Integrating a Software Engineering Approach into an Ada Closed Laboratory:

Submitted

As a Technical Paper

To
SIGAda'99 Conference

May 1, 1999

Abstract

A manual for a closed laboratory incorporating software engineering concepts was developed and implemented for Ada'95 during the 1998-1999 academic year. The software engineering concepts were applied during the development and implementation. The purpose was to teach student programmers (and prospective software engineers) disciplined methods for developing software. The findings were: 1. the closed laboratory provided an ideal environment for students to explore, learn and implement software development methods as well as learn the Ada language. 2. The closed laboratory was a major contributor for students to understand the course contents and built a bridge between the class lectures and open lab projects. 3. The closed laboratory helped students to accomplish their open lab projects in a more efficient and professional way by following software development methods. A survey conducted at the end of the semester indicated the success of implementation of software development methods in the Ada closed laboratory. The responses showed that students enjoyed the closed laboratory and considered it an exciting and interesting learning experience.

1. Introduction

Ada was developed in response to the crisis in software development and specifically embodies and enforces many modern software development principles. Teaching the Ada language we wanted to integrate these principles into our class. Traditional computer science laboratories and projects highlight the details of programming languages only from a syntactic, semantic or algorithm perspective. In this paper we introduce an approach which integrates software development methods into the development of the Ada closed laboratory and open lab projects. The laboratory activities included software design, test plan, implementation and debugging as well as personal software process.

The closed laboratory met at an assigned time and place with a laboratory instructor each week. Every student had a computer to work. Each closed laboratory activity typically illustrated the concepts from lectures by using examples and problems that were designed to challenge the student. Lectures and lab activities were synchronized so that the lecture topics were reinforced by the closed laboratory activities.

The closed laboratory is a recent concept in computer science. With the publication of the ACM Curriculum'91 Report [1], laboratory exercises were suggested for many of the knowledge units. However, a precise definition of what constituted a closed laboratory activity was not included. The Denning Report [2] introduced the term closed laboratories without defining exactly what they were as follows:

1. A scheduled time when students work on programming assignments under supervision.
2. A scheduled drill and practice time when students work on mini-problems under supervision.
3. The use of specially prepared laboratory materials where students interact with the computer as they would a microscope.
4. The labs should help students discover principles and solutions under supervision.

The open lab projects were similar to traditional computer science lab assignments. An instructor provided an initial set of requirements. Each student needed to complete the lab assignment outside of the class individually. The difference was that students needed to submit their design, test plan and Personal Software Process Summary Form in addition to the source code, output and executable code. Both Ada programming skills and software engineering process were emphasized in our new approach. Students learned Ada programming and the software development methods in class, practiced them in the closed lab and then applied them to their open lab projects. The content of the closed lab was utilized in the open lab (out of the closed lab) projects.

The goals for the closed laboratory were:

1. to help students achieve better mastery of the course materials including software development methods and the fundamentals of Ada programming.
2. to help students to explore and experience the disciplines of software engineering
3. to prepare students to develop (open lab) projects in a more efficient and professional way.

II. Background

At Embry-Riddle Aeronautical University the software engineering process is emphasized in both the undergraduate and graduate curriculum in computer science. Ada was adopted as a primary language in the department. Integration of software engineering concepts into the Ada closed laboratory was considered important. At Embry-Riddle University, we believe it is critical for student programmers (and prospective software engineers) to start off with disciplined methods for developing software.

The closed laboratory was integrated into the computer science curriculum in the 1998-99 Catalog at the Embry-Riddle University. One credit of the closed laboratory was added to traditional three-credit Ada programming courses. The purpose was to provide a time and environment for students to learn the software development process and programming. Since there weren't any existing Ada'95 closed laboratory manuals available at the time, the author developed a closed laboratory manual in the summer of 1998. The manual was developed on the base of the textbook, "Ada'95 Problem Solving and Program design" by Feldman and Koffman [3]. The manual was implemented with about 200 students in the fall semester of 1998 and the spring semester of 1999.

III. Software Engineering Approach

To help students avoid poorly designed programs and ineffective processes, it is important to expose them to a professional approach for designing programs at an early stage. Students could learn and implement software development methods in an interactive closed laboratory environment.

Traditionally, a three-credit programming course consisted of three one-hour lectures per week and open lab assignments. Typically, an initial set of requirements were always provided by an instructor. However, these initial requirements were sometimes incomplete or ambiguous. Unfortunately students had to refine them by using their own judgement and had to build the bridge between lectures and open lab assignments by themselves. The submission of an assignment usually included the source code, output and executable code. The grade was based on the correctness of the algorithm and output.

Comparing to the traditional programming course, our new way was three one-hour lectures followed by a two-hour closed laboratory activity each week, and then followed by an open lab project. Lectures introduced new concepts. The closed laboratory was designed to illustrate and explore the concepts from the lectures at the same time that it established the foundation of the open lab project. Then, the open laboratory project combined outcomes from both lectures and the closed laboratory.

In the closed lab environment students could have a discussion to refine the open lab project requirements until all key decisions had been addressed. These included the

scope of the project and input/output data requirement. Then they conducted their own design and wrote a test plan for their project in the closed laboratory. Each student implemented his open lab project design outside of the close laboratory individually. The open lab project submission included the design, test plan and Personal Software Process summary form as well as the source code, executable code and output.

1. Design

A software design specifies how a program will accomplish its requirements. The quality of software is only as good as the process used to create it. According to Grove's research [4] conducted with Personal Software Process summary form, proper design methodology in software development can reduce the number of defects and program development time and produce better quality software.

The closed laboratory was an ideal environment for students to practice software designing before coding. The closed lab could provide students step-by-step guidance and force them to conduct detailed design. With the help of laboratory instructors, it was easier for students to come out with a workable design. When mistakes were made or clarification was needed, immediate help was available from the lab instructor. As an outcome of a complete design, each student would have their data design, architecture design, object/package or algorithm design for their project.

Basically two types of designs were introduced. The procedural design was introduced at the beginning for small programs. The object-oriented design was used for large programs. It took time to teach object-oriented design and programming in the traditional classroom environment simply because of its complexity (the number of files and the length of the files). Given the package specification file, students could practice to develop the package body and then test it in the closed laboratory. Subsequently the package could be implemented in an open lab project. The advantage of doing this was that the defects of a package could be detected and corrected early in the closed laboratory with the help of the lab instructor.

The beauty of object-oriented technology is reusability. The package developed in the closed laboratory and stored in the library could be used repetitively in other projects later. For example, a Time_of_Day package was created and tested in the closed laboratory. The package converted seconds to hours, minutes and seconds, and displayed the system time or a randomly generated time. The package was used in a simulation open lab project to display a number of randomly generated times for a period of one hour.

2. Testing Plan

Testing is a major part of software development. In the industrial world, more time is spent in testing than in any other phase of software development. In the academic environment the sample testing data is usually given from the instructor for each lab assignment for grading purposes. Running a program with specific input and producing

the correct results only establishes that your program works for that input. Testing a program involves running it multiple times with various input data and observing the results. It is important to test your program with various kinds of input. A good testing plan can detect software defects effectively.

In the closed laboratory environment, students wrote a test plan for their project by filling out a test plan form (see below) after their project design,. Sometimes these plans were swapped with another student for testing and comments. The top part of the test plan form used by students was as follows:

Test Unit: _____

Description: _____

Test Run#	Test Data	Expected Result	Actual Result	Comment

3. Debugging

The closed laboratory provided the environment for students to learn debugging skills. We encouraged students to do code review before their first compile since most software defects result from simple oversights and goofs. Doing code review first will save a lot of compile time. To do a code review you study the printed source code to find errors. According to Humphrey's book [5] "A typical engineer will find only about 2 to 4 defects in an hour of unit testing but will find 6 to 10 defects in each hour of reviewing code." The key to conducting effective code review is having an effective code review procedure for personal use. We introduced Humphrey's Ada Code Review Guideline and Checklist for our students to use.

Using debugging tools in an Ada compiler is important but it is not taught in the traditional classroom or textbooks. The closed laboratory environment maked it possible to teach how to utilize those tools. The debugging activities introduced in the closed laboratory are listed as follows:

a. Using "Message box" in the ObjectAda IDE environment to debug syntax errors.
b. Using "Watch" in the ObjectAda IDE environment to trace values
c. Using "Step Over" in the ObjectAda IDE environment to debug logical errors.
d. Writing stubs to locate the logical errors
e. Using the run-time errors specification list to fix predefined run-time exceptions.

4. Personal Software Process

Personal Software Process **(**PSP) is a method used to help software engineer to track time and measure product quality. In Hou's PSP research experiment paper [6], she indicated "The PSP addresses quality and efficiency in software development directly. PSP users experience improvements in their ability to estimate the size and time it will take to build a component. They also greatly reduce their defect rates." Introducing PSP in the closed lab helps students understand software development process.

In the PSP lab students were given a program scenario in the closed lab environment and learned to use a time log to track the time they spent in project development by phase and used the data from the time log to fill out a PSP summary form. Students PSP summary forms can be summed up as follows:

	Planning	Design	Code	Compile	Test	Total Time	Hours
Student1	8	33	56	35	33	165	2.75
Student2	10	56	37	76	23	202	3.37
Student3	20	23	54	79	33	209	3.48
Student4	5	8	65	135	34	247	4.12
Student5	**5**	**7**	**57**	**145**	**45**	**259**	**4.32**
Student6	4	45	46	34	65	194	3.23
Student7	**12**	**32**	**57**	**16**	**12**	**129**	**2.15**
Student8	30	34	68	16	35	183	3.05
Student9	18	23	35	79	45	200	3.33
Student10	48	34	43	67	21	213	3.55
Maximum:	48	56	68	145	65	259	4.32
Minimum:	4	7	35	16	12	129	2.15
Average:	16	29.5	51.8	68.2	34.6	200.1	3.34

Note that student 5 spent less time on design and more time on compiling. On the other hand, Student 7 spent more time on design and less time on compiling. The table illustrates the importance of the design phase and stimulates student interest in the phases of software development.

PSP was introduced in Ada programming courses at Embry-Riddle University in 1995-1996 (See the publication Hilburn [7]). The problem was not enough time for students to conduct the process step-by-step in the allotted time. The closed laboratory overcame the problem.

IV. Administration and Assessment

Our typical closed laboratory had 18 students with one laboratory instructor. Twelve laboratories were implemented during a semester. One laboratory was conducted per week. Each student needed to attend to his/her registered session. Every student had a computer to do the work. Sets of programs and data files used for the lab activities were obtained from the network. Individual activities were not graded with a score, but attendance, completeness and accomplishment of activities were recorded as a factor in the final grade of the four-credit course. The output of each activity was checked and

initialized by the laboratory instructor. Laboratory manuals were distributed the week prior to the laboratory session so that students could have time to read and get prepared for the laboratory activities.

Students who could not finish in the allocated were required to finish them independently and turn them in to the laboratory instructor in the next week's closed laboratory session.

1. Staffing

Junior students were selected as laboratory instructors. They were recommended by the faculty of the department as responsible students with good academic status. Their responsibilities were to answer questions from students and check if each student performed activities correctly. For each lab there was a check-off sheet. At various points of each lab, students were required to show code, answer questions, or explain some behavior that they have observed. When a student reached one of these "check-off" points, the student signaled a laboratory instructor. The laboratory instructor initialed the corresponding entry on the laboratory check-off sheet. If the response was not correct or was incomplete, the laboratory instructor could help until the student was comfortable with the concept being explored.

2. Academic Assessment

At the end of each laboratory, students turned in their check-off sheets to the laboratory instructor. To receive full points from each laboratory session, each student must have attended and completed all the laboratory activity correctly.

V. Findings and Conclusion

The findings of the closed laboratory implementation are as follows:

1. The closed laboratory provided an ideal environment for the students to explore, learn and implement software development methods as well as for learning the Ada language.
2. The closed laboratory was a major contributor for students to understand the course contents and built a bridge between the class lectures and open lab projects.
3. Through the closed laboratory, students accomplished their open lab projects in a more efficient and professional way. As the result, each student developed his open lab project by following software development methods.

The impact on students was evaluated in an anonymous survey conducted at the end of the 1998 fall semester:

1. The closed laboratory is a major contributor to their understanding of course contents including software development methods and fundamentals of Ada programming.
2. The closed laboratory materials helped them to accomplish their open lab projects.
3. Students enjoyed the closed laboratory, considering it an exciting and interesting learning experience.

Sixty questionnaires out of seventy returned from the students. Table 1 below shows 52% of students strongly agree and 35% of students agree that the closed laboratory did help them understand their course materials better.

Table 1

Strongly agree	Agree	Not Sure	Disagree	Strongly disagree
31	21	4	3	0
52%	35%	7%	5%	0%

Table 2 below shows, 82% of students agree or strongly agree that the contents of the closed laboratory helped develop their open lab projects.

Table 2

Strongly agree	Agree	Not Sure	Disagree	Strongly disagree
24	25	7	3	0
40%	42%	12%	5%	0%

Table 3 below shows, 27% of students consider that the closed laboratory was excellent learning experience and 52% consider it as good learning experience.

Table 3

Excellent	Good	Average	Below average	poor
16	31	6	3	2
27%	52%	10%	5%	3%

These results indicate that emphasis on the software engineering process in the development and implementation of the Ada closed laboratory was successful. Overall, the implementation of the closed laboratory appears to have been accepted by students. In the closed laboratory environment, the students not only explored and experienced the traditional programming class, but also performed software engineering process activities that are almost impossible to do in the traditional classroom.

[1] Tuck, A. B. Editor, “Computing Curricula 1991: Report of the ACM/IEEE-CS Joint Curriculum Task Force.” Final Draft, December 17, 1991.

[2] Denning, P. J. “ Computing as a Discipline.” Communications of the ACM, Vol. 32, No. 1, pp. 9-23.

[3] Feldman, M. B. and Koffman, E. B. “Ada’95 Problem Solving and Program Design.” Addison-Wesley, 2nd Edition, 1997.

[4] Grove, R. F. "Using the Personal Software Process to Motivate Good Programming Practices." SIGCSE Bulletin, ITiCSE 98 Proceedings, Volume 30, Number 3, September 1998.

[5] Humphrey, W. S. “ Introduction to the Personal Software Process.” Addison-Wesley, 1997.

[6] Hou, L. and Tomayko, J. “Applying the Personal Software Process in CS1: an Experiment.” The proceedings of the Twenty-nine SIGCSE Technical Symposium on Computer Science Education, February 1997.

[7] Hilburn T. and Towhidnejad, M “ Doing Quality Work: the role of Software Process Definition in the Computer Science Curriculum” The proceedings of the Twenty-eight SIGCSE Technical Symposium on Computer science Education, February 1997.