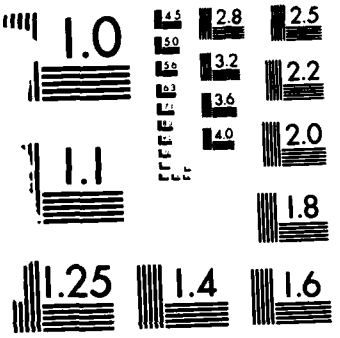
AD-A165 331 MONITORING AN ADA SOFTWARE DEVELOPMENT(U) MARYLAND UNIV 1/1
COLLEGE PARK V BASILI ET AL. DEC 82 N00014-82-K-0225

UNCLASSIFIED F/G 9/2 NL



Newsletter #2

(1)

NEWSLETTER - DECEMBER 1982
VOLUME I, NUMBER 2

MONITORING AN ADA SOFTWARE DEVELOPMENT

Victor Basili
John Gannon
Elizabeth Katz
Marvin Zelkowitz
University of Maryland

John Bailey
Elizabeth Kruesi
Sylvia Sheppard

General Electric Company

This research program is monitored by the Office of Naval Research (ONR) under contract #N00014-82-K-0225 to the University of Maryland with funding from ONR and the Ada Joint Program Office.

This newsletter is the second in a series describing a collaborative effort by research teams from General Electric and the University of Maryland. The purpose of this research effort is to monitor the use of Ada on a realistically large and complex software development project within industry. In particular, we are interested in investigating areas of success and areas of difficulty in designing and coding with Ada so that proper emphasis will be placed upon these areas in future Ada training courses. We area also interested in identifying metrics that are useful for evaluating and predicting the complexity, quality, or cost of Ada programs.

The first newsletter was published in the July/August 1982 issue of Ada LETTERS. A copy of that newsletter may be obtained by writing to Dr. Elizabeth Kruesi, General Electric Company, 1755 Jefferson Davis Highway, Arlington, Virginia 22202.

We are issuing periodic newsletters for two purposes: (1) to receive feedback from our colleagues regarding our approach, goals, and results and (2) to quickly disseminate our results so that others can benefit from our experience. Please address comments or questions to E. Kruesi.

In this newsletter, we briefly describe our approach to data collection followed by a description of the software development project that we are monitoring. We then address several central issues related to the use of Ada in the design phase of this project.

Data-Collection Goals

A major purpose of this research project is to integrate measurement into the software development process. As a result, we are collecting detailed information from this software development to characterize both the underlying process and the evolving product. This gives us the immediate benefit of having a complete record for gaining insight into the project's successes and failures. We also view this as a step toward selecting a general set of measures and measurement procedures that will be useful for any software development project.

There is an increasing awareness in the software engineering community of the need for systematic measurement. However, there has been little agreement of what characteristics to measure and of how to make use of these measures. Our approach to measurement for this project was to define a number of goals or objectives for the data-collection effort. We then defined a number of specific questions or hypotheses related to each goal. Data collection forms and procedures were developed to address these questions. The final step involved integrating these forms and procedures into the software development methodology.

Appendix A of the first newsletter delineated eight goals that were concerned with characterizing the software development process and the programmers' use of Ada as a design and coding language. The appendix of the current newsletter lists an additional set of goals and questions. These focus upon measurements to predict programmer effort and program quality and upon the ease and efficiency with which those measurements can be collected.

The Software Development Project

The software project involves re-

implementation in Ada of a portion of a working ground support system for communication satellites. The original system consists of approximately 100,000 lines of FORTRAN and assembly code which was developed by General Electric's Space Systems Division in Valley Forge, Pennsylvania. With the help of the original designers, we selected a subset of the original requirements for redesign and implementation in Ada. This subset was chosen to meet two criteria. First, we wanted a self-contained unit of several functions that could be developed and tested apart from the larger system. A second criterion was that the development effort be of a size and complexity to be completed by three programmers and a program librarian within an eleven-month period. It was known that some of this calendar time would be required for training in Ada and in the project methodology. We also anticipated a lower-than-normal level of productivity from the programmers due to the extremely thorough data collection procedures instituted by the research team and to the lack of a production quality compiler and set of support tools. The software is described in more detail in the first newsletter.

Project Phases

The project began in February 1982 with a month of formal training in Ada. Following this, the lead programmer and back-up programmer wrote the project requirements using the requirements from the larger project as a starting point. The resulting requirements document describes a complete subsystem which is executable apart from the larger system.

None of the team members worked on the original system and they have not had access to the original design or code. It was assumed that such accesses might bias their Ada design to be like the FORTRAN-based design for the original system. The designers of the original system were, however, available for consultation during this period. A formal requirements review was held in early May.

Following this review, a high-level design was produced in an Ada-like program design language (PDL). This design was then refined into a more detailed design for each module of the system. Coding began in August. The code is currently being compiled on the NYU Ada/Ed interpreter. In addition, the team is attempting to unit test and integrate as much of the software as possible. The software development is scheduled for completion in December 1982. Analysis of the data collection by the General Electric - University of Maryland teams will continue through July 1983.

The remainder of this newsletter describes results from the Ada training and issues related to the methodology used on this project.

Ada Training

As noted above, the first month of the project was devoted to training. The training began with a series of twenty-one videotapes which were produced by Honeywell. These tapes feature a series of lectures by language designers Ichbiah, Firth, and Barnes and encompass a total of fifteen hours. The team then attended six day-long lectures by George W. Cherry of Language Automation Associates. The lectures extended over a four-week period. Between classes the team practiced compiling and executing sample programs on the NYU Ada/Ed interpreter. The DoD Draft Reference Manual for the Ada Programming Language and a number of relevant articles were also provided.

The formal training was completed in March, although the backup programmer and third programmer spent parts of the next three months writing small Ada programs to try out various features. Thus, the training phase continued beyond the end of formal training for these two team members. During this period, each programmer was asked to complete an attitude survey. The programmers were also interviewed to obtain additional information about their reactions to the training and to Ada in general.

A major factor that appeared to influence their initial reactions to Ada was the extent of their experience with other programming languages. The programming team was originally chosen to provide a diversity of backgrounds. The lead programmer had ten years of experience in the application area and prior supervisory experience. He is, however, typical of long-time industry programmers in that he had not had experience with a wide variety of langauges; he had used only FORTRAN and assembly languages. The other two programmers on the project had less experience in the application but a greater diversity of language experience. At the conclusion of the Ada training course, they expressed a

much higher degree of confidence than the lead programmer in their ability to use Ada's features effectively. The greater confidence of the two programmers was probably partly due to their past experience and partly due to the extra time they spent trying various features of Ada.

Software Design Methodology

As noted earlier, the formal requirements review was followed by a two-step design phase. These two steps emerged as a result of several issues that arose in attempting to use Ada as a program design language (PDL). Before discussing these issues, it is worth noting that there were several software development techniques that were integrated in a straightforward manner into the design methodology. These included the use of a project librarian to maintain strict configuration control, a strong reliance on design walkthroughs for error detection, and the use of a lead programmer to supervise the evolution of all components in the system.

While these techniques worked well, the team's attempts to use Ada as a PDL were less straight-forward and required some degree of experimentation. We feel there is value in describing the team's experience in using Ada as a PDL. We then turn to what is probably a more critical design issue - the question of the basis for decomposing a system into components.

Ada as a PDL

PDL is rapidly replacing flowcharts as a design medium and has been used to assist in many stages of the software life cycle. Besides serving as an aid for simplifying the coding process, it is used as a means of tracing functionality from the requirements, as a medium of communication among designers, as a deliverable to verify the feasibility of a design, as an aid to creating test plans, and as documentation to aid in maintaining the system.

There appears to be general agreement in the Ada community that Ada provides many of the features required of a design language and can therfore serve as the basis for a PDL. We have learned, however, that there is a big step between this notion and actually defining the structure of a useful PDL. Over the past year, there has been much debate in the Ada community over several attributes of an Ada-based PDL. The primary issues are whether the PDL should be acceptable to an Ada compiler or to some other processing tool and whether it should be composed of a subset or superset of Ada.

Our team began the design phase with the intent of using compilable Ada as the design language. As part of the initial training period, the use of a PDL was explained along with the concept of stepwise refinement.

We recognized immediately that when the primary emphasis was to design using compilable Ada, the resulting design evolved as increasingly detailed threads of functionality rather than as complete descriptions of the system at each level. That is, each team member tended to follow one function through the various design levels, filling in greater detail at each lower level for that function. This contradicted the idea that a proper stepwise refinement should result in a complete description of the system at each level before being further refined to the next lower level of detail. In an attempt to eliminate this tendency, two design phases were defined. The first phase involved a brief description of each known component in the system. The purpose of each component was described in only enough detail to provide traceability of functionality from the requirements into the design. Any known inputs and outputs were also noted. The purpose of this design was to focus the team's attention on each complete level of design detail and to emphasize traceability to the requirements.

The goal of the second phase was to write a more precise design. In addition to providing specific algorithms and complete interface specifications, all data types were defined and all objects declared. A design walkthrough was held at the end of each phase for each component. These two design phases were followed by the coding phase for each component.

We do not necessarily feel that the approach used, and outlined here, was the best way to have developed the software. Although it did result in an apparently workable design, we experienced several difficulties during development and are currently in the process of evaluating the causes and potential remedies for these. More detail will be published about this in the near future.

Design Decomposition Issues

At this point, all components have been designed and nearly all have been coded. A review was conducted early in October to evaluate the development and, in particular, to address Ada's impact on the design. In addition to the General Electric - University of Maryland research teams, the review was attended by the chief engineer of the original system and by two members of General Electric's Corporate Research and Development staff who have extensive Ada experience. The outcome of this evaluation will be covered in more detail in a later newsletter. A few of the major points are worth mentioning here.

The design was judged to be a good, workable design. The implementation used a wide variety of the features of the language, including such advanced features as task types and discriminant records. The design, however, was characterized as a functional one which is more like than unlike the original (FORTRAN-based) system. This result is not surprising considering that the requirements were defined in such a way that one might easily expect a functional design to emerge. A question was raised concerning whether an alternative design approach, variously referrred to as object-oriented, message-based design, or designing with abstract data types had been considered. Although examples of data abstraction and encapsulation were presented in the Ada course, the focus was on the language features that support those ideas rather than on the ideas themselves. It was agreed that this approach, being new to all members of the team, would have required additional education and guidance beyond the scope of the classroom examples. The relevant training for alternative design approaches has to come early in a development since it impacts the very first design decisions and perhaps even the requirements analysis phase.

Preliminary Recommendations for Training Courses

As a result of observations made throughout this project and on the basis of interviews with the team members, we have several specific suggestions for improving Ada training. We suggest, first of all, that different courses be tailored to those with different backgrounds. The breadth of a programmer's previous language experience will undoubtedly be an important factor. Programmers who have used a number of different languages, particularly Pascal and Algol, are familiar with a greater number of relevant concepts than those who have used only assembly languages or FORTRAN. The latter group includes a great many industrial programmers.

The programming team felt unanimously that any Ada training course should begin with a discussion of the software engineering concepts that are supported by various features in the Ada language. Such an introduction, for example, would include a discussion of the importance of a concern for software modifiability, re-usable software components, and runtime reliability. The specific software engineering techniques which support these concepts could then be introduced. For example, Parnas' notion of information hiding could be discussed as a design technique which is intended to result in modifiable software. After discussion of these concepts and techniques, the training could then proceed to a concentration on the syntactic structures of the language. The rationale for this type of approach to training is that one does not necessarily gain an understanding of how to use a language feature simply from viewing examples of the feature in sample program segments. Each new feature needs to be taught in such a way that it can incorporated into a framework of related ideas already present in the programmer's knowledge base.

Beyond these specific suggestions, we recognize the need for a coherent software development methodology using Ada. While the language was designed to support a number of software engineering concepts, it is not immediately obvious how to integrate these concepts into a coherent methodology. As noted earlier, our software development team used a functional decomposition for the design of the system. This may not be the best basis for decomposing the system if one is attempting to maximize its later maintainability. On the other hand, other aspects of the team's design strategy were successful, such as assuring traceability of functionality from the requirements.

The current Methodman effort, supported by the Ada Joint Program office, is tasked with defining a recommended set of methodologies to be employed for Ada development and maintenance efforts. Our experience suggests that the Methodman effort is badly needed.

It is important to note that these observations have resulted from a preliminary analysis and are representative of our current reactions to using the Ada language on this project. We are gathering a great deal of objective data about the use of Ada during the design, code, and testing phases of this project. As part of the data collection, for example, we are tracking all errors detected. We are also obtaining the subjective judgments of people well versed in Ada concerning the extent to which the language has been used properly or improperly on our project. Finally, we are obtaining a number of measures (both static and dynamic) to characterize the resulting software. A full picture concerning the programming team's difficulties and successes with Ada must await the collection and analysis of the complete set of data.

APPENDIX

Area C*: Goals Relating to Metrics for the APSE

General Goals

Provide a database for future Ada projects to be used to predict important properties of these projects (e.g., development effort).

Select a set of measures that can be collected during development and are useful for predicting the operational characteristics of the system such as its reliability and maintainability.

Select a set of measures to provide software developers and managers with useful feedback during software development.

Note: These goals will be addressed by evaluating the usefulness of a number of different types of measures. A basic distinction can be made between measures of the software development process and those of the product. Appendix A of the first newsletter was largely concerned with measures of the process. The goals listed in the current newsletter are primarily concerned with product measures. Product measures can be broken down into static and dynamic (run-time) measures of the product. The following goals support the more general ones listed above.

---

* Areas A and B ("Generic Goals for any Software Development Project" and "Goals Relating to Ada as a Design and Implementation Language") were listed in Appendix A of the first newsletter.

Goal C1: Select a set of static (size, control and data) metrics for the APSE

1) Are there differences in the implications of various counting measures? Are some measures more useful than others?
2) Do certain program measures provide enough information to make other measures superfluous?
3) Which static metrics can be applied throughout the design and code phases. Which cannot?
4) Which static metrics help predict run-time behavior (e.g., reliability, etc.)?
5) Which static metrics can be measured most easily?

Goal C1.1: Develop a set of size metrics for the APSE

1) What size metrics best predict effort?
2) What serves as a useful size metric (e.g., lines of code, modules) in Ada?
3) What constitutes a statement in Ada?
4) How should an executable statement be defined in Ada?
5) What features of Ada should be grouped when counting the number of times certain features are used?
6) How useful is Halstead's software science approach with Ada?

Goal C1.2: Develop a set of control metrics for the APSE

1) How can tasking and exceptions be integrated into the control metrics?
2) How useful is McCabe's cyclomatic complexity measure? How does the cyclomatic complexity compare with the essential complexity?
3) How useful are measures of nesting complexity and depth?

Goal C1.3: Develop a set of data metrics for the APSE

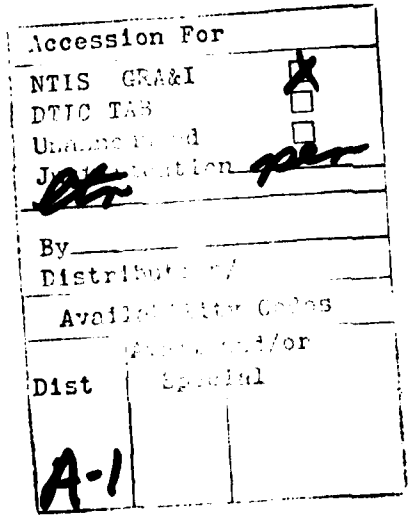
1) How can the complexity of data structures be measured?
2) What influences the number of programmer defined types?
3) How does the use of Ada influence the number of inputs to and outputs from a module?

4) How does Ada influence the use of global data?
5) How does the use of modules affect the treatment of data within a program?
6) How should the span of a variable be measured? Is there a use for the span information?
7) What do the data bindings suggest about the structure of the system?
8) Does the density of the data flow across modules provide useful feedback about the structure of the system? i.e., are information flow metrics (Henry & Kafura) useful?

Goal C2: Select a set of dynamic (test coverage and execution) metrics for the APSE

Goal C2.1: Develop a set of test coverage metrics for the APSE

1) Do any of the following measures of test coverage lead to a useful strategy for testing: number of statements executed? number of decisions executed, or number of independent paths executed?
2) Can these measures be extended to provide test coverage for concurrent processing or will new measures need to be developed? Are there measures to detect starvation, potential deadlocks, etc.?
3) Are there other features of Ada (e.g., exception handling) that require new measures for test coverage? What are those measures?



Goal C2.2: Develop a set of execution metrics for the APSE

1) What are useful execution metrics?
2) What additional information do execution statistics provide beyond what can be gained from a static view of the system?
3) Are there measures of execution complexity?
4) Are certain Ada features or combinations of features expanded into very fast or very slow code?

Goal C3: Develop a subjective evaluation system for evaluating some program and design features that are not easily or practically measured in other ways

1) Can a diverse set of experts (Ada, applications, and methodology experts) accurately evaluate the subjective aspects of the project?
2) How well do the results of these evaluations correlate with results from objective measures?
3) How well do these evaluations correlate with the opinions of the development team?
4) Can we conclude anything from the subjective results?

DTIC

FILMED

4-86

END