COMPUTER SCIENCE EDUCATION 5, 1–13, (1994)

Planning for Software Maintenance Education Within a Computer Science Framework

Hossein Saiedian
James Henderson
University of Nebraska at Omaha

This article proposes that software maintainers would benefit from specialized education and training with regard to software maintenance. A clear distinction is drawn between the tasks involved in software development and those involved in software maintenance. It is shown that neither the average computer science undergraduate degree program, nor experience in software development fully prepare a programmer for the particular challenges faced in software maintenance. The traditional and evolving coverage of software maintenance in U.S. graduate and undergraduate computer science and software engineering degree programs is discussed. A set of proposals is given for introducing undergraduate computer science students to the field of software maintenance. An alternate approach is recommended for introducing experienced development programmers to software maintenance.

1. INTRODUCTION

Many software engineering researchers have stated that the software industry is at a critical point in its evolution, some say a "crisis" stage. Pressman [1: 18] explained the prevalent problems of this crisis as follows:

1. Schedule and cost estimates are often inaccurate.
2. The "productivity" of software people has not kept pace with the demand for their services.
3. The quality of software is less than adequate.

Correspondence and requests for reprints should be sent to Hossein Saiedian, Department of Computer Science DSC 203, University of Nebraska, Omaha, NE 68182.

lihood, easier to maintain, this still fails to address a large portion of the problem.

As an example of the unaddressed magnitude of the problem, Weinman [3: 32] stated that "software inventory worldwide comprises more than 100 billion lines of working source code, with Cobol making up more than 80% of that. Conservative estimates of the cumulative cost of producing those systems is $2 trillion." With that kind of investment, it is unlikely that companies are going to have all of these systems replaced at any point in the near future. Therefore, as long as they exist, we will have sizable maintenance problems. Even when the last of these legacy systems is replaced, the systems that succeed them may be viewed as equally difficult to maintain by those called upon to do so many years later.

3. DIFFERENTIATING DEVELOPMENT AND MAINTENANCE

Is software maintenance truly that different from software development? The answer seems to be, quite clearly, yes. As Wedo [4: 28] pointed out: "Maintenance is not identical to development and a programmer should recognize this." Let us look at some of the ways that software maintenance differs from software development.

Wedo [4] further pointed out that one of the most significant differences between maintenance and development is that development involves designing and writing programs that do not yet exist, whereas maintenance entails programs already running. Since this operational program is already running there is no luxury of time. Any creation of problems or delay in the fixing of a problem will have a direct effect on users of the system today.

Paraphrasing Chapin [5: 5], maintenance programmers are subject to more pressure than development programmers, in part because of the large number of varied projects that a maintenance programmer completes during a time period. Chapin even went so far as to assert that maintenance programmers have greater demands placed upon them for reasoning, creativity, imagination, problem solving, and a facility for expression. While this may be overstated, the field of software maintenance is certainly a mentally challenging one.

One of the most interesting differences that Glass and Noiseux [6: 18] espoused is that "software maintenance remains a bastion of the individual worker." They noted that quite frequently a programmer will be solely responsible for one or more programs. Maintainers must be flexible enough to adapt to different styles of code and distinguish

However, software maintenance is given scant individual attention in the majority of U.S. computer science programs.

4. THE COVERAGE OF SOFTWARE MAINTENANCE IN EDUCATION

It is hard to get an accurate, objective assessment on the degree to which software maintenance is covered in U.S. computer science degree programs. However, Collofello [9: 26] had this to say: "Although software maintenance is widely regarded as being the dominant activity performed in most software development organizations, very little attention has been paid to software maintenance in either university or industrial training courses." Many software engineering textbooks mention maintenance, but often in the vein of, "develop your code so it is maintainable in the future."

Leventhal and Mynatt [10] surveyed undergraduate software engineering textbooks with regard to course content and found that 80% had some coverage of maintenance. Approximately 50% of these courses reported that the coverage was limited, 35% moderate, and 15% in-depth. Thus, only 10% of undergraduate software engineering textbooks provide in-depth coverage of software maintenance, whereas 60% provide limited or no coverage at all.

Cornelius, Munro, and Robson [11] vividly illustrated the paradox that this limited coverage of maintenance brings about. They pointed out that the emphasis in most computer science and software engineering courses is on developing new software. They stated that students are rarely exposed to existing code developed by someone else. Furthermore, they explained [11: 233]: "Once students take up employment in industry, the situation is almost reversed." Clearly, many students are not directly educated for the work that they will be doing, nor are many programmers properly trained to go from development to maintenance. Now let us examine some things that can be done to improve this education and training situation.

5. EDUCATION AND TRAINING FOR MAINTENANCE

Obviously, the instructional needs are somewhat different for students versus programmers who already have substantial development experience. We will examine the needs of students for specific educational experiences and the needs of development programmers for specific

Wedo [4] emphasized that the programmers first job has got to be learning enough to perform corrective maintenance due to its critical nature. He also distinguished between a detailed examination of a subprogram's code and an understanding of the functions of the program of a whole. Finally, Brooks [16] recognized that the essence of high-level program understanding is not just in understanding the code, but also in the mappings between the programs and its problem domain. We will discuss the need for training in the lower level reading and understanding of individual programs in the following.

Aside from the aforementioned knowledge, the education necessary for students tends to diverge from the training necessary for development programmers. We first examine the educational needs of students and then the training needs of development programmers.

5.1 Educating Students On Maintenance

Specific to the educational needs of students are two pressing needs: teaching program reading and software maintenance exercises. As children, when we are taught English (or any other native language), we are first taught to read and then to write. However, when we learn a programming language, we are generally only taught to write. Bergland [17] pointed out that we miss something by not learning to read programs first. Furthermore, Glass [12: 201] quoted Bill Gates (founder and head of Microsoft) as follows:

> The best way to prepare to be a programmer is to write programs and to study great programs that other people have written . . . I went to the garbage cans at the Computer Science Center and I fished out listings of their operating system.

Teaching students how to really read and comprehend programs is, of course, related to program comprehension, mentioned earlier, but it is at a different level. Students need to be given exercises where they have to read an undocumented program that they know nothing about and accurately figure out what it is doing. Unfortunately, they may very well be doing the same thing on their first day at work.

In addition to program reading, students need to be exposed to significant maintenance exercises. Calliss and Trantina [18] discussed a project that requires groups to communicate with each other as they work on an evolving program.

Tomayko [13] provided good coverage of a software maintenance course that he developed. His discussion of the course brings out that the course cannot be purely theoretical and must have a significant

need for instructors with experience in coping with the complexity of maintaining large systems [20] and the lack of adequate texts on the subject [21]. Nevertheless, because only a small minority of computer science graduates go on to pursue a masters degree in computer science, most entry level programmers aren't receiving this instruction. An additional factor according to Cardow [20] is that undergraduates need more focus in technical and support issues, whereas graduate students need more focus in management and support issues. We need to add software maintenance classes or make software maintenance a very significant component of software engineering classes at the undergraduate level to ensure that those who need it most get this information.

5.2 Training Development Programmers for Maintenance

The training of development programmers to do maintenance has quite a bit in common with the educating of students to do maintenance, but there are significant differences. According to Calliss [21: 320], "people in industry need to know that what they are learning will be of value to them in their work." Although this is certainly true of students as well, it is probably safe to say that people working in the industry are more inclined to take things with a grain of salt. Speaking of software engineering education in general, Besemer, Decker, Politi, and Schnoor [22: 401] cited the following reasons why we have to view educating those in industry somewhat differently:

> Academia and industry attempt to educate the software engineer with methods that lie at opposite ends of a spectrum, neither of which is sufficient alone. University courses try to give a broad view of software engineering, but the brevity of the courses dictates that the student will not attain a firm grasp of the problems and principles.

In the realm of understanding programs, we know (or have a good probability) that a programmer who has been working in software development has a reasonably strong background in writing software. However, we have little, if any, indication that said programmer knows how to read software and accurately assess its function. Therefore, exercises are called for to ensure that these programmers are able to apply their skills in reverse.

Software maintenance project exercises are also called for, but in the case of already experienced programmers, they will take the form of context exercises. These exercises will require the programmer to develop a solution to a complex problem within the constraints of an

grammers. Additionally, programmers need to be trained in modern software maintenance techniques and the tools to support them. Perhaps most significantly, we need to show programmers a structured approach for gaining an understanding of the software components for which they are responsible.

Computer science students need to be instructed in how to read (as well as write) computer programs, preferably learning to read them before they learn to write them. We must provide software maintenance courses for students that require working in groups, and use both well-documented and poorly documented large software artifacts to teach good software maintenance practices. There is currently a dearth of such software maintenance courses, though the situation is improving with the proposal of a software maintenance unit in the Software Engineering Institute curriculum [7]. However, the proposed curriculum has been implemented (or partially implemented) only at a relatively small number of schools and then (quite frequently) only at the graduate level. This situation leaves a large portion of the programmers likely to be performing software maintenance without the benefit of these courses.

Programmers who lack maintenance experience, but who have worked in software development, have a different set of needs than computer science students. These development programmers need instruction in the reading of software components for program understanding. Also, these programmers need training in the solving of software problems within the constraints of an existing system such that they don't unnecessarily detract from the existing structure. The recommended training approach for these development programmers would be a full-time training program for their first several weeks after hiring, followed by several weeks of half-day instruction while beginning the comprehension of their primary software components.

If we want to make the maximal progress in software maintenance, we must invest in training for maintenance personnel. For software maintenance, the three primary personnel sources are experienced maintenance programmers, recent college graduates, and experienced programmers who lack maintenance experience. We may initially assume that the first group has the requisite skills, but we cannot assume the requisite skills among the other two groups.

Undergraduate (and graduate) computer science degree programs must offer significant software maintenance instruction and we must provide software maintenance instruction for programmers who lack such experience. If we do so, shorter learning curves will, in all likelihood, become the norm, and we will enjoy greater productivity within software maintenance organizations. Being a major portion of the soft-

[12] R. L. Glass, *Software Conflict: Essays on the Art and Science of Software Engineering*. Englewood Cliffs, NJ: Prentice-Hall, 1990.

[13] J. E. Tomayko, "Teaching maintenance using large software artifacts," in *Software Engineering Education* (Lecture Notes in Computer Science 376), N. E. Gibbs, Ed. Berlin: Springer-Verlag, 1989, pp. 3–15.

[14] P. Oman, "Maintenance tools," *IEEE Software*, vol. 7, no. 3, pp. 59–65, May 1990.

[15] S. D. Fay and D. G. Holmes, "Help! I have to update an undocumented program," in *Proc. Conf. Software Maintenance*, Nov. 1985, p. 194.

[16] R. Brooks, "Towards a theory of the comprehension of computer programs," *International Journal of Man–Machine Studies*, vol. 18, pp. 543–554, May 1983.

[17] G. D. Bergland, "Guided tour of program design methodologies," *IEEE Computer*, vol. 14, no. 10, pp. 13–37, Oct. 1981.

[18] F. W. Calliss and D. L. Trantina, "A controlled software maintenance project," in *Software Engineering Education* (Lecture Notes in Computer Science), J. E. Tomayko, Ed. Berlin: Springer-Verlag, 1991, p. 25.

[19] K. R. Pierce, "The benefits of maintenance exercises in project-based courses in software engineering," in *Proc. Conf. Software Maintenance*, Nov. 1992, p. 324.

[20] J. E. Cardow, "Can software maintenance be taught?" in *Proc. Conf. Software Maintenance*, Nov. 1992, p. 322.

[21] F. W. Calliss, "An outline for a software maintenance course," in *Proc. Conf. Software Maintenance*, Nov. 1992, p. 320.

[22] D. J. Besemer, K. S. Decker, D. W. Politi, and J. F. Schnoor, "A synergy of industrial and academic education," in *Issues in Software Engineering Education*, R. Fairley and P. Freeman, Eds. New York: Springer-Verlag, 1989, pp. 399–413.