Int. J. Engng Ed. Vol. 20, No. 2, pp. 267–276, 2004
Printed in Great Britain.

0949-149X/91 $3.00+0.00

Necessary Metamorphoses of a Software Engineering Program*

DANIELA ROSCA, WILLIAM TEPFENHART and JAMES McDONALD
Software Engineering Department, Monmouth University, West Long Branch, NJ 07764, USA.
E-mail: drosca@monmouth.edu

We present the main lessons learned over the 16 years we have been running a graduate degree in software engineering at Monmouth University. It covers the challenges in delivering a program that meets the needs of industry and students in a highly dynamic field. The evolution of the curriculum induced by the domain's continuous advances and industry practice is presented. This evolution is an example of a transition from a 'computer science curriculum with an engineering flavor' towards a software engineering curriculum. The special meaning of continuous course content development in software engineering is argued through issues pertaining to dated textbooks and ever-changing programming languages, operating systems, and software tools. The paper also presents our experience of dealing with the diversity of the student body, and its influence on the curriculum and course content. The paper concludes with recommendations for constructing a similar program and ideas for future developments.

INTRODUCTION

ALTHOUGH SOFTWARE engineering was recognized as a distinct field in 1968 at the NATO-sponsored conference on the subject [1], it took universities and colleges a significant amount of time to respond to that fact. It was not until 1986 that Monmouth University (MU) started a graduate program dedicated to software engineering, which was offered by its Computer Science Department. In 1995 Monmouth created the first Software Engineering Department in the United States. Now it is one of the pioneer universities offering a Bachelor's degree in software engineering.

One motivation for creating a separate software engineering program and department was the awareness of the skills that industry would like students to have upon graduation, which are not stressed by most computer science curricula. These skills include teamwork, communications, time management, engineering problem-solving, quantitative and qualitative process management, reuse, requirements management, system architecture, testing and project management.

As one of the few universities with such an extensive and comprehensive experience in offering software engineering programs, we have learned much about providing such a program. With more and more undergraduate software engineering programs appearing, we feel it would be beneficial to other institutions for us to share our experiences with them. A summary of the problems encountered and the lessons learned are presented here:

- *Revolutionary curriculum changes*. One can expect to revisit the overall curriculum of the program every four to five years, in order to accommodate changes in industry practice and educational expectations.
- *Continuous development of course content*. This is critically necessary, due to the dynamics of the field. The continuous development of course content implies also a continuous development of course projects, and dealing with dated textbooks, ever-changing operating systems, programming languages and software tools.
- *Difficulties of attracting and retaining faculty*. The need for new faculty to have a record of sustained scholarly accomplishments and industrial experience imposes great restrictions on the number of available candidates.
- *Diversity of the student body*. Issues raised by a diversity of educational backgrounds, employment status, educational goals, and communication skills bring challenges that need to be dealt with by any software engineering program.

The remainder of the paper discusses in detail the topics presented above. We will look at the curriculum evolution over the history of our program, discuss various issues involved in the continuous changes of the software engineering course content, outline our experiences in hiring and retaining the faculty, and show the influence of the diversity of the student body on the curriculum and course content. Based on all these experiences, the final sections present our advice for those interested in starting a graduate program in software engineering and identify where we expect the MU program to head in the future.

* Accepted 6 October 2003.

REVOLUTIONARY CURRICULUM CHANGES

Over its short history, software engineering (SE) as a field has been a moving target. We have observed the introduction of the Capability Maturity Model, the Unified Modeling Language, Personal and Team Software Process, and corporate adherence to ISO standards emerge as major forces within software engineering organizations. Therefore, a curriculum that addresses the skills and practices required by professionals in this field needs to continuously reinvent itself over time. In order to accommodate industry's needs and keep pace with the advances of software engineering as a field, we have added or dropped courses and added new tracks and programs. The decisions were made in the context of creating and maintaining a balance between the theory, technology and practical aspects of software engineering.

Changing the curriculum follows a well-defined process. First, the faculty discusses the need for change. Next, the program director writes a proposal identifying the new curriculum, and any additional courses that might be required. The proposal is put forward to the faculty in the department for comments. Once the proposal is approved within the department, it is sent to the chairs meeting within the school. After the eventual suggestions for change are incorporated into the proposal, it is submitted to the university graduate studies committee.

At the school chairs meeting, it may be decided that a stronger business case is required. An external body typically develops this business case. It is either a survey developed by an independent firm or by an external industrial advisory committee. The business case reflects the needs and state of industry, which will attract new students.

Next we present the evolution of the Monmouth University's graduate software engineering curriculum. This evolution shows a gradual transition from a software engineering program created inside a Computer Science department, towards a program with engineering courses that span the entire software lifecycle. It incorporates the results of a strong collaboration between academia and industry [2].

The initial curriculum (1986)

The initial curriculum consisted of 30 credits, with six core courses and four electives (see Fig. 1). The core courses covered in detail only the implementation (in Ada) and project management aspects of the software lifecycle, due to the limited availability of faculty with an appropriate background. The curriculum looked more like 'a computer science curriculum with an engineering flavor' [3], covering classic computer science courses like algorithms, operating systems, computer architecture and database management systems. The practical training of students was accomplished in a 3-credit practicum course, which consisted of a team project that would develop a software system from initial requirements to the final, tested and documented product. The early curriculum was biased more towards theoretical aspects (notice the heavy concentration on the mathematical foundations of SE), with less exposure to specific SE technology and practice.

1991 curriculum changes

This curriculum added a number of SE courses, including formal methods, formal specifications, the software process and SE environments (see Fig. 2). However, it still had a bias towards computer science, offering an artificial intelligence course, four courses of mathematical foundations and formal methods, and four courses in network technology, due to our geographic location in an area dominated by the telecommunications industry.

1995 curriculum changes

In 1995 the curriculum was substantially changed to include 36 credits, with 10 core and

Core Courses (6 courses = 18 credits)
SE 501 Mathematical Foundation of Software Engineering I (3 credits)
SE 505 Programming-in-the-large (3 credits)
SE 510 Computer Network Design (3 credits)
SE 516 Software Engineering I (3 credits)
SE 518 Project Management (3 credits)
SE 525 System Project Implementation (3 credits)

Elective Courses (4 courses = 12 credits)
SE 502 Mathematical Foundation of Software Engineering II (3 credits)
SE 506 Programming-in-the-small (3 credits)
SE 509 Programming Languages (3 credits)
SE 511 Protocol Engineering (3 credits)
SE 512 Algorithms Design and Analysis (3 credits)
SE 514 Computer Architecture (3 credits)
SE 515 Operating Systems Implementation (3 credits)
SE 517 Software Engineering II (3 credits)
SE 519 Database Management (3 credits)

Fig. 1. 1986 curriculum.

Core Courses (6 courses = 18 credits)
SE 501 Mathematical Foundation of Software Engineering I (3 credits)
SE 505 Software System Design (3 credits)
SE 506 Formal Methods in Programming (3 credits)
SE 516 Software Engineering (3 credits)
SE 518 Project Management (3 credits)
SE 525 System Project Implementation (3 credits)

Elective Courses (4 courses = 12 credits)
SE 502 Mathematical Foundation of Software Engineering II (3 credits)
SE 503 Introduction to Computer Communication Networking (3 credits)
SE 509 Programming Languages (3 credits)
SE 510 Computer Network Design (3 credits)
SE 511 Protocol Engineering (3 credits)
SE 519 Database Management (3 credits)
SE 522 Software Engineering Environments (3 credits)
SE 532 Software Process Quality (3 credits)
SE 534 Formal Specifications of Software Systems (3 credits)
SE 536 Fundamentals of Computer Security (3 credits)
SE 538 Advanced Topics in Networking Topology (3 credits)
SE 540 Introduction to Artificial Intelligence (3 credits)

Fig. 2. 1991 curriculum.

two elective courses (see Fig. 3), in order to comply with the Software Engineering Institute model curriculum [4]. That curriculum covered the entire software lifecycle in detail, by offering three new courses, specifically in requirements, implementation and reuse, and testing and quality. A former elective, software systems security, became a core course. Having such a heavy core, this curriculum offered little flexibility for learning aspects of SE that students would be most interested in. Another major change was reflected in the introduction of several new courses that would form 6-credit elective specialization tracks: in distributed software systems, software management, information systems, and real-time systems. These tracks were introduced as a response to the needs and feedback from our local industry and government collaborators [2]. The curriculum change was made possible by hiring faculty with both theoretical background and working experience in industry, supplemented with substantial help from adjunct faculty with expertise in specialized areas of SE.

1996 curriculum changes

In 1996 minor changes were made in the curriculum. It remained a 36-credit program, but students now had nine core and three elective courses, which offered a bit more flexibility than the previous program. The curriculum covered all the aspects of the software lifecycle. The capstone course was either a 3-credit practicum, or 6 credits of thesis research. The introduction of a thesis option was made possible by attracting faculty with the desire to engage in research activities.

1998 curriculum changes

The 1998 curriculum, which is the curriculum that we currently follow, represented another major change by providing for much more flexibility in a 36-credit program, with five core and five elective courses, and a 6-credit practicum or a 6-credit thesis (see Fig. 4).

All the knowledge areas of the Software Engineering Body of Knowledge (SWEBOK) project, as described in the Stone Man report [5], can be identified in this curriculum. The recognition of the importance of exposure to practical experience in a software engineering program has lead to an increase in the practicum project from 3 to 6 credits, and to the introduction of term projects in most of the courses in the curriculum.

Two of the former core courses, mathematical foundations of SE and principles of SE, have been transformed into preparatory (bridge) courses (see Fig. 4). Together with three other programming courses, the 'bridge' program is offered for students with an undergraduate major other than computer science, computer engineering, electrical engineering, or information systems. After taking the 15-credit preparatory courses and a one-semester project course, students can receive a certificate in software development if they do not wish to pursue a Master's program.

The 1998 curriculum has added a new course, The Process of Engineering Software, which largely follows Watts Humphrey's Personal Software Process (PSP) principles [6]. The introduction of this course was justified by the need for graduates who are aware and have the necessary skills for predictably producing high-quality systems, in a timely and cost-effective manner, using reusable components as much as possible in their work. In spite of the hard work necessary for the manual input of the data for the various forms and templates involved in the PSP, students have given us very positive feedback about the usefulness of the principles learned in this course. For alleviating the clerical work related to the manual input of data, we created a semi-automated tool to support the PSP process [7]. This tool was the

Core Courses (10 courses = 30 credits)
SE 501 Mathematical Foundation of Software Engineering I (3 credits)
SE 504 Principles of Software Engineering (3 credits)
SE 505 Software System Design (3 credits)
SE 506 Formal Methods in Software (3 credits)
SE 507 Software Systems Requirements (3 credits)
SE 508 Software Implementation and Reuse (3 credits)
SE 512 Software Testing and Quality (3 credits)
SE 513 Software Systems Security (3 credits)
SE 518 Software Project Management (3 credits)
SE 525 System Project Implementation (3 credits)

Advanced/Elective Specialization Tracks (2-course track = 6 credits)

Distributed Software Systems	SE 526 Networked Software Systems I
	SE 527 Networked Software Systems II
Software Management	SE 531 Software Organization Management
	SE 532 Software Quality Management
Information Systems	SE 541 Information Systems Architecture
	SE 542 Information Systems Engineering
Real-Time Systems	SE 551 Real-Time Software Analysis & Specification
	SE 552 Real-Time Software Design & Implementation

Fig. 3. 1995 curriculum.

Preparatory Courses (15 credits)
CS 500 Program Development
CS 503 Fundamental Algorithms I
CS 505 Operating Systems
SE 501 Mathematical Foundation of Software Engineering
SE 504 Principles of Software Engineering
Core Courses (15 credits)
SE 500 The Process of Engineering Software
SE 505 Software System Design
SE 506 Formal Methods in Software
SE 507 Software Systems Requirements
SE 512 Software Testing and Quality
Capstone Course (6 credits)—Practicum/Thesis

Specialisation Tracks (15 credits)

Organizational Management Track
Required (9 credits)
SE 531 Software Organizational Management
SE 532 Software Quality Management
SE 518 Software Project Management
Guided Electives (6 credits)
BM 525 Management of Human Resources
BM 565 Management of Technology
SE 560 Software Risk Management
SE 565 Software Metrics

Telecommunications Track
Required Courses (9 credits)
EE 537 Wireless Communications
SE 526 Network Software System I
SE 527 Network Software System II
Guided Electives (6 credits)
CS 526 Performance Evaluation
CS 535 Telecommunications
EE 505 Communications Technology
EE 581 Data Networks
SE 513 Software System Security
SE 598T Special Topics (Telecommunications)

Embedded Systems Track
Required Courses (9 credits)
SE 526 Network Software System I
SE 551 Real-Time Software Analysis and Spec.
SE 552 Real-Time Software Design and Impl.
Guided Electives (6 credits)
CS 525 Simulation
CS 526 Performance Evaluation
EE 509 Digital Signal Processing
SE 508 Software Implementation and Reuse
SE 513 Software System Security
SE 527 Network Software System II

Information Management Track
Required Courses (9 credits)
SE 541 Information Systems Architecture
SE 542 Information System Engineering
SE 518 Software Project Management
Guided Electives (6 credits)
BM 520 Information System in Organisation
BM 565 Management of Technology
BM 571 Introduction to US Health Care
CS 517 Database Systems
CS 530 Knowledge-Based Systems
SE 508 Software Implementation and Reuse
SE 526 Network Software System I

Fig. 4. The 1998 curriculum.

result of a two-semester practicum project of one group of students.

The elective courses included in this curriculum were necessary for completing a chosen specialization track, such as organizational management, telecommunications, embedded systems, and information systems. These 15-credit tracks were much more comprehensive than their counterparts in the 1995 curriculum. They comprise courses from other disciplines, such as business, electrical engineering and computer science. However, students were able to select elective courses across tracks if they did not want to pursue a specialization. A brief description of the specialization tracks follows:

- The *Organisational Management* track prepares students to become software development managers or specialists in software process improvement. Topics of study include process improvement, quality management, organisational development and management, risk management and project planning and management.
- The *Telecommunications* track prepares students to become specialists in telecommunications. Topics of study include networks, software systems security, and evaluation of telecommunications systems.
- The *Embedded Systems* track prepares students to become specialists in embedded systems development. Topics of study include specification and analysis of embedded real-time systems requirements, design and implementation of embedded real-time software systems, performance evaluation of embedded real-time software systems, and development of real-time components.
- The *Information Management* track prepares students to become chief information officers or specialists in information systems integration and development. Topics of study include information technology management, specification and analysis of information systems, evaluation of information systems, and development of information systems software components.

2002 curriculum changes

The 2002 curriculum added a new specialization track: Management of Software Technology. This is offered in collaboration with the Monmouth University School of Business. The idea of this track grew out of the recognition that industry is outsourcing increasing amounts of software development. This track prepares students to be chief technology officers or specialists in the acquisition of software systems for businesses. Topics of study include assessing the impact that software can have on organizations, development of requirements for system acquisition via purchase or outsourcing, assessment of software technologies with regard to organizational needs, and implementing a controlled introduction of technology into an organization.

CONTINUOUS DEVELOPMENT OF COURSE CONTENT

Technologically, the computing field has undergone significant changes that have forced alterations in the material taught within Software Engineering courses. Since the inception of our SE Master's program, we have witnessed the widespread adoption of Object-Orientation (along with massive changes in techniques and methodologies), the phenomenal explosion of the World Wide Web, the emergence of Java, and the move of security requirements from corporate to consumer platforms, just to name a few of these changes. Therefore, the material covered within a curriculum that addresses the technological understanding required by professionals in this field needs to be continuously updated over time. This problem emerges in several different forms, particularly:

- continuous course content changes;
- dated textbooks;
- operating system/programming language bigotry; and
- continuous development of course projects.

We will now discuss each of these areas in greater detail.

Course content changes

One can expect to have to revise course material every year. This is necessary to accommodate technological changes and incorporate new industrial practices. For example, since the inception of our program we have changed the programming languages taught in class from Ada to C++ and Java; we have added object-oriented analysis methods to the structured analysis methods in the requirements engineering course [8]; in the design course we have made the transition from structured design to object-oriented design, component-based design, and architectural design. In the testing course we have added segments on testing applications that are constructed using commercial off-the-shelf (COTS) components, using automated testing and test management tools. For project management, we have gradually introduced more content on the use of scheduling tools such as MS project and risk simulators like Risk+, and discussion of the use of buffer tasks in the planning of software development projects [9].

Dated textbooks

As technology changes and software engineering evolves, the ability of texts to keep up with the changes is severely stressed. An instructor will find himself or herself utilizing three or four texts in order to properly cover a topic area. Books will seemingly contradict each other, only because they were published two years apart. Often, a book that is only three years old will contain many concepts that have already been superceded. Many excellent textbooks have not been updated to use current representations, such as UML, for instance. These

generate the need to continuously research the new and updated prints, and take into consideration student feedback on the usefulness of the recommended textbooks.

Operating system/programming language bigotry

Few topics seem to generate as much debate as which operating system (OS) or programming language should be selected as the lingua franca for course work. It seems that everyone has an opinion or a realistic need to learn one environment over another. The selection of one environment over another has significant impact on the tools available for use by the instructor, the knowledge that the instructor has to bring into the classroom, and the equipment that must be maintained. In our case, over the years we have migrated from UNIX platforms to Windows, and to dual-boot machines that run both Windows and Linux. Most of the students are familiar with both operating systems, since different instructors favor one OS over the other. They appreciate the flexibility offered by the dual-boot machines available in our labs.

Continuous development of course projects

Faculty, students, and industry have universally recognized the need for hands-on experience. Without practical training, students and industry complain that the material will be too theoretical and that graduates would have trouble applying the theory to real-world projects. This has led us to incorporate projects into the majority of courses taught in the program, while maintaining a balance between the theoretical and practical aspects of the courses. The projects are administered at the beginning of the semester and have a couple of milestones spread along the semester. The instructors check the documents and/or software applications delivered at each milestone and provide feedback to the students. For some projects, the problem to be solved is proposed by the instructor; for others, students can propose their own project theme. The members of the project teams are either established by the students, when they are not new to the program, or, when no preferences are expressed, the instructor makes the choices. The teams have the authority to choose their leaders and the role of each member.

The introduction of projects into a Software Engineering course encompasses its own set of difficulties. While a simple program for shuffling cards may suffice to teach students about algorithms and data structures in a programming course, software engineering has to deal with much larger problems in order to demonstrate the value and need for an engineering process. The result is that projects have to be big, but not so big that they cannot be performed within the confines of the course. Because the project has to be big, it has to be structured such that the students can incrementally develop it as the course unfolds.

As the course content, technology and available tools change, the course projects need to change too. We have found that the size issue can sometimes be addressed by partially completing the project before presenting it to the students. This might require the development of a set of requirements before introducing a larger project into a software design course, providing some economic or financial analyses before introducing project into a software project management course, or developing requirements and code before introducing a project into a testing course. In any case, such a strategy requires that the instructor spend significant time doing the background work and documenting the results of that work so that the students can make good use of it as they proceed with the next steps. In this way, the students are encouraged to concentrate on tasks for a specific project that are unique to the course in which the project is being used.

DIFFICULTIES OF ATTRACTING AND RETAINING FACULTY

Software engineers, even in the current difficult economic times, are a highly sought-after commodity. It is extremely difficult for any software engineering program to both attract and retain their faculty. We have noticed that the stability of the faculty makes a program more attractive to prospective students.

It is very difficult to attract appropriate faculty. In particular, faculty members usually have to be acquired from computer science backgrounds and/or industrial practice. The problem with faculty from computer science backgrounds is that their backgrounds are in computer science rather than software engineering. The problem with acquiring faculty from industry is that they often do not have documented credentials and a documented trace of their scholarly work.

With the need to continuously update course content and curricula in order to keep up or advance the state of the field, the load on a faculty member in software engineering tends to be significantly greater than in some other academic areas. Given that it is very difficult to hire faculty with the appropriate academic and industrial backgrounds, many of the hires are often non-tenure track.

The only real solution for the administration is to provide competitive salaries and support consulting or research activities. This enables faculty to make up any shortfalls in salary and keep abreast of the industry needs and practices. With respect to this issue, MU offers faculty one day a week to spend on research or consulting activities.

DIVERSITY OF THE STUDENT BODY

In the 16-year history of the software engineering program at MU, we have observed increasing diversity within the student population. This diversity spans several dimensions: educational background, employment status, educational goals and native language. The successful program must address all these dimensions of diversity.

Educational backgrounds

Consistent with the origins of the program, many students in the graduate program have undergraduate degrees in computer science. These students have strong programming skills but very seldom have the engineering discipline that emphasizes understanding the problem to be solved, or the process to be followed. These students tend immediately to start coding once they receive a problem to be solved. Instructors have been asked on more than one occasion why it was necessary to design a program when they could write one faster.

We also have a large population of students that are coming into the graduate program from other engineering and non-engineering disciplines. These students usually are much more accepting of engineering processes but have relatively weak programming skills and minimal knowledge about how computers function. To accommodate them, we have had to incorporate a set of preparatory courses to provide the programming skills and computer knowledge necessary to succeed in the program.

We are expecting a new group of students to begin entering into the graduate program in two years. These students will have undergraduate degrees in software engineering and will already have a good understanding of engineering practices balanced with programming skills. At this point, we anticipate that our program will have to address increasingly more advanced software engineering topics that may be beyond the knowledge of the other two groups of students.

Employment status

The employment status of students has significant impact on the program. It affects how long students are in the program, the effort that they put into assignments, their willingness to accept course material, and when classes are offered. It should be noted that (with a few exceptions) students entering into the program full-time usually find work at the end of their first year and become part-time students. The majority of our student population attends school part-time while being employed full-time in the software industry, so most of our classes are offered in the early evening to accommodate them.

The fact that the average student is employed full-time and attends classes part-time means that they may be in the program for as long as eight years. In fact, the population of students is much more stable than the curriculum. Some students have graduated on curriculums that have been replaced twice since they enrolled in the program.

Employment in the software industry has significant impact on the willingness of some students to accept the concepts taught in the classroom. These students have already acquired work habits that are not consistent with best practices. Students often state that they do not perform a particular engineering practice at work and that they do not see a need for it. Of course, many of these same students talk about how their projects at work tend to be chaotic. Other students report the difficulties they have encountered in trying to practice in their conservative organizations what they have learned in class. Either case tends to undermine the instructor in presenting new material in the classroom. Here is one of the situations where the instructor's industrial experience plays an important role in both selecting the material to be taught and in responding to student concerns regarding the usefulness of the topics learned in the real world.

Employed students also tend to focus on what they immediately need to succeed in today's workplace. There is often an insistence on learning a product (such as Oracle or Sybase) rather than the concepts (i.e. database principles). This emphasis on skill rather than knowledge runs counter to the main goal of the program, which is to produce software engineers who can lead their organizations into the future. We have incorporated some of these products into our classrooms, but the main aim of the courses remains to teach the engineering principles of the field, which can be applied to a large number of products.

Students who are not employed in the industry have problems prioritizing the material being taught or placing it in the context of delivering a product. If they are required to know C++, they assume that all employers develop code in C++. They are often surprised when they get a job and discover that they will have to learn a new programming language. Students are occasionally concerned that courses cover many different methods and approaches to achieve a given goal rather than emphasizing one method. They have to be taught to understand that the knowledge and skill they acquire in school will have to blend into whatever organization they join and that they need to engage in a lifelong learning process that is inevitable in this dynamic field.

Educational goals

It would be nice if all students entered the program with the desire and goal of becoming a software engineer and delivering a specific kind of product. However, the educational goals of the students range from wanting to know all about software and engineering to the other extreme, where they only want to get the credentials that will allow them to earn a higher salary. Our

student body appears to be driven by a small number of educational goals. These are:

- to get the business and process knowledge that will allow them to manage software projects and people;
- to acquire the skills and knowledge that will allow them to be more productive in their chosen career;
- to start a career in which they can have a significant income; or
- to get a job in the software field that does not involve a lot of coding.

The major impact of these goals concerns the subject areas that interest the student. We have had to tailor our curriculum to respond to these different goals. We find a significant fraction of the students are very interested in the process, project management, and organizational management courses. Others find that the courses on requirements and software testing give them an entry point into a part of the software business that does not appear to require major coding efforts. Finally, the courses that emphasize specific types of software systems (real-time, information, and embedded systems) attract those students that are interested in gaining the particular knowledge and skills that will allow them to master their chosen field of work.

Communications skills

There is significant diversity among our students in terms of their communication skills. However, communication skills are critical in software engineering. The average software engineering student will probably produce more documents and make more public presentations than the average English major. Communications have to be precise, unambiguous, complete, logically sound and well structured. Oral presentations have to convey complex information under time constraints. Students have to learn to gauge how much information is to be conveyed. This requires that they judge what their audience can be expected to know and what must be presented. Although typical undergraduate general education programs attempt to teach these skills, most students who enter our graduate program require additional coaching and training in this area.

International students are often at a disadvantage, due to the fact that English is their second language. This affects their writing ability, where a weakness in vocabulary often prevents them from expressing themselves clearly and succinctly. It also undermines their confidence in public speaking, due to concerns about their command of the language and fears that others will not understand them because of their accents.

International students are not the only ones with problems in communication. Many of the students, particularly those with computer science backgrounds, are not used to writing technical documents. While they may be good at writing code, they often have difficulty expressing themselves succinctly in a written document.

The most direct approach to dealing with significant changes in the student population has been to adapt the curriculum and individual courses to meet the changing needs of our students. Employed students are encouraged to express their perspectives on the material, so that their experiences can be shared with students that have not yet entered the field. In some classes, programming assignments can be written in Java or C++, depending on the students' choice.

Another change has been the incorporation of more term papers into course work so that students get greater experience in writing. Papers are graded on technical content, structure, adherence to topic, and on the use of language. Corrections are suggested and students have a chance to resubmit corrected work. With respect to verbal communications, students are required to make oral presentations of their term projects. In this way, until they reach the capstone project, students will have had the opportunity to exercise their communications skills several times.

GUIDANCE ON STARTING AND MANAGING PROGRAMS

Based on the experiences described above in starting and managing Monmouth University's software engineering program, we would offer the following advice to academic departments that are considering a similar program:

1. Conduct research to determine the most current curriculum recommendations of the IEEE, ACM and other sources.
2. Find out, by participating in national groups and committees that develop those recommendations, what future changes are likely to take place.
3. Enlist the academic institution's industrial advisory boards to determine how the general recommendations need to be tailored to suit the needs of local industry.
4. Form a task force with professors from both SE and CS departments to make sure the two departments will not conflict with each other. Also, invite an external reviewer who can offer concrete guidance, based on personal experiences in building such a program at another university.
5. Recruit full-time faculty who are competent to teach the required variety of courses and who have industrial experience in applying software engineering techniques in real-work environments. Do not expect this to be an easy task. You may need to manage the program initially with significant help from part-time faculty.
6. Expect that the curriculum will need to change over time to accommodate both changes in the discipline as well as changes in the needs of local employers.

FUTURE DIRECTIONS

Having looked at the past, it is now appropriate to look to the future for our program. In particular, we recognize a need for another set of changes. The introduction of an undergraduate software engineering program will have profound consequences on the graduate program, forcing severe changes in its curriculum. The redesigned curricula should allow the new graduates of the Bachelor's degree in software engineering to have the opportunity to extend their knowledge and skills to new frontiers.

With this in mind, we plan to modify our graduate program, such that the students with a Bachelor's degree in SE will be required to take seven elective SE courses and a 6-credit thesis. This would make our SE graduate program similar in structure to Master's programs in electrical engineering, mechanical engineering, etc., throughout the United States.

Another issue, which is beyond the scope of this paper though, is the influence that the licensing of software engineers will have on the design of the curriculum. However, the directions and discussions that are taking place with regard to licensing have to be followed so that appropriate changes can be implemented in the curriculum.

CONCLUSIONS

This paper has presented the main problems and lessons learned from one of the oldest programs in software engineering in the United States. The evolution of the graduate curriculum over its 16 years of existence has been described as an example for other colleges and universities considering implementing a software engineering degree. We expect this evolution to continue in the future, as the SE field is a constant moving target, and as we will graduate the first class of Bachelors in SE.

We have argued that continuous updating of the course content has a special meaning in software engineering, due to the dynamics of the field. In this respect, we have shown the impact of the advances in the field on the textbooks used and the need for continuous reevaluation of the chosen programming language, operating system, or software tools used in class.

The paper has presented the difficulties we have experienced in attracting and retaining faculty over the years, due to the need for new faculty to have both a record of scholarly accomplishments and industrial experience. The emphasis here is on the conjunction of these two requirements, which sets great restrictions on the pool of available candidates.

We have shown how various issues related to the diversity of the student body influence the curriculum and course content. As such, the educational backgrounds, employment status, educational goals, and communication skills of the student body are challenges any software engineering program has to solve.

Based on our experience in dealing with these problems, we have offered some guidelines for those interested in starting a similar program.

As a measure of the success of our continuous efforts to improve, we have seen the program enrollment increasing steadily, and doubling in the last three years. Apparently our strategy to continually update the program appears to be working, as mentioned by one of our alumni, Kevin McKee (VP Information Services, Health Network America, Eatontown, NJ):

> *The education I received while getting my Master's degree in Software Engineering at Monmouth University has been invaluable. Not a day goes by when I do not use something that I learned in the program.*

REFERENCES

1. P. Naur and B. Randall (eds.), *Software Engineering: A Report on a Conference Sponsored by the NATO Science Committee*, NATO (1968).
2. G. Powell, J. Diaz-Perrera and D. Turner, Achieving synergy in collaborative education, *IEEE Software*, Nov/Dec (1997) pp. 58–65.
3. P. Dart, L. Johnston, C. Schmidt and L. Sonenberg, Developing an accredited SE program, *IEEE Software*, Nov/Dec (1997) pp. 66–70.
4. M. Ardis and G. Ford, SEI report on graduate software engineering education, *Proceedings of the Software Engineering Education Conference*, Springer-Verlag (1989).
5. Guide to the Software Engineering Body of Knowledge, Stone Man Version, SWEBOK, Feb. 2000 (http://www.swebok.org/).
6. W. Humphrey, *A Discipline of Software Engineering*, Addison-Wesley, 1997.
7. D. Rosca, C. Li, K. Moore, M. Stephan and S. Weiner, PSP-EAT: Enhancing a personal software process course, *Proceedings of* FIE'01, p. T2D18.
8. D. Rosca, An active/collaborative approach in teaching requirements engineering, *Proceedings of* FIE'00, pp. T2C9–12.
9. J. McDonald, Teaching software project management in industrial and academic environments, *Proceedings of CSEE&T* (2000) pp. 151–160.

Daniela Rosca is Assistant Professor in the Software Engineering Department at Monmouth University. She received her Ph.D. degree in Computer Science from Old

Dominion University. Prior to obtaining her doctoral degree, she worked for the Institute for Computer Technologies as a Senior Scientist. She was responsible for developing artificial intelligence applications. Her research interests are positioned at the intersection of software engineering and artificial intelligence. Dr Rosca's main areas of specialization are business rules, a special type of software requirements, and software process. Her research in terms of educational methods has focused on active and cooperative learning, multidisciplinary teaming, and software tools for enhancing the learning process.

William Tepfenhart is the author of several textbooks on object orientation. He is currently an Associate Professor in the Software Engineering Department at Monmouth University investigating the potential of software solutions to enhance the effectiveness of collaboration in engineering endeavors. Prior to his entry to the academic world, he was employed as a developer and technologist at AT&T Laboratories, where he worked on applications associated with the long-distance network, establishment of engineering practices at a corporate level, and working with advanced object-oriented technologies. He had previously worked as a Senior Scientist at Knowledge Systems Concepts, investigating the use of artificial intelligence systems for the USAF. Prior to that, he was an Associate Research Scientist at LTV, where he worked on applications for manufacturing devices composed of advanced materials.

James McDonald, Associate Professor and Chair of the Department of Software Engineering at Monmouth University, holds a Bachelor's degree in Electrical Engineering from New Jersey Institute of Technology, an M.S.E.E. from Massachusetts Institute of Technology and a Ph.D. from New York University. Dr McDonald has an industrial background in both software development and digital hardware design. He is a senior member of the IEEE and a member of the IEEE Computer Society, the Association for Computing Machinery, the American Society for Engineering Education and the Project Management Institute.