III. RATIONALE

This section identifies the problem with Software Engineering instruction and proposes possible solutions. Students who lack a formal course in Software Engineering graduate entrenched with Von-Neumann type programming habits and use them in the work place. These have resulted in the current abundance of ad hoc code, strangling the industry. Thus, at least one course in Software Engineering is required for every software engineer. However, current methods often leave graduates wondering how Software Engineering principles taught in class are useful in the real world. This leads to the conclusion that there should be "something else" which complements the formal Software Engineering course.

Formal instruction in Software Engineering is generally provided as a high level course (in the junior or senior year) when many students have completed the requisites

necessary for programming and developed programming habits (good or bad) which will be hard to change. Most of these habits are then intermixed with those prevalent practices in the work place leading to chaos and adding to the software crisis. This problem is more pronounced with those not having formal training in Software Engineering. On the other hand many software engineers wonder how the Software Engineering principles taught in class are useful in the real world. They have learned the principles without learning how to practice them in real life.

The remainder of this section provides solutions to the above problems under the assumption that Software Engineering will be taught at small institutions. It also supports as to why Ada is a good choice in the Software Engineering curriculum. It is important to note that no matter which of these best suits a particular institution, certain key problems must always be addressed. These areas present themselves in the form of courses in mathematics and technical writing. Experience at GSW has shown that many students lack the necessary mathematical background to comprehend many Software Engineering techniques (e.g., logic, proof techniques, algebras). In addition, the majority of undergraduate students have not had previous exposure to reading and comprehending, much less writing, technical documents. A software engineer must be proficient in both of these areas, and therefore, curricula must be adjusted appropriately when incorporating a Software Engineering education.

Three solutions to the above problem were identified based on [For90]. The first is to offer Software Engineering very early in the curriculum (at the freshman or sophomore level). After this, students are required to utilize these principles (and get "hands-on" experience) in all courses where they are expected to develop any software system. The major drawback of this "bottom-up" approach [For90] is that beginning computer science students may not have the background necessary to fully grasp concepts taught in the course (e.g., mathematical background). In addition, organizational restrictions may prevent

teaching such a course early in the curriculum. If this approach is not feasible, then the students must, at a minimum, be taught basic Software Engineering principles (without getting into too much detail) from the outset; adherence to these principles should be rigidly enforced. This approach creates a strong foundation and it provides students with the necessary "hands-on" experience throughout their academic career. However, it is not flexible; it will be difficult for students to change into the existing computer science curriculum. It is also risky for the school, faculty, and students because it will be difficult to "bail out" if problems are encountered ([For90]).

The second solution is to offer Software Engineering in the middle of the curriculum (i.e., early in the junior year) and have students utilize the principles in advanced courses containing some form of software development (e.g., operating system, compiler design, real-time systems, etc.). This approach, however, may encounter resource difficulties; most institutions like GSW use either Pascal or C as the primary language for advanced programming courses. Further, most students opt to use Pascal for the development of much of their software. Standard Pascal prevents students from fully utilizing several key Software Engineering principles (e.g., separation of module interfaces and their implementation, independent compilation units, generics, concurrent programming). Some use C as the implementation language for such projects, but C does not adequately address all these principles. Further, C programs are often not very easy to understand and maintenance is often difficult. Unfortunately, Ada, which does not have such problems [Sam 86], is often not a feasible alternative because of its cost.

The final solution to this problem is to provide Software Engineering in the senior year followed by additional course(s) in which students utilize the principles of Software Engineering. The first course (basic Software Engineering course) should cover the principles of Software Engineering. The course material must be complemented with a thorough understanding of the problem domain for an extended software development project.

To be realistic, the requirements must be somewhat flexible. Subsequent course(s) will primarily involve completing this project. At GSW, two follow-up courses that taught the Ada programming language in conjunction with the basic Software Engineering principles were developed. Students, working in groups, used Ada as the implementation language for the project outlined and researched in the first follow-up course. This project took them through a major portion of software life-cycle and, at the same time, helped them get first-hand experience in utilizing Software Engineering concepts and principles. In addition, they were exposed to the critical aspects of working in teams and trying to deliver a "quality" product on time.

For institutions like GSW the third "top-down" approach [For90] maybe the most feasible of the three approaches to overcome the current problem with Software Engineering education. Students will not only learn the principles but get to use them in a non-trivial project. Further, they will get a feel for the real world and what they can expect in the work place when they are ready to enter the work force.

IV. CURRICULUM

This section lists the curriculum that was developed for the three-course Software Engineering curriculum. The most important result expected was students who would be more productive in the work place earlier than by current instruction methods. Students would not only know the principles but would also have used them in their project(s) and learned from their experiences. Other results expected from this project include (1) a streamlined curriculum which may be adopted by other, similar, institutions wishing to establish a Software Engineering track in their undergraduate programs, (2) determination of the feasibility of such a curriculum, and (3) identification of other courses in Software Engineering education. The feasibility of such a curriculum is discussed in section V. Discrete Structures and Technical Writing were the only courses identified as other

important courses that were feasible at GSW. This section contains the outlines developed for the three-course Software Engineering sequence.

As mentioned earlier, a Software Engineering course was already (Summer Quarter 1991) established before the start of this project. Some preliminary work with respect to the course outlines for the follow-up courses was also done. A formal outline for the follow-up courses was submitted to the College and was accepted in the Fall Quarter of 1992. The following is a summary of the outline of the courses for the complete three-course sequence offered at the senior level. The numbers listed are actual course numbers. Relevant books and/or articles were made available to students through the college library.

Software Engineering (CSC 430):

Objectives: To expose students to the problems associated with developing software system constrained by user requirements, and cost and time estimates in a contract. Students will learn basic principles and techniques used in Software Engineering. The techniques will be utilized in a term project.

Prerequisite(s): Data Structure and Algorithms (CSC 310) is required and Design of Operating Systems (CSC 421) is desired.

Contents: The following topics will be covered in this course: (1) introduction to Software Engineering, (2) qualities of good software systems, (3) Software Engineering principles, (4) software models and design techniques, (5) specification techniques, (6) verification techniques, (7) tools and environments, and (8) software reuse.

Textbook: [Ghe91]

Software Engineering with Ada - I (CSC 432):

Objectives: To teach students how to use the Ada programming language to achieve the goals of Software Engineering. Students will be introduced

to Ada with emphasis on relating Ada constructs to Software Engineering principles. The first part of a term project is required. The project will be done in teams whose size will depend on the enrollment. Students enrolling in CSC 432 are expected to enroll for CSC433.

Prerequisite(s): Software Engineering (CSC 430)

Contents: The following topics will be covered in this course: (1) overview of Software Engineering, (2) introduction to Ada, (3) software architecture, (4) data types in Ada, and (5) control flow and program structure. All Ada constructs will be related to the corresponding Software Engineering principles so that students realize how the constructs help in software development.

Textbook: [Boo87a], and [Boo87b] as reference if required.

Software Engineering with Ada - II (CSC 433):

Objectives: A continuation of CSC 432, this course will be devoted to reusability, concurrent and real-time processing, and programming in the large. Students are expected to complete the project started in CSC 432.

Prerequisite(s): Software Engineering (CSC 430) and Software Engineering with Ada - I (CSC 432)

Content: The following topics will be covered in this course: (1) review of basic Ada concepts, (2) reusability and Ada generics including input-output facilities, (3) Ada tasks, and (4) exception handling.

Textbook: [Boo87a], and [Boo87b] as reference if required.

V. EXPERIENCES

This section presents the experiences during the development of the Software Engineering course sequence. These are given with the intention that other institutions

learn from them before establishing such a curriculum. It was determined that such a course sequence is feasible as long as there is significant institutional support specially from the person responsible for allocating the hardware and the software resources. It would be nice to report that everything went well in this project but, unfortunately, this was not the case; several problems were encountered. These are: (1) small class size, (2) level of student preparedness, (3) limited computing facilities and library holdings, and (4) problems related to the allocation of hardware and software resources. These problems are discussed in this section.

A major problem in the pilot study was enrollment; the Computer Science department at GSW is relatively small. This problem is inherent to small institutions and there might be no solution to it. In the Fall Quarter of 1991 only a few students were registered for the Software Engineering course. However, enrollment increased when this course was offered during Winter Quarter 1993. This may be attributed to the fact that students had to successfully complete this course to enroll in the follow-up courses (dealing with Ada). Also, the Software Engineering course was made a required course in the Computer Science curriculum. Whether this is merely a one-time increase or not remains to be seen.

A problem related to the small department size was the loss of students in the elective courses. As mentioned earlier, the Software Engineering course (CSC 430) at GSW College is a required course but the subsequent courses are electives. It was observed that the class size for the first follow-up course, CSC 432, decreased by about 50 percent. This may create problems with respect to the project; students work in group(s) on a problem to be used in the entire course sequence. With loss of team members they have to lose some time in joining up with other teams and getting back on track. This, however, is a valuable experience for them in terms of the dynamic "real-life" environment. This problem was not encountered in the current project since a new problem was selected for the elective courses CSC 432 and CSC 433. But, another problem arose - due to personal reasons a key player

in the project withdrew from the class just four weeks before the end of the second course. This hurt the project; the loss of a team member affected, to some extent, the progress of the project and it lowered the morale of the team. The effect was more pronounced due to the small class size.

It was pointed out earlier that student needed appropriate mathematics and writing courses. A mathematics course, Discrete Structures (CSC 235), was recently added to the curriculum so most senior students did not have a chance to complete this course. Adding other courses involves departments other than the Computer Science department and this may be solved only through some administrative channels. Unless such courses get established and stabilized, students' preparedness will be lower than expected.

Concurrent processing, an important aspect of Software Engineering (and Ada), could not be covered to a great extent due to limited computing facilities. Real justice to the concurrent processing in the follow-up courses could only have been done if additional hardware support was available. Funds were not expected for developing a laboratory containing equipment to assist in concurrent software development; simulating such a system by input and output through files and standard devices was the only solution. This, naturally, did not give students the real experience in concurrent processing.

The library holdings at a small liberal arts college like GSW are very limited in the area of Computer Science. This restricts students from being current with the new ideas and techniques. Although the facilities for inter-library loans are available at GSW, the turnaround time is over a week - too long for a quarter that lasts only 10 weeks. Institutions wishing to establish a Software Engineering track should be willing to invest in some of the important journals in the field.

A significant problem encountered in this project was related to the allocation of hardware and software resources (i.e., systems related). As with many institutions,

specially small ones, facilities and funding are limited. The Computer Science department enrollment at GSW is less than 25% of total Division enrollment; this has moved the funding priority to the other departments in the Division. The purchase of the SCO OpenDesktop and Alsys Ada and the allocation of a dedicated host for this project indicated a possible improvement in the computing environment. Unfortunately, this was not the case and the cooperation received during this project was not as much as assured before the start of the project. Several problems like delayed system setup, unreliable and "unfriendly" computing environment were encountered in the project. A delayed system setup did not give people involved enough time to familiarize themselves with the SCO OpenDesktop and the Alsys Ada environment. This affected the project and often a "quick and dirty" approach was often the only way out. Also, due to the unreliable and unfriendly environment, students were often diverted from the "real" problem and the advantages offered by the Ada environment were shadowed by the unreliable and unfriendly computing environment. It is strongly felt that all problems related to the systems could have been easily overcome by significantly greater support from the person responsible for allocating hardware and software resources.

Despite the number of problems faced it is believed that students got a feel for the problems they might encounter in their work place, and, more importantly, they learned, to some extent, to overcome these problems. Although it is difficult to judge from a one-time experience, the feeling is that the objective of educating students in Software Engineering was met and those who stayed through this project until its completion were better positioned for a fruitful career in Software Engineering. It is strongly believed that this course sequence stabilize over time with better results after a few iterations and when the non-inherent problems like student preparedness, computing and library resources, and specially those related to the system are resolved.

VI. SUMMARY

A software engineer must have at least one formal course in Software Engineering. Further, there should be one or more courses that complement this formal course. To this effect, a streamlined sequence of courses to achieve the goals of Software Engineering was presented in this report. This sequence is expected to produce graduates who will be more productive software professionals. This three-course sequence was successfully established in the 1992-1993 academic year in the Computer Science department at Georgia Southwestern College. The objective of educating students in Software Engineering was met and those who went through this project are positioned to be better software engineers than they would have been otherwise. Using the Software Engineering adaptation of Bloom's taxonomy of educational objectives [Blo56] given in [For90] it is believed that students who have been through this course sequence may have reached the "synthesis" level (this is a subjective evaluation). Ada played an important role in reaching that level. It remains to be seen how the objective of this three-course sequence will be further met in the future and whether students reach the "evaluation" level in the taxonomy of [For90].

Several problems were encountered and other institutions desiring to set up such a curriculum have to design strategies to overcome them before any benefits can be derived. Some problems like those associated with small class size are inherent to small institutions. Other problems, like the level of student preparedness, can be overcome through some institutional help. Equipment problems are funding based and can be overcome by aggressively soliciting external funds so that proper laboratories can be set up. Finally, systems related problems should not be problems at all; they should be overcome long before undertaking a project of this nature.

REFERENCES

ACM91 *Computing Curricula 1991 Report of the ACM/IEEE-CS Joint Curriculum Task Force*, IEEE Computer Society Press, 1991.

Blo56 Bloom, B., *Taxonomy of Educational Objectives: Handbook I: Cognitive Domain*, David McKay, New York, 1956.

Boe83 Boehm, B., "The Hardware/Software Cost Ratio: Is It a Myth?" *IEEE Computer*, vol. 16, no. 3, March 1983.

Boo87a Booch, G., *Software Engineering with Ada, Second Edition*, The Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA, 1987.

Boo87b Booch, G., *Software Components with Ada: Structures, Tools, and Subsystems*, The Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA, 1987.

CBB91 *The College Blue Book, Degrees Offered by College and Subject*, 23rd Edition 1991.

Fai85 Fairley, R.E., *Software Engineering Concepts*, McGraw Hill Book Company, New York, NY, 1985.

For90 Ford, G., "1990 SEI Report on Undergraduate Software Engineering Education," *Technical Report, CMU/SEI-90-TR-3 ESD-TR-90-204*, March 1990.

Ghe91 Ghezzi, C., Jazayeri, M., Mandrioli, D., *Fundamentals of Software Engineering*, Prentice Hall, Inc., Englewood Cliffs, NJ, 1991.

Sam86 Samment, J.E., "Why Ada is not just another Programming Language," *Communication of the ACM*, Vol. 29, No. 8, Aug. 1986.