LESSONS LEARNED IN USE OF ADA^{TM}-ORIENTED DESIGN METHODS

Carolyn E. Brophy+, W. W. Agresti* and Victor R. Basili+

+ Dept. of Computer Sciences
University of Maryland
College Park, Maryland 20742

* Computer Sciences Corporation
System Sciences Division
8728 Colesville Road
Silver Spring, Maryland 20910

Abstract

As Ada is introduced into new environments, both managers and developers need to understand the ways in which the decision to use Ada as the target language will affect the software development lifecycle. The Flight Dynamics division at NASA Goddard Space Flight Center is involved in a study analyzing the effects of Ada on the development of their software. This project is one of the first to use Ada in this environment. In the study, two teams are each developing satellite simulators from the same specifications, one in Ada and one in FORTRAN, the standard language in this environment. This paper will address the lessons learned during the design phase including the effect of specifications on Ada-oriented design, the importance of the design method chosen, the importance of the documentation style for the chosen design method, and the effects of Ada-oriented design on the software development lifecycle. It is hoped that the issues faced in this project will show more clearly what may be expected in designing with Ada-oriented design methods.

Introduction and Experiment

As Ada is introduced into new environments, the need arises for both managers and developers to understand the ways in which the decision to use Ada as the target language will affect the software development cycle. This becomes an especially important issue for NASA, who is planning to use Ada for its space station project. In doing this, NASA needs to understand the effects of Ada on their traditional software development approach.

The experiment in progress is being conducted by the Software Engineering Laboratory (SEL) of the National Aeronautics and Space Administration's Goddard Space Flight Center (NASA/GSFC). NASA/GSFC and Computer Sciences Corporation (CSC) are cosponsors of the experiment, which is supported by personnel from all three SEL participating organizations (NASA/GSFC, CSC, and University of Maryland) [Agresti et al. 86].

The basic goals of the whole study [McGarry & Nelson 85] are to

(1) characterize the development process with Ada, and

(2) determine the impact the use of Ada will have on reusability, reliability, maintainability, productivity, and portability.

Two teams are each developing a Gamma Ray Observatory (GRO) satellite dynamics simulator from the same specifications. One team is using FORTRAN, as usual, as the target language. The other team is using Ada. The purpose of the GRO dynamics simulator is to test and evaluate GRO flight software under conditions that simulate the expected in-flight environment as closely as possible [Agresti et al. 86].

Both teams began in January, 1985. The Ada team however, had a three or four month training period before beginning development. The software development lifecycle as it applies to this project is shown in Figure 1 [Agresti 86], [McGarry, Page et al. 83].

The Ada team is now in the midst of the implementation phase. The FORTRAN team's simulator is almost ready to enter production use.

Ada can be viewed in two ways. We can look at it as a programming language only, or we may view it as an orientation toward a new approach to problem solving. In the former context, the changes only affect the phases from coding on. As a new approach to problem solving, all phases of the software development cycle are affected, beginning with the specifications. A previous study [Basili et al. 85] showed that developing an Ada product from a set of specifications that were written knowing that FORTRAN would be the implementation language lends itself to a FORTRAN-style design. Changing to Ada in this latter case requires a break from the mechanisms of the past.

In giving up their traditional language, FORTRAN, NASA is giving up a legacy of reusability. This reuse exists in their specifications, design, and code. So it is hoped that with the adoption of Ada they will gain something. For example, can Ada provide a new and

Ada is a trademark of the U.S. Department of Defense - Ada Joint Program Office.

Contact: Carolyn Brophy, Dept. of Computer Science, University of Maryland, College Park, MD 20742, (301) 454-6154.

Support for this research provided by NASA grant NSG-5123 to the University of Maryland.

better legacy of reusability, not only as a programming language, but also as a design methodology with its appropriate set of documents and orientation? We are interested in learning what are the advantages and disadvantages of an Ada development and maintenance orientation. So we have embarked upon a study which threw away the old FORTRAN legacy, and with it, the reuse leverage of the past.

Questions we are attempting to answer in this study include: How might the requirements be represented so as to avoid the FORTRAN legacy? What is the appropriate design technique, knowing that the implementation language will be Ada? What design method is appropriate for the specific application, and can it be scaled up to the problem size? Is it teachable and usable by the existing staff? What kind of training is needed? Can it be documented? What are the milestones needed for an Ada development?

Because of the nature of this experiment, we have a clean start, rewriting the specifications, refining and adapting a new design approach, and developing new forms of documentation. There is a motivation for innovation that might allow for the development of better specifications, design, code and documentation because of the experimental nature of this study. We want to record the successes and failures, advantages and disadvantages, and capture them in a lessons learned document so that future developments can gain from our experience. The SEL is preparing a document that will cover the entire lifecycle with respect to the lessons learned on this project.

The information for this paper is from a survey given to the Ada designers after the design was completed. The rest of this paper consists of a list of "lessons learned", each followed by the Ada team experiences which led to that particular conclusion.

Seven Design Lessons

1. Choose a specification method that does not constrain design.

Presently in this environment, the specifications the development team receives are heavily biased toward FORTRAN. In fact the high level design for the simulators is actually in the specifications document, and has not changed for several years. Therefore, to really explore various design methodologies, the Ada team found they had to rewrite the specifications to remove the bias toward FORTRAN and the whole FORTRAN legacy. The specifications were rewritten using the Composite Specification Model [Agresti 84].

It was at this point when the highest level of the design began to take shape. The problem domain lends itself well to an object oriented view, so problem solving proceeded along this line.

Team members felt that the resulting specifications were language neutral. The team had not yet had extensive experience with Ada, and this particular specification method pre-existed Ada. New specifications freed the team from the FORTRAN oriented design built into the original specifications. One person felt that even the new specifications had a design bias built in. However this one was an object oriented one, and it was felt that it did not limit development with Ada.

The team felt that rewriting the specifications increased their understanding of the problem more than merely analyzing the original specifications would have done. One person said one additional consequence of rewriting the specifications was that this also prevented them from postponing some important questions until implementation, which would have meant major design changes at that point.

It seems clear that new Ada developments will require more time up front in the requirements, specification and design phases. However, this extra effort should pay off because of the deeper understanding of both the problem and the solution domains. This yields a higher quality product, better documentation of these earlier phases, and a cost savings during testing and maintenance.

2. Choose a method that exploits new Ada features.

If a methodology does not do this, why use Ada rather than another language? Thus much of Ada's benefits stem from packages, tasks, and generics which are central features distinguishing Ada from most other languages.

One of the study objectives was to experiment with various design methodologies. The Ada team did high level designs with three [Agresti, Brinker, Lo et al. 85]. They used structural decomposition, Cherry's PAMELA [Cherry 85], and Booch's object oriented design [Booch 83]. They found that structural decomposition did not encourage use of Ada's unique features at all. PAMELA, which was designed for use with embedded systems, was viewed as too oriented toward concurrency for this application. Booch's object oriented design methodology did not provide enough guidelines in its representations for a project this big. It left too much up to the designer's judgement.

As a result, the team developed their own object oriented methodology, which incorporates ideas from both Cherry's and Booch's methods [Seidewitz 85], [Stark & Seidewitz 86], [Seidewitz & Stark 86]. The methodology produces object diagrams as the final result of object/data flow analysis. Two orthogonal hierarchies exist:

(1) parent-child hierarchy (object decomposition)

(2) seniority hierarchy (an object using services of another is senior to the used object).

The new object oriented methodology maps very well into Ada, as both are developed with modern software engineering concepts in mind (e.g., data abstraction, information hiding). Objects easily convert to packages, and packages encourage modularity.

One of the successful results from the design is the modularity. The team felt this helped make interfaces easier to design, and increased interface reliability is expected at testing. Another important effect of modularity in the design is the ease of adding new programmers to the project and phasing out other programmers if required.

Another successful point is that the original design is still being followed in implementation, without major changes. The changes that have been made are additions. The team now feels that enough attention was not given to type specifications during design. However, it was felt the object diagrams were quite helpful as a framework for discussing proposed changes.

3. Team needs to know different design methods to converge on an appropriate design.

Most programmers/designers in this environment use functional decomposition as their design method. Part of the training for the Ada team was the use of other design methodologies. Cherry's PAMELA and Booch's object oriented design methodologies are radically different from the standard procedural decomposition used in this environment. Such exposure was one source of broader insight into problem solving for the team. Thus, including various design methodologies in training, especially the one to be used for that project, is very important. This is needed to really exploit Ada's features; it is not enough just to know the language.

An appropriate design both exploits Ada's features and makes implementation easier. We have already discussed the first issue. Concerning the second issue, the team has found that implementation was significantly promoted by their design. This design in turn was developed from their design methodology, which owes much to other methodologies as well. It was easy for a programmer to code from the design documents also. This was true even when the coder was not the designer for that section of the project. This has an important benefit in that it permits the build up of staff during the coding process allowing parallel development. In a project with tight schedule and high people resources, managers may be able to increase the staffing to minimize time.

4. Pay attention to how the design is documented.

Object diagrams (see Figure 2) are the key type of documentation produced by the team's object oriented methodology. Structure charts are the documentation produced with the standard FORTRAN design process.

Lack of a specific methodology at the start of the project was a problem for the team, though unavoidable in this case because of the objectives of the study. The representations changed over time as the methodology developed, which was a big problem, since it made it difficult to keep the design documents consistent. To apply a methodology well, everyone needs to know the ground rules at the start. This facilitates understanding between developers on the team, as well as between the team and managers.

The key issue here is the importance of people's expectations in what they see. Less precision in the structure charts and FORTRAN presentations at the Preliminary and Critical Design Reviews has been more acceptable than would be allowed with Ada documentation. Since the representations are so different for the Ada documentation, any unspoken understandings and intuition is lost.

Managers found they could not understand the object diagrams at these reviews. They tried to look at them as though they were the familiar structure charts, and could not visualize the design. Object diagrams contain a high level of detail in order to express all the relationships they are capable of expressing. If some type of modification was made to suppress details of relationships between modules so that some relationships could be shown between a greater number of modules, the gap between object diagrams and structure charts would be lessened.

Even so, training is needed to make the object diagrams familiar to managers and customers. Unfamiliarity leads to concerns that something is being hidden. The developers get less feedback on the design as well, when the design is not understood due to the representation.

One clear implication of this experiment is the need for education of the managers and customers in both Ada and the new concepts of software engineering. An Ada-oriented development requires a fair amount of knowledge on the part of the reviewers. There is both more and different types of information to examine to validate each of the phases of the lifecycle.

5. Designing with Ada may imply different starting and ending points of the design phase.

The legacy is that the starting point for design is a specifications document already containing the preliminary design. As we have seen, a preliminary design oriented toward FORTRAN would severely limit an Ada design because it would not take advantage of Ada's features. In this case therefore, with the specifications rewritten, less design existed in the specifications document. But since some design is there and this is also unfamiliar, the line where requirements analysis of the specifications stops and the design phase begins seems fuzzy.

The milestones of the design phase may also be different. In the usual software lifecycle with FORTRAN, it was well accepted what a Preliminary Design Review (PDR) and Critical Design Review (CDR) are. The breaks between lifecycle phases seemed logical and real. However there is no direct conversion for Ada, since the new object oriented methodology and its documentation is so different from the traditional ones. Again, preliminary design seems to fade into detailed design, and detailed design fades into coding. This made PDR and CDR seem to come at arbitrary times rather than at logical points in the design process.

The team was divided in how prepared they felt for PDR and CDR. One team member in particular felt more prepared for these than usual because he understood the design and its implications so well. Others felt less prepared than usual due to the newness of the methodology and representations, and unsureness of how to map the state of the design into the sorts of things generally expected at PDR and CDR.

One might consider the PDR occurring later than normal but with more rigor. The PDR could be represented by high level compilable design elements and CDRs might be staged for different design elements by examining more detailed Ada PDL pieces.

6. Ada gives an opportunity for compilable design elements

Ada can work well as a compilable program design language (PDL). The PDL used with FORTRAN is pseudocode. The advantage of compilable PDL is of course, that interface checking and type checking may be done, which helps assure validity of the design in a way otherwise not possible at this early a stage. To do this requires more precision in the design process than the standard FORTRAN design process now takes. However it also provides more assurance and confidence during the PDR and CDR.

7. Costs money not to reuse previous designs.

This whole discussion speaks of the real cost of a changeover to Ada as being the legacy accompanying FORTRAN which is lost. This is the case when Ada is viewed not merely as a programming language, but as including a whole new problem solving "world view". This legacy includes old specifications, old design, old code, intuition, and institutional knowledge which is not recorded anywhere. The Ada team found itself facing a bewildering number of questions needing to be answered again for Ada, as well as brand new ones, once they began using this new technology.

Summary

When Ada is designated at the start as the language of choice, it may influence many aspects of design. Observation of this project is continuing. As the study proceeds we will be very interested in seeing if what is gained makes this lost legacy worth losing.

Acknowledgements

The Ada experiment is managed by F. McGarry and R. Nelson of NASA/GSFC. The authors would like to thank them and the Ada team for their cooperation and assistance.

References

[Agresti 84]
Agresti W., "An Approach to Developing Specification Measures", *Proceedings of Ninth Annual Software Engineering Workshop,* Goddard Space Flight Center, Greenbelt, MD 20771, November 1984.

[Agresti 85]
Agresti W., "Ada Experiment: Lessons Learned (Training/Requirements Analysis Phase)", Goddard Space Flight Center, Greenbelt, MD 20771, August 1985.

[Agresti 86]
Agresti W., "SEL Ada Experiment: Status and Design Experiences", *Proceedings of Eleventh Annual Software Engineering Workshop,* Goddard Space Flight Center, Greenbelt, MD 20771, December 1986.

[Agresti, Brinker, Lo, et al. 85]
Agresti W., Brinker E., Lo P., et al, "GRO Dynamics Simulator in Ada (GRODY) -- Preliminary Design Report", Goddard Space Flight Center, Greenbelt, MD 20771, December 1985.

[Agresti et al. 86]
Agresti W., Church V., Card D., et al, "Designing with Ada for Satellite Simulation: A Case Study", *Proceedings of First Annual Symposium on Ada Applications for the NASA Space Station,* Houston, Texas, June 1986.

[Basili et al. 85]
Basili V.R., Katz E.E., Panlilio-Yap N.M., Ramsey C.L., and Chang S., "Characterization of a Software Development in Ada," *IEEE Computer,* Vol. 18, No. 9, Sept. 1985, pp. 53-65. September 1985.

[Booch 83]
Booch G., *Software Engineering with Ada.* Menlo Park, California: Benjamin/Cummings Publishing Co., Inc., 1983.

[Cherry 85]
Cherry G.W., "Advanced Software Engineering with Ada -- Process Abstraction Method for Embedded Large Applications", Language Automation Associates, Reston, Virginia, 1985.

[McGarry, Page et al. 83]
SEL-81-205, "Recommended Approach to Software Development", McGarry F., Page J., Eslinger S., Church V., and Merwarth P., Goddard Space Flight Center, Greenbelt, MD 20771, April 1983.

[McGarry & Nelson 85]
McGarry F., and Nelson R., "An Experiment with Ada -- The GRO Dynamics Simulator Project Plan," Goddard Space Flight Center, Greenbelt, MD 20771, April 1985.

[Murphy & Stark 85]
SEL-85-002, "Ada Training Evaluation and Recommendations from the Gamma Ray Observatory Ada Development Team", Murphy R., and Stark M., Goddard Space Flight Center, Greenbelt, MD 20771, October 1985.

[Seidewitz 85]
Seidewitz E., "Some Principles in Object-Oriented Design", Goddard Space Flight Center, Greenbelt, MD 20771, August 1985.

[Stark & Seidewitz 86]
SEL-86-002, "General Object Oriented Software Development", Seidewitz E., and Stark M., Goddard Space Flight Center, Greenbelt, MD 20771, August 1986.

[Seidewitz & Stark 86]
Seidewitz E., and Stark M., "Toward a General Object Oriented Software Development Method", *Proceedings of First Annual Symposium on Ada Applications for the NASA Space Station,* Houston, Texas, June 1986.

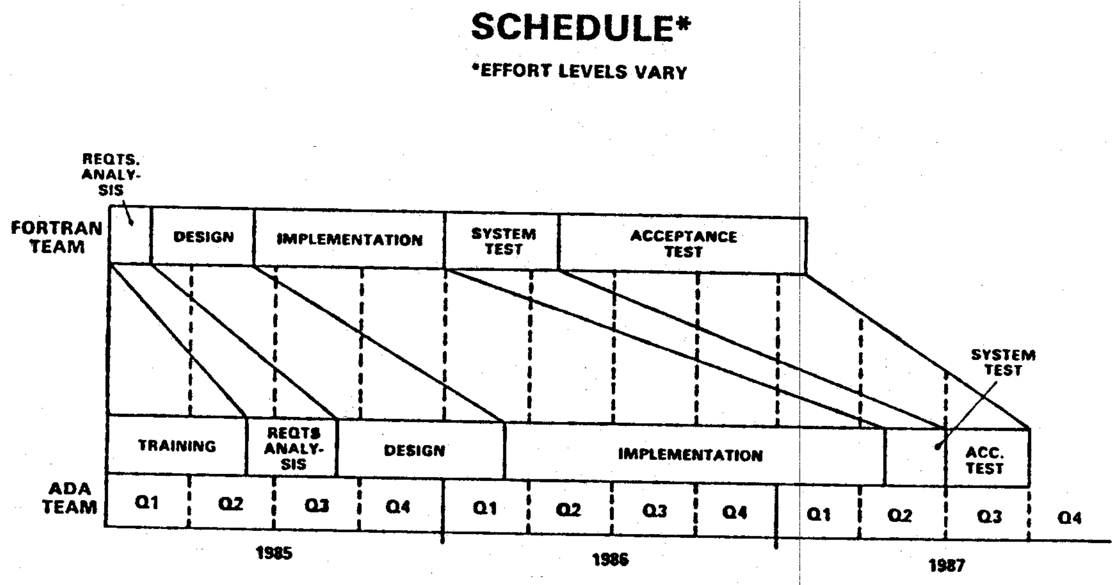

Figure 1. Schedule for the GRO experiment

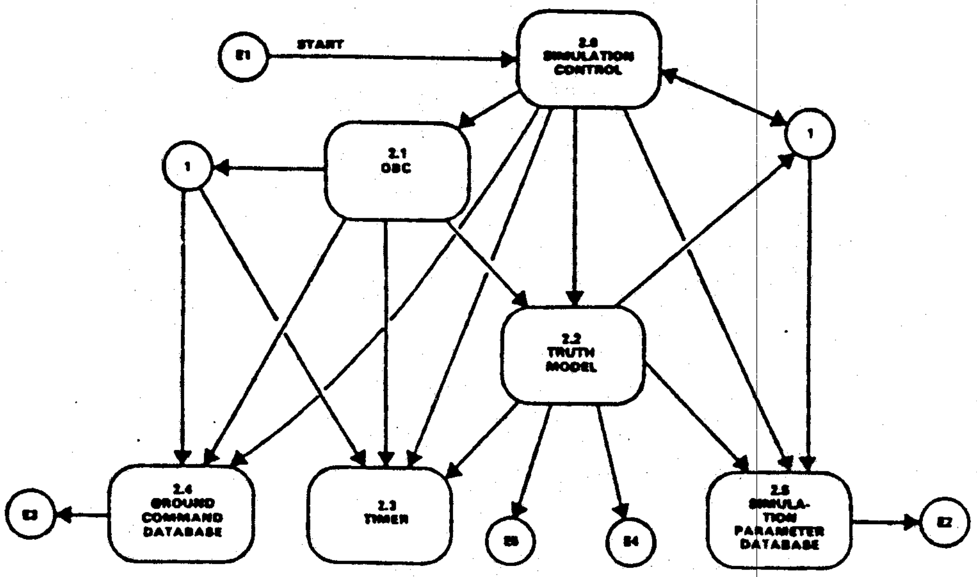

Figure 2. Example of an Object Diagram.
Seniority Hierarchy of Packages

Biographies

Carolyn E. Brophy is a graduate research assistant at the University of Maryland, College Park. Her research interests are in software engineering, and she is working with the NASA Goddard Software Engineering Laboratory. Ms. Brophy received a B.S. degree from the University of Pittsburgh in biology and pharmacy. She is a student member of ACM.

William W. Agresti is with Computer Sciences Corporation in Silver Spring, Maryland. He supports the NASA Goddard Software Engineering Laboratory, where he is currently project leader of the Ada development team. His research interests are in software process engineering, and he recently completed the tutorial text, *New Paradigms for Software Development*, for the IEEE Computer Society. He received the B.S. degree from Case Western Reserve University, the M.S. and Ph.D. from New York University.

Victor R. Basili's biography and picture are included with the paper "TAME: Tailoring an Ada Measurement Environment" by V. R. Basili and H. D. Rombach in these proceedings.