Ada EDUCATION IN A SOFTWARE LIFE-CYCLE CONTEXT

Anne J. Clough
Ada Office
The Charles Stark Draper Laboratory, Inc.
555 Technology Square
Cambridge, Massachusetts 02139
(617) 258-2748

ABSTRACT

This paper describes some of the experience gained to date from a comprehensive educational program undertaken at The Charles Stark Draper Laboratory to introduce the Ada[1] language and to transition modern software engineering technology into the development of Ada and non-Ada applications. Initially, a core group, which included managers, engineers and programmers, received training in Ada. An Ada Office was established to assume the major responsibility for training, evaluation, acquisition and benchmarking of tools, and consultation on Ada projects. As a first step in this process, an in-house educational program was undertaken to introduce Ada to the Laboratory. Later, a software engineering course was added to the educational program as the need to address issues spanning the entire software life cycle became evident. Educational efforts to date will be summarized, with an emphasis on the educational approach adopted. Finally, lessons we have learned in administering this program will be addressed.

Introduction

Early in 1984, a laboratory-wide committee was set up at the Charles Stark Draper Laboratory, Inc. in Cambridge, Massachusetts, to assess the impact of Ada and the advances in software technology that this new DoD-mandated language would impose on the development of software. As a result of recommendations of this committee and support of upper-level management, a concerted effort is being undertaken to bring this technology in-house. A multi-level education and training program has been set up, Ada products are being evaluated and procured, consulting and support services are being provided as Ada projects become a reality at the Laboratory. This paper will concentrate on the education and training efforts to date.

---

[1] Ada is a registered trademark of the U.S. Government (Ada Joint Program Office).

At the heart of Draper's educational plan was the formation of a small, highly motivated and qualified group of individuals responsible for supporting the introduction of Ada technology throughout the Laboratory. A team of instructors from Raytheon/Mid-Atlantic Systems Facility and Raytheon/Equipment Development Laboratories assisted in this effort. Two courses were offered - a 16 to 20-hour Fundamentals of Ada tutorial for managers and an 80-hour Designing and Programming with Ada course for engineers and designers. Twenty managers and thirty engineers/designers participated in this initial phase. This group, chosen from a wide cross-section of projects in the Laboratory, continues to provide support to Ada activities. An Ada Advisory Committee chosen from this core group provides essential advice, feedback and support to the overall effort.

In order to coordinate, plan and implement all Ada-related activities, an Ada Program Office was established. Education, training, and the acquisition of basic tools were first priorities. Video courses and computer-aided instructional aids were evaluated and purchased to supplement more formal education. An in-house course was developed, and compilers and other support tools were evaluated and acquired. In addition, the Ada Office has followed closely and participated in the larger Ada community and publishes an Ada newsletter to keep the Draper technical staff informed of developments in this area. Figure 1 presents the initial plan for the acquisition of Ada technology at the Laboratory, and in fact, quite accurately describes what has happened during the past two years.

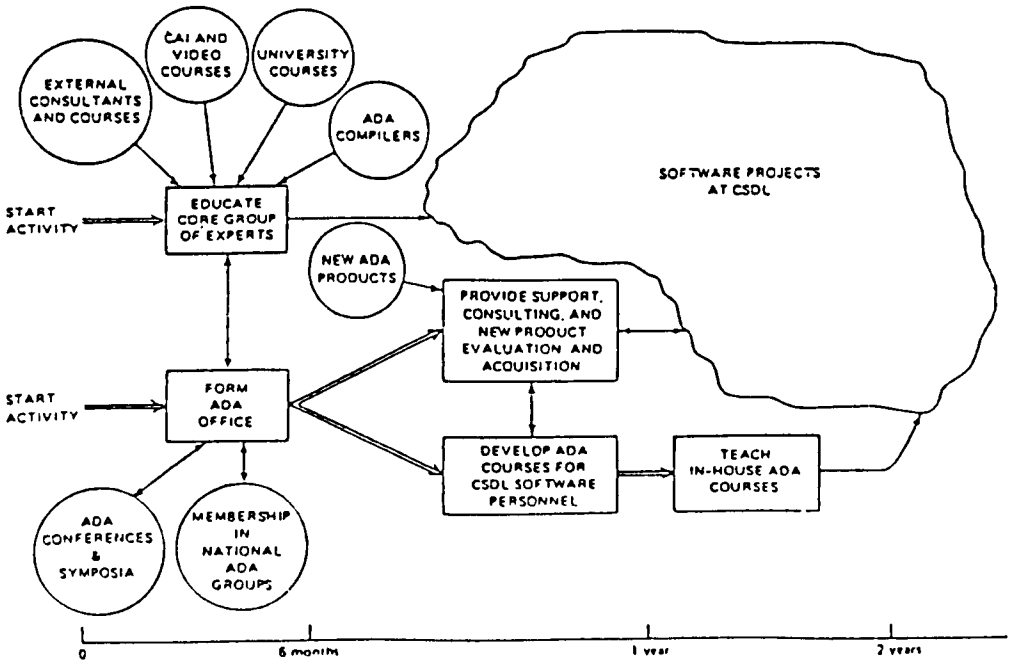

FIGURE 1. ADA TECHNOLOGY PLAN OVERVIEW

Developing an In-House Ada Curriculum

Because Ada is a very large language and at times complex, it was felt that traditional training techniques might not prove adequate. A three-tiered method was adopted which essentially takes a top-down approach to introducing the language. The first "pass" through the language presents an overall view. It concentrates on the need for a new approach in developing software and presents the history, development environment, and features of Ada. Initial exposure concludes with a look at simple, but complete, examples. The second pass studies Ada's features in more detail, but still does not emphasize syntax or grammar rules, or the more obscure, difficult, or infrequently used aspects of any language feature. A third and final pass then carefully examines each feature in detail with sufficient time allowed for discussion, questions, and programming practice.

In practice, this approach has proved to be very effective for several reasons. First, because of the structure of the course, it is possible for students to choose the level of participation desired. Participants who attend the first portion of the course receive an overview of the goals and features of Ada. Administrators, for example, often choose this level and find it appropriate for their purposes; they can exit the course with a cohesive set of knowledge. Those attending the first two segments of the course will learn to develop and recognize high quality software design in Ada from a conceptual viewpoint, rather than with an emphasis on detailed rules. This might be an appropriate level of detail for software project managers. Those participating in the entire course receive thorough hands-on training in the effective use of Ada, an essential requirement for the software practitioner.

A second reason that this approach proved effective is the direct result of the richness and complexity of the language. It is necessary to understand language features at a high level. "Why do we have this feature?" "How will it benefit me as a developer of software to be able to use this feature? "Where - in what context - will it be used?" If the instructor is not careful to address these issues at the beginning, it becomes very difficult to differentiate the forest from the trees, or lose sight of the trees themselves while we focus on a small portion of one tree. In addition, the very fact that we "visit" a language feature at least three times during the entire course makes the practitioner ultimately comfortable with that feature. Initially, he/she may be struggling with the concept itself ("just what is a generic?"), but ultimately it becomes familiar and the software developer can begin to realize and appreciate the extra capabilities that many of these unfamiliar Ada features provide to the developer.

Texbooks selected for this course are: "Software Engineering with Ada" by Grady Booch and "Programming with Ada" by J. G. P. Barnes. These are supplemented by pertinent articles and materials throughout the course. The bibliography at the end of this paper lists some of the materials that have been used both in this course and in a separate software engineering course.

Homework is an integral part of the course. Students design and implement Ada applications of increasing complexity as the course progresses. Though first sessions of the in-house course and the core course that preceded it were hampered by the lack of a validated compiler or even a compiler that could handle the full Ada language, the availability of a DEC VAX/VAX compiler now makes assignments more meaningful. Certainly hands-on work using a competent, fully-validated compiler is essential. Certificates are awarded to all participants in the course who satisfy homework requirements. This certificate is added to their personnel records, thus providing more incentive to complete all homework assignments and enabling the Laboratory to identify those staff members with Ada expertise.

Sixty hours of instruction are required for the entire course. Classes meet for 2 1/2 hours two mornings a week during working hours. Three sessions of the entire course have been given - approximately 110 people have participated, 45 have completed the full course.

Developing a Software Engineering Curriculum

Ada education at Draper Laboratory is very definitely software engineering with Ada. The emphasis throughout is on "engineering" software for large systems and all features are introduced and taught in that context. Ada, of course, is unique in that it has been expressly designed with features to encourage modern programming and software engineering practices. Designed for portability and reuse, providing effective encapsulation and data abstraction facilities, Ada has the potential to substantially change the way software is produced. As such, it is imperative that the importance of software design, the development of an appropriate Ada style, and the proper use of this language be emphasized in any Ada educational effort. Developing the "Ada mind-set" is important. As emphasized by many Ada experts and practitioners, a syntax-driven educational approach will not work and will most likely produce poorly constructed programs, disappointing results, and consequently negative feelings about the language itself. Software engineering therefore becomes a priority in our educational efforts throughout the entire Ada course, with each language feature discussed within this context. In addition, special sessions deal with Ada as a program design language, object-oriented design techniques, and investigating whether or not, and how well, Ada does meet the goals of software engineering.

Having emphasized that our Ada educational approach is heavily software engineering driven, it is nevertheless necessary to assert that one course cannot do it all. It is not possible to provide in a single course of any reasonable length a complete treatment of Ada and a comprehensive treatment of software engineering at the same time. Nothing less than changing the model of software design, development and maintenance acquired from previous language experience will suffice. Each sequential phase of the life cycle must be evaluated in terms of what

skills are required for effective and efficient production of software and the proper use of Ada.

The Ada course introduced software engineering concepts that may not have been consciously considered by students before that time. However, the need for more software engineering knowledge became apparent. To that end, comprehensive software engineering training, not foreseen in the original Ada program plan, is being developed by the Ada Program Office.

A software engineering course which deals with the entire life cycle has been added to Draper's educational program. Topics ranging from system definition, software costing and software standards to requirements analysis, design, testing, maintenance and configuration management are covered. Tools that can aid or automate various portions of the life cycle are presented.

The course was initially conceived as having a complete Ada orientation, both because it grew out of the Ada course and because it is being developed by the Ada Program office. However, widespread interest in software engineering by both Ada and non-Ada software developers led to a course that has both language-independent and Ada-dependent portions.

An integral part of this course is a workshop that allows participants to apply both software engineering principles and Ada implementation techniques to a real application as the course progresses. A space station command and control problem, adapted from an application designed and implemented for MITRE Corporation by a Boston University, College of Engineering student team,[2] was used for this purpose. An exercise had to be chosen that could be completed in a three-month time span but yet would be interesting enough and challenging enough to motivate the workshop members. Teams of approximately eight members each are given the documentation that has resulted from the system definition and scheduling phase of a project . This documentation is not complete; therefore one of the first things each team must do is get back to the "customers" -- (the instructors in this case) -- and fill in the gaps that remain in the system description. Each team then develops the application -- conducts requirements analysis, designs the software architecture, does low-level algorithmic design, codes and tests the solution. At this point, the two teams swap software and documentation, and each verifies the other team's software. Since the application is developed in Ada, the design portion of the course concentrates heavily on design methodologies and techniques suitable for developing Ada applications. Software requirements reviews, preliminary design reviews, detailed design reviews as well as testing and final reports are presented during regularly scheduled class sessions so that all members of the class can benefit from seeing the application progress through all stages of the life cycle.

---

[2] Ruane, Michael F. and Vidale, Richard F., Assessing Ada: Implementation of Typical Command and Control Software Functions.

Each presentation of a life cycle topic is completed before the workshop group begins work in that portion of the life cycle. Classes meet for 2-1/2 hours two mornings a week during working hours for thirteen weeks. The workshop then continues for an additional month at which time the entire class reconvenes to review testing and final reports by the workshop participants. The workshop schedule mirrors a 30-30-15-25% life cycle model -- one month for requirements analysis, one month for design, 2 weeks for coding and 3 weeks for testing.

As in the Ada course, members can choose their level of participation consistent with their own requirements and schedules. A participant can take part in the language independent portions only or in the entire course with or without the workshop. Exercises are provided so that all participants, whether or not they are members of the workshop, will gain experience applying the concepts that are presented. Certificates will again be presented to indicate participation and fulfillment of course requirements.

Lessons Learned -- Ada Education

A very pleasant outcome of the Ada effort thus far is an ever-growing group of people within the Laboratory who are being exposed to Ada and who are becoming enthusiastic about the language. This group includes people at all levels and across a wide variety of application areas. Many were frankly skeptical initially and have been impressed by Ada and its power and promise, especially in the area of the mission-critical embedded systems that are an important part of the Laboratory's activities.

At this point, we have had enough experience in Ada education that we can begin to assess its effectiveness. We can look critically at our course materials and see where they have been successful and where improvement is needed. We listen carefully to the comments of our students and attempt to tailor this course so that it meets our current and future needs. Some of what we have learned in this process follows.

In the Ada course, two areas of difficulty for the beginning student have caused us to make adjustments in the presentation of course material. The first, the strong typing of Ada, which is initially frustrating, actually becomes one of the first pleasant surprises for the student. Ada allows us, actually urges us, to define our own data types. An object is given a type when it is declared. Thereafter, an object's type is invariant throughout program execution. Values of one type cannot be assigned to variables of another type. Standard operators cannot be used with variables of different types. For the student accustomed to working with languages that do not have strong typing features, this seems very restrictive and he is at least initially annoyed every time the compiler flags a typing error and makes him explicitly convert values from one type to another before an operation can be performed or an assignment statement can be executed. However, the first time that the compiler catches a typing error that would have formerly

slipped through and become an elusive bug in a less strongly-typed language, a new convert to strong typing is won.

The second difficulty for the new student involves the input/output feature of Ada. This has necessitated more emphasis on input/output early in the course in recognition that even simple programs need some rudimentary I/O. Since input/output in Ada uses packages and generics as well, I/O can be confusing and can seem needlessly awkward to a student accustomed to working with a language with built-in input/output facilities. Time spent familiarizing the student with exactly how this works in Ada not only eases his frustrations but also provides him with an example and model of an use of packages and generics. This can be very helpful in understanding and using these concepts later on.

The problem of presenting topics in an optimum sequence is not a trivial one, both in terms of maintaining class interest and applying the concepts presented. As an example, in order to cover Ada types completely, much material must be presented. However, if some effort is not made to disperse this material throughout the detailed portion of a course, rather than present it in a single block, it will surely be difficult to maintain interest. The "divide and eventually conquer" approach to Ada's typing topics also benefits the student when derived types are presented at enough distance from the concept of subtypes so that the two do not become hopelessly muddled. An additional consideration is that of allowing sufficient time to apply those features covered at the end of any course. This can be a serious problem if tasking is the last topic presented as is the case in many Ada curriculums. Since most programmers tend to think in a sequential manner and tend to have the most difficulty dealing with concurrency and the issues concurrency raises, putting this topic to the end of a course will not give the student sufficient time to apply these new concepts. Not only will students be unable to appreciate Ada's tasking facility, but they will also have real hesitancy to use this feature at all when beginning to design systems in Ada.

We have found that students at all levels want more complete and concrete examples of good Ada systems. "Real" work-related examples are especially helpful. The experienced programmer wants to concentrate on the unique features of Ada -- he prefers to learn on his own the simple statements, constructs and expressions that are similar to those found in most high order languages. Students would like each Ada construct to be accompanied by many examples of its use -- the Language Reference Manual syntax format supplemented by many more examples would be useful. The Ada-unique packaging and generic features have proven to be accessible to most students who very quickly perceive their power and begin to use these features effectively. Exception handling, and the way Ada implements it, always initiates lively discussion. While quite valid concerns about the misuse of this feature are often expressed, students soon produce code that uses the exception handling feature effectively.

Tasking is perhaps the most difficult feature for new students of Ada. Many traditional languages do not have features that allow parallel processing. Because of this, most programmers have a great deal of experience -- or all of their experience -- in sequential programming.

This lack of experience in programming concurrent processes, coupled with the unique problems that can arise such as deadlock, starvation and timing considerations, make this feature a difficult one to both teach and learn. It has been necessary to expand this portion of the curriculum. A tool developed at Draper, which graphically shows tasks operating concurrently and explicitly shows such things as actuation, suspension of tasks, rendezvous, and termination, has helped the educational effort in this area. However, an advanced Ada course which would concentrate in large part on tasking should perhaps be considered.

Lessons Learned -- Software Engineering Education

Although we have not had as much experience with the software engineering course as with Ada education, the first session of this course has been very successful. Participation, as has also been the case in our Ada educational efforts, has included a wide cross-section of the Laboratory both in terms of application areas and job level. Managers are participating both in the course and in the workshop, as are entry-level engineers and programmers. A great deal of enthusiasm centers around the workshop approach as this provides a convenient mechansim to apply techniques and tools discussed in class in an essentially "no-risk" situation. There is a great deal of learning that takes place in the workshop groups, as people with diverse backgrounds and experience are taking part. In the classroom as well, much information exchange is taking place and a wide range of expertise is being tapped. This combination has resulted in a very effective learning forum. The Ada Program Office is coordinating the course and supplying most of the instruction; however, a number of presentations given by experts both within the Draper community and outside as well, have greatly enhanced the course offerings. Through the very active participation of its members, all participants in the course are being challenged to think about the way they are currently developing software. In addition, any new methods being presented, whether they be requirements analysis, design or testing methods, are subjected to the most rigorous scrutiny. "Will this method work as advertised by its proponents?" "Will it work in the type of application that I develop?"

As mentioned earlier, participants can choose their own level of participation. Though course developers had assumed that members who had no familiarity with Ada would choose to participate in the language-independent portions only, in actuality most members have opted for the entire course. Because of this, several sessions were added to familiarize non-Ada participants with Ada's unique features. An unexpected side effect appears to be a group of people interested in registering for our next Ada course.

Have we presented these two courses in the correct order? Shouldn't a software engineering course precede a course in Ada? Although this will be the case for the group of people just mentioned, in general, the opposite approach has worked quite well. First of all, the Ada course has a good amount of software engineering content. In addition, having

the majority of course participants conversant with Ada has enabled us in this second course to consider a number of design methodologies uniquely suited to Ada, has enabled us to conduct a non-trivial workshop in Ada and has allowed us to deal with some of the more advanced and difficult aspects of the language, especially in the tasking area.

Future Plans

As Ada applications continue to be introduced into the Laboratory, the Ada Program Office will continue its efforts in education and its efforts to provide a more supportive programming environment. In Ada itself, an advanced course concentrating heavily on the tasking aspects of the language and providing more guidance on developing embedded applications may need to be added to the curriculum already developed. Utilizing the low-level features of the Ada language may need closer examination as well. For the course already "on the shelf," tuning and tailoring for Draper's particular requirements will be a continuing process. The top-down, three-level approach has proved quite effective. Perhaps a separate course for administrators or a separate course for managers will need to be given at some point in the future -- our essentially modular approach would make that very easy to prepare. Continuing seminars sponsored by the Ada office provide an opportunity for those who will not be using Ada immediately to keep their Ada skills up to date and enable those presently involved in Ada applications to keep informed about new Ada methodologies, techniques and tools.

Software engineering will continue to be emphasized. Growing interest within the Laboratory ensures a repetition of the software engineering course discussed in this paper. In future sessions, different applications may be given to each team so that, in the testing phase, teams can test applications that they have not developed. Since the major thrust of the software engineering course is on the requirements analysis, design, implementation and testing portions of the life cycle, further courses or intensive seminars could be added on the system definition and scheduling phase. The software planning phase and software cost analysis could be covered in more detail. Review techniques, maintenance, security and configuration management are other possible topics for future in-depth coverage. Possibilities for further growth in training and education surely exist.

Conclusions

Experiences in education and training at Draper Laboratory illustrates the effectiveness and long-term benefit of establishing an in-house capability in this area. Many training offerings are available that provide intensive, short-term training in Ada; fewer offerings are available in software engineering. The long-term effect of some of these offerings is often questionable. Certainly a five-day or two-week

intensive "hands-on" approach to teaching Ada will not really allow students to either become comfortable with the new concepts presented or to grapple with the more difficult issues. A course spread over a longer period of time -- our courses traditionally have a 3-4 months span -- allow the student time to assimilate new ideas, raise questions and most importantly get real hands-on experience with non-trivial applications. In addition, having in-house support for Ada and software engineering ensures that, long after a course has been completed, the instructor or instructors are available for consultation and assistance. This latter advantage cannot be overemphasized when new technology is being introduced if the desire is to truly assimilate and integrate that technology into the software development process.

BIBLIOGRAPHY

1. Albrecht and Gaffney, "Software Function, Source Lines of Code and Development Effort Prediction: A Software Science Validation", IEEE Transaction on Software Engineering, Vol. SE-9, No. 6, November 1983, pp. 639-648.

2. Ausnit, Cohen, Goodenough & Eanes, "Ada in Practice", Springer-Verlog, 1985.

3. "An Object Oriented Design Handbook for Ada Software", EVB Software Engineering, Inc., 1985.

4. Barnes, J.G.P., "Programming in Ada", Addison-Wesley Publishing Co., 1984.

5. Boehm, B., "Software Engineering Economics", Prentice-Hall, 1981.

6. Boehm-Davis & Ross, "Approaches to Structuring the Software Development Process", General Electric Company, October, 1984.

7. Booch, Grady, "Software Engineering with Ada", Benjamin/Cummings Publishing Company, Inc., 1983.

8. Brooks, Frederick, "The Mythical Man-Month", Addison-Wesley Publishing Company, 1975.

9. Buhr, R.J.A., "System Design with Ada", Prentice-Hall, Inc., 1984.

10. Fairley, Richard, "Software Engineering Concepts", McGraw Hill Book Company, 1985.

11. Freeman & Wasserman, "Tutorial on Software Design Techniques", IEEE Computer Society Press, 4th Edition.

12. Helmbold & Luckham, "TSL: Task Sequencing Language", Proceedings of the Ada International Conference, 1985, Cambridge University Press, Cambridge, pp. 255-274.

13. Mardrioli, Zicari, Ghezzi and Tisato, "Modeling the Ada Task System by Petri Nets", Computer Language, Vol. 10, No. 1, pp. 43-61, 1985.

14. Myers, Glenford, "The Art of Software Testing", John Wiley & Sons, 1979.

15. Pressman, Roger S., "Software Engineering: A Practitioner's Approach", McGraw-Hill Book Company, 1982.

16. Ruane, Michael F. & Vidale, Richard F., "Assessing Ada: Implementation of Typical Command and Control Software", Boston University, College of Engineering, Boston, MA, 1984.

17. Snaufer, Joseph B., "Practical Guidelines for Testing Ada Programs", Master's Thesis, Arizona State University, 1985.

18. Szulewski & Sodano, "Design Matrics and Ada", Proceedings of the 1st Annual Washington Ada Symposium, Sponsored by ACM, 1984, pp. 105-114.

19. Wegner, Peter, "Self-Assessment Procedure VIII", Communications of the ACM, Vol. 24, No. 10, October, 1981.

20. Weiner & Sinovec, "Software Engineering with Modula-Z and Ada", Wiley & Sons, 1984.

VIDEO TAPES

1. Ichbiah, Barnes and Firth on Ada, Alsys, Inc., 1984.

2. Software Engineering Training Curriculum, R.S. Pressman & Associates, Inc., 1985.

CAI

1. Lessons on Ada, Volume I and II, Alsys, 1983 and 1984.