SOFTWARE DEVELOPMENT IN ADA

Victor R. Basili
Elizabeth E. Katz
University of Maryland

1. Introduction

Ada will soon become a part of systems developed for the US Department of Defense. NASA must determine whether it will become part of its environment and particularly whether it will become a part of the Space Station development. However, there are several issues about Ada which should be considered before this decision is made. What information is needed to make that decision? What are the training needs for Ada? How should the life cycle be modified to use Ada most effectively? What other issues should management consider before making a decision? These are but a few of the issues that should be considered.

One means of considering these issues is the examination of other developments in Ada. Unfortunately, few full-scale developments have been completed or made publicly available for observation. Therefore, it will probably be necessary to study an Ada development in a NASA environment.

Another means related to the first is the development of Ada metrics which can be used to characterize and evaluate Ada developments. These metrics need not be confined to full-scale developments and could be used to evaluate on-going projects as well.

The remainder of this paper describes an early development in Ada, some observations from that development, metrics which have been developed for use with Ada, and future directions for research into the use of Ada in software development in general and in the NASA Goddard environment in particular.

2. Overview of a Previous Project

In a previous project conducted by the University of Maryland and General Electric, we monitored a software development project written in Ada by integrating measurement into the software development process. Our goal was to identify areas of success and difficulty in learning and using Ada as a design and coding language. The underlying process and the evolving product were measured, and the resulting information characterized this project's successes and failures. Observations from the project might be used to make recommendations about training, methodology, and metrics to the Ada users community. This experience with data collection and metrics also will aid in the selection of a general set of measures and measurement procedures for any software development project.

This work is supported in part by the Office of Naval Research and the Ada Joint Program Office under grant N00014-82-0225.

The project studied involved the redesign and reimplementation of a portion of a satellite ground control system originally written in FORTRAN. Four programmers were chosen for their diverse backgrounds and were given a month of training in Ada and software development methodology. They designed the project using an Ada-like PDL although a processor for the PDL was not available at that time. The design evolved into Ada code which was processed by the NYU Ada/Ed interpreter. The design and coding phases of the project extended from April 1982 to December 1982. Some unit testing of the project was done during the summer of 1983 using the ROLM compiler; however, the entire system has not been tested.

We used a goal-directed data collection approach from the beginning. Goals and objectives for the study were defined. Specific question and hypotheses were associated with each goal. Data collection forms and procedures were developed to address these questions. The forms and procedures were integrated into the software development methodology. The final step of this approach involved analyzing the data in order to answer the questions and either accept or reject the hypotheses.

Most recently, the data have been analyzed. All the data from the forms were entered in a database as were the data gathered by a processor which parses the design and code, checking for correct syntax and taking various measurements. Our observations are summarized below and elaborated upon in [2] and [3].

3. Observations from that Project

Although the project studied ended part way through development, the results indicate what might happen in early stages of development in other projects. The data can be compared with the corresponding stages of other projects. The results from this project may prevent others from making costly management mistakes.

Learning Ada takes time. In this project it consumed 20% of the total effort. That time must be included in any estimate of effort for early projects using Ada. Training will probably have to be a continuing process as the team members learn the finer points of the language.

Ada is more than syntax and simple examples. The underlying software engineering concepts must be taught in conjunction with the support Ada provides for those concepts. Most programmers are not familiar with the methodologies developed in the seventies that Ada supports. Training in software engineering methodology and how to use it in the environment of a particular application is an absolute necessity for the proper use of Ada.

We do not know how Ada should be used. Ideally, our understanding of the software engineering concepts Ada supports would make the use of Ada natural. However, many people learn by example, and we do not have many good examples of how Ada should be used. We do not know how and when to use exceptions, tasks, and generics. We need to study various alternatives and show how they work with examples from various environments.

Design alternatives must be investigated. The design for this project was functional and more like than unlike the earlier FORTRAN design. This may be the

best design, but a group at General Electric developed an object-oriented design for the same project [4]. It is not clear which design, if either, is most appropriate. Just as a combination of top-down and bottom-up development is appropriate to many applications, a combination of functional and object-oriented design might well be most appropriate. Only after we know which type of design, or combination thereof, is best suited to the particular application can we teach people which design approach to use. Without such training, programmers will rely on their experience with other languages and will probably produce functional designs.

Proper tool support is mandatory. This project was done without a production-quality validated compiler. In addition to that very necessary tool, a language-oriented editor, which could have eliminated 60% of the observed errors, would have been desirable. This would have allowed the programmers to focus their attention on the logic errors that undoubtedly remain in the design and code. Data dictionaries, call structure and compilation dependency tools, cross references, and other means of obtaining multiple views of the system would have helped. A PDL processor with interface checks, definition and use relation lists, and various metrics would also be helpful.

Some methodology must be followed for a project to be successful. The methodology and tools to be used should be understood before the project begins. The effect of the lack of good tools is mentioned above. In addition, the PDL was loosely defined until after design began. Effective design reading might have caught many of the errors. Even if we wanted to test this project after a compiler became available, we would have needed to create a test plan after the requirements were completed. However, that aspect of the methodology was deemed unimportant. The language is only one aspect of the environment and methodology. It cannot save a project in which the rest of the methodology is ignored.

We believe that this project is atypical in that it was done before a compiler was available and was not finished. However, it is typical in that training consumed an enormous amount of effort and the programmers were not familiar with the underlying software engineering concepts of Ada and that it might look like the beginning of many projects. The learning curve in methodology is quite large. As we study more projects that use Ada, we will learn how to teach it, how to use it, and where we might make mistakes. Until then, we need to study Ada and its use further.

4. Metrics for Ada

In conjunction with the project described above, a number of metrics specific to Ada have been developed. Some of these have been used to evaluate the use of packages on that project and the other design presented in [4]. Two of the package metrics characterize the visibility of packages and the use of data hiding via packages. These and other metrics for packages are further described in [5].

Other aspects of Ada might also be measured. Although we have not studied these in detail at this time, metrics for tasking might characterize the shared code and evaluate the use of concurrency. Metrics for exception handling might measure

the locality of the exception handlers or the complexity of those handlers. However, we must determine how these aspects of Ada should be used before we try to assign qualitative values to these measures.

In addition, we are developing a taxonomy of evaluative, predictive, and characteristic metrics that might be used for Ada projects in particular but also non-Ada software developments. Metrics are placed in eight categories which fall roughly into two groups. The first group contains the process categories such as resource use, changes, and environment. The second group contains the product categories such as size, control, data, language, and operation. This is but one example of a categorization, and determining which categories are most pertinent to one's environment is a difficult task. However, we attempt to provide a set of metrics which can be used in conjunction with the data collection paradigm described above.

In addition to the categorization, the taxonomy also contains a formalization for describing metrics via formula generators. This is a notation for describing sets of metrics so that the myriad of combinations of metrics can be discussed without enumerating them. An earlier version of this work appeared in [1], but a better formalization is being developed.

5. Future Work

Ada is a new language and it is only starting to be used. We do not know how to teach people to use Ada correctly. We do not even know how Ada should be used. However, we plan some further research into Ada in order to answer some of the questions above.

We plan to continue our work with Ada-specific metrics. We would like to apply these metrics to various projects and compare the measures to our perceptions of the projects. Also in this area, we would like to develop more elaborate metric tools.

Also in the area of tools, we plan to categorize tools and techniques by the faults they will prevent, the faults they will detect, the faults they might detect, and the faults they will not detect. If we know the types of faults code developed in this environment usually contains, we might be able to apply the appropriate tools or techniques to best discover those faults.

There were many drawbacks to the project presented above. The training should have contained specific and more detailed examples. A clearly defined methodology, incorporating Ada, should have been used. Finally, the project should have been taken to completion. We plan to monitor other large projects in which these problems have been corrected. At least one of these will probably be done in the NASA environment to determine how Ada fits into that environment.

In addition, we would like to study various design alternatives. Comparisons of when to use an object-oriented versus a functional design would probably help in Ada training. However, we currently do not know when each type of design should be used. We need to determine some means of comparing designs and evaluating the various alternatives. Controlled experiments would be one vehicle, along with

the larger projects, for these studies of design.

There many interesting problems associated with Ada. We are addressing only some of those problems. We welcome any comments on our research and encourage others to investigate these and other aspects of Ada.

6. Acknowledgements

We wish to thank John Bailey, Shih Chang, John Gannon, Elizabeth Kruesi, Nora Monina Panlillo-Yap, Connie Loggia Ramsey, Sylvia Sheppard, and Marvin Zelkowitz for their contributions as the other monitors of the GE project.

7. References

[1] Victor R. Basili and Elizabeth E. Katz, "Metrics of Interest in an Ada Development," IEEE Workshop on Software Engineering Technology Transfer, Miami, FL, April 1983, pp. 22-29.

[2] Victor R. Basili, Nora Monina Panlillo-Yap, Connie Loggia Ramsey, Shih Chang, and Elizabeth E. Katz, "A Quantitative Analysis of a Software Development in Ada," University of Maryland Computer Science Technical Report, UOM-1403, May 1984.

[3] Victor R. Basili, Elizabeth E. Katz, Nora Monina Panlillo-Yap, Connie Loggia Ramsey, and Shih Chang, "A Quantitative Characterization and Evaluation of a Software Development in Ada," submitted to *IEEE Computer.*

[4] A.G. Duncan, J.S. Hutchison, J.W. Bailey, T.M. Chapman, A. Fregly, E.E. Kruesi, D. Merrill, T. McDonald, and S.B. Sheppard, "Communications System Design Using Ada," Proc. 7th Intl. Conf. on Software Engineering, Orlando, FL, March 1984, pp. 398-407.

[5] John D. Gannon, Elizabeth E. Katz, and Victor R. Basili, "Metrics for Characterizing Ada Packages" under draft.