FEASIBILITY OF A GRAPHICAL DESIGN FOR AN ADA SOFTWARE DEVELOPMENT

Pamela S. Woodard
Naval Training Systems Center
12350 Research Parkway
Orlando, FL 32826-3224

William F. Parrish, Jr.
Naval Training Systems Center
12350 Research Parkway
Orlando, FL 32826-3224

ABSTRACT

The use of Ada for training system software requires that a greater emphasis be placed on the design phase of a software development. The representation of an Ada design must provide a means to describe a variety of components such as packages, procedures, tasks and generics, and a means to specify the interconnection information between all components. A graphical design methodology is a promising technique which may offer an effective means to document an Ada design so that it can be quickly and correctly understood by programmers, project engineers and program managers. This paper describes the benefits of a graphical design and presents four methodologies that are beginning to be used to design training systems software. Structured Analysis and Design Technique (SADT), Yourdon-DeMarco Structured Analysis and Design (YDSAD), Process Abstraction Method for Embedded Large Applications (PAMELA), and Object Oriented Design (OOD) are discussed. Advantages and disadvantages of the various techniques with respect to total life cycle support are presented. Factors such as ease of learning, ease of use, understandability, and automated support are given prime consideration. Evaluation results are presented in tabular form.

INTRODUCTION

The Naval Training Systems Center has recently begun to specify that Ada will be used for all training system software development. Along with the introduction of Ada, emphasis is being placed on new software engineering tools and techniques. Of considerable interest are those tools and techniques which make extensive use of graphics and can be used for the total life cycle. Emphasis is also being placed on techniques which can be used to document reusable software.

Low cost graphics hardware and software along with recently developed design techniques afford new opportunities for designing and supporting software. In the past software designs have been documented primarily using text. This paper describes the superiority of graphics over text for conveying design information.

The purpose of this paper is to increase training systems software developers awareness of these tools and techniques. Greater use of these methodologies will stimulate the development of more and better automated tools. Evaluations presented should aid software developers in selecting a methodology appropriate for their particular development.

Tutorial presentations of the various methodologies are not intended. A comprehensive description of the various methodologies is beyond the scope of this paper. Textbooks and/or courses are available for all of the methodologies described. A brief description of each method is provided in order to make the comparison of the techniques understandable.

USE OF GRAPHICS IN SOFTWARE DESIGN

People relate easily to pictures and graphics. We use pictures to present and explain all kinds of information. Charts and graphs have become familiar and help us to assimilate and understand the data in business presentations, newspapers, and textbooks. In an article which surveyed the current use of graphics in programming, George Raeder states that it is "commonly acknowledged that the human mind is strongly visually oriented and that people acquire information at a significantly higher rate by discovering graphical relationships in complex pictures than by reading text" (1).

Advantages of Graphical Representations

Raeder describes several reasons why pictures have an advantage over text in providing information clearly and quickly.

Text is a sequential mode of expression. We must read or scan through preliminary information in order to find a particular description or explanation. Pictures offer us random access to data. Our eyes can rapidly move to any area of a drawing and locate required details. Important features can be highlighted to focus our attention quickly.

Text is a one dimensional medium consisting of words. Pictures provide three dimensions for the description of information. Physical properties such as shape, color, texture, and size can be used to enrich the presentation. Because pictures can provide data with more variety, they can present the same data more concisely and compactly than text.

Pictures generally transfer data faster than text. The data can be accessed and understood quicker and with less effort than a textual description.

Pictures are often used to present abstract ideas in ways that make them simpler for people to comprehend. When trying to understand an abstract concept, many people come up with a mental image which enables them to understand the abstraction. A picture can present an already prepared image to a reader and speed up understanding of new or complex ideas.

Visual Programming

The area of visual programming has received an increasing amount of attention from researchers in recent years. Visual programming is the use of graphics to display and/or interact with software and its associated documentation. Grafton and Ichikawa identify three related areas which comprise visual programming: software views, graphical programming and animation (2).

Graphics techniques can be used in many ways to represent different views of software which can clarify and present a software system from different viewpoints for the system designer, programmer, manager, user, and maintainer. Graphics allow the system to be portrayed using symbols, icons, arrows, and other connectors in a structured format.

Graphical programming offers a new approach to the creation of programs. The notion is that a programmer would create a software representation by the manipulation of symbols and icons. The symbols would then be translated into Ada.

In the area of animation, researchers are using graphics to offer the ability to observe the control and data flow of an algorithm as it executes. This gives a programmer the capability to quickly and easily understand the execution of an algorithm and clarify its performance characteristics.

This paper will concentrate on the use of graphical techniques to portray software views, particularly for Ada software systems.

Graphical Software Views

Graphics have been used for years in the computer programming field. One of the earliest and best known graphical techniques for software design is the flowchart. Flowcharts were originally developed for assembly language programming and offer a good method for the depiction of fairly simple control structures.

Current higher level, structured languages have developed to the point where flowcharts are no longer sufficient to provide a good view of a software system. Complicated data structures and data types, more complex control structures and concepts such as packages, generic procedures, and concurrent tasks cannot be described using flowcharts. More sophisticated graphics techniques have been developed for higher level languages and currently the adoption of the Ada language is pushing the development of automated graphical techniques as a part of CASE (Computer Aided Software Engineering) tools.

STRUCTURED ANALYSIS AND DESIGN

A variety of structured software design methodologies were developed in the 1970's as software projects became increasingly complex and new tools were needed to deal with this complexity. Methodologies based on top-down, functional decomposition are still the most widely used techniques today. Two of the best known of these methodologies are the Structured Analysis and Design Technique (SADT) from SofTech, Inc. (3,4) and the Yourdon-DeMarco Structured Design methodology.

Structured Analysis and Design Technique (SADT)

SADT is based on the premise that any amount of complexity can be understood if it is presented as smaller pieces which together make the whole. The basic maxim of SADT is that everything must be broken into 6 or fewer pieces. The technique requires that the user begin by looking at the topmost level of a system. This level is then broken down into two to six pieces of subject matter. The SADT maxim is then applied to each of these pieces of the system, and the process continues recursively until all lowest level pieces can be completely understood with no further decomposition.

The output of SADT is a hierarchically organized structure of diagrams which is called a SADT model. Each diagram portrays a limited area of the total system.

SADT Graphic Representation

The SADT graphical design notation uses two basic components - boxes and arrows. Each of the pieces of subject matter from the SADT decomposition is expressed within a box. The four sides of the box are always used to represent input, control, output, and mechanism. The pieces of information within each box are related and connected by arrows. The arrows connect the output of one box to the input or control of another box.

The same graphic notation is used to describe both data and activities. In an activity model, the control arrows represent the dominant constraints on the system and determine when activities take place. In a data model, the input arrows represent the dominant constraints for a data box. An abstract SADT diagram for an activity model is shown in Figure 1.

The SADT diagrams use a coding scheme to label arrows called ICOM codes. An ICOM code begins with the letter I, C, O, or M (standing for input, control, output, or mechanism) followed by an integer numbered from left to right or top to bottom. The external, boundary arrows of a diagram are labeled with ICOM codes which will match the corresponding arrows on the corresponding parent box. Arrows can branch or join to display distribution. Subdivision is represented by arrows for bundle or spread. Two way interfaces are illustrated using dots above or below the arrowheads.

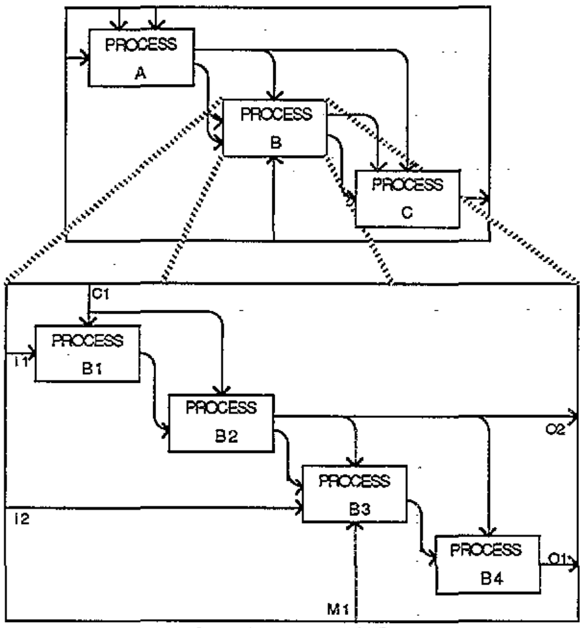

Figure 1. Abstract SADT Diagram

The graphic representation of SADT can be used with any language. There are no graphic structures which support particular features of the Ada language. SADT is used primarily as a requirements definition methodology and is often interfaced to other software design methodologies to develop the actual detailed system design. At present SADT is less effectively used for actual detailed design of software systems. This may change as automated support tools are developed. Both SofTech and Mitre are developing graphic tools linking SADT with Ada packages for life cycle software development.

Yourdon-DeMarco Structured Analysis and Design

The basic concepts of structured design were outlined originally in an article by Stevens, Myers and Constantine in 1974 for the IBM Systems Journal. Further developments and refinements have been made by a number of contributors including Yourdon, DeMarco and Jackson. The following discussion of the Yourdon-DeMarco structured system design methodology is based on material found in texts by DeMarco (5) and Page-Jones (6).

The purpose of structured analysis and design is to develop a system through partitioning of the problem into modules which are then organized into hierarchies. The methodology consists of structured analysis applied during the analysis phase of a software project and structured design techniques used during the detailed design of a software system.

Yourdon-DeMarco Graphic Representation

The Yourdon-DeMarco structured design methodology results in two graphic representations of a system - data flow diagrams and structure charts.

During the structured analysis phase the system is partitioned into data and processes. The data flow diagram, or bubble chart, is an output of this phase and is the graphic representation of this partitioning. The data flow diagram portrays the system as a network and is composed of four basic graphic elements called the data flow, the process or transform, the data store and the terminator or source/sink. An abstract data flow diagram is shown in Figure 2.

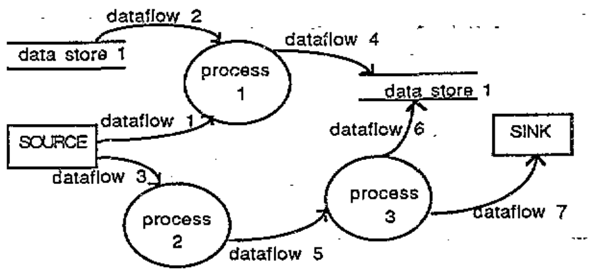

Figure 2. Abstract Data Flow Diagram

The data flow element illustrates the flow of data through the system using an arrow. The arrow is labeled with the name of the piece of data being described and the direction of the arrow indicates the direction of data flow. The process icon indicates a transformation of incoming data flow(s) into outgoing data flow(s). Processes are represented as bubbles or circles labeled with the process name. The data store is a repository for information and can be thought of as a file, database, table or any other mechanism which can be used to store data. A data store is represented as two parallel lines labeled with the name of the data storage area between the lines. Terminators, or sources and sinks, are used to mark the boundaries of a particular system model, and are represented by rectangular boxes.

The structure chart is the graphic representation of the design and is used to illustrate the partitioning into modules, and the hierarchy, organization, and communication of the modules. An abstract structure chart is shown in Figure 3.

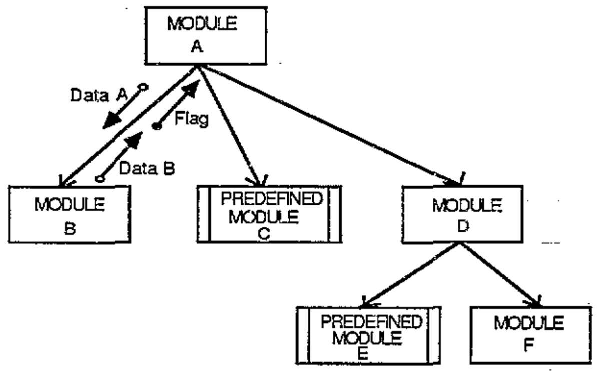

Figure 3. Abstract Structure Chart

A module is represented as a rectangular box and labeled with the module name. A predefined module is a module which already exists in a software library and is represented by adding a second vertical line on each side of the module box. Connections and communications between modules are shown using arrows. In the figure, Module A calls Modules B, C, and D. Module A sends DATA A to Module B and Module B returns DATA B and a flag to the calling module.

The graphic representations used in the Yourdon-DeMarco methodology are not specific to a particular computer language. A growing number of CASE (Computer Aided Software Engineering) tools are becoming available which automate the generation of data flow diagrams and structure charts.

PROCESS ABSTRACTION METHOD FOR EMBEDDED LARGE APPLICATIONS

The methodologies previously discussed are applicable primarily to the requirement analysis and design phase of a software development. George Cherry's PAMELA methodology supports more of the software life cycle (7). PAMELA encompasses features of other well known methodologies. Features of the SADT and OOD design are included. A syntax directed editor and automatic code generator called AdaGRAPH is available from The Analytic Sciences Corporation.

PAMELA allows a designer to graphically design a system by generating a set of hierarchical diagrams. The diagrams consist of data flow and control arrows, task idiom symbols, shared storage symbols, and called or calling process indicators as well as several other indicators. An example PAMELA diagram is shown in Figure 4. Figure 4 shows a process X1 calling a process X2. In PAMELA terminology the !A means here_is_an_A and the ?B means give_me_a_B.

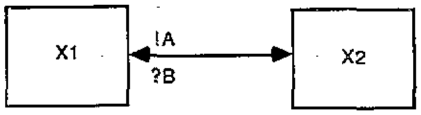

Figure 4. Example PAMELA Diagram

Thirteen idioms are used to represent Ada tasks or procedures. These idioms consist of such things as cyclic activity controllers, monitors, and state machines. Using these idioms a software design can be expressed from specifications down to a primitive level.

PAMELA diagrams offer a very powerful aid in validating a software design. Engineers and programmers can "walk through" the diagrams to determine if the design accurately models the physical entity being implemented or simulated. Once the design has been validated the AdaGRAPH tool can be used to generate a substantial amount of Ada code. Algorithmic details must be added by a programmer to complete the code.

PAMELA diagrams are also very useful for formal design reviews. Since the diagrams are hierarchical, only the level of detail required needs to be viewed.

OBJECT ORIENTED DESIGN

A software development methodology called Object Oriented Design (OOD) has been developed in the 1980's and implemented in Smalltalk and Ada. This approach views a software system as a set of objects and the operations performed on these objects, rather than a set of actions or processes.

The use of an object oriented approach requires that the decomposition of a system be based on objects. Proponents of this method suggest that the system to be modeled or simulated be described by one paragraph. Once written the paragraph is analyzed to determine the objects (nouns) and operations (verbs) of interest. Languages such as Ada are better suited than traditional languages for this design approach, since Ada provides packages and tasks as structured building blocks in addition to the traditional building block of subprograms.

Booch has developed a graphical notation for an object oriented approach to software development (8) and has written a book describing its application to software development in Ada (9). The graphical notation uses rectangles and parallelograms to represent subprograms, packages and tasks. Arrows are used to illustrate communication between units. Objects are represented as amorphous shapes. An example of this notation is given in Figure 5.

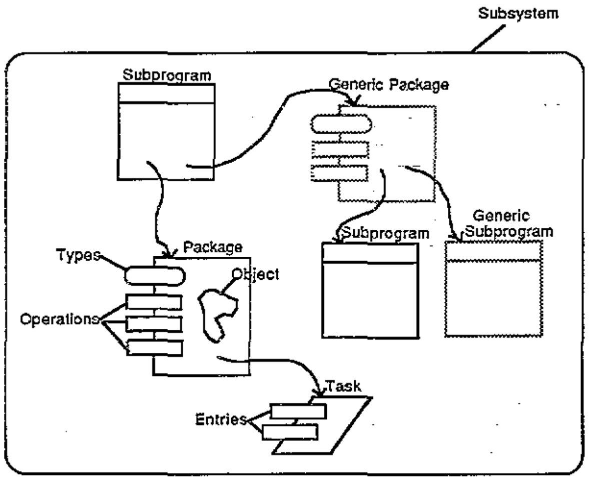

Figure 5. Abstract Object Oriented Design

Booch's graphical notation for OOD maps well to the Ada language. Automated support for OOD development is offered by Rational as a hardware/software development system. This is not an economical solution for many users.

SUPPORT FOR REUSABLE SOFTWARE

All of the techniques discussed should help to document reusable software. Reusable software does not happen automatically. Software must be designed to be reusable. Many factors go into making software reusable. However, the most important is probably understandability. Software will be reused only if it can be easily understood.

SUMMARY

As with most things, each method has its strengths and weaknesses. These are shown in Table 1. The first three columns represent the ease with which one can learn, use, and

METHOD	Easy to Learn	Easy to Use	Easy to Understand	Life Cycle Support	Automated Support
SADT	Moderate	Hard	Moderate	Very Limited	Very Limited
YDSAD	Moderate	Moderate	Moderate	Good	Good
PAMELA	Easy	Easy	Easy	Good	Good
OOD	Moderate	Moderate	Easy	Limited	Limited

Table 1. Comparison of Graphical Methodologies

understand the methodology. Discriminators used in these columns are easy, moderate, and hard. Easy means that an engineer with some software experience should be able to readily learn and use the methodology. Moderate means that some computer science knowledge is required. Hard means that substantial software engineering knowledge and experience is necessary. Column 4, life cycle support, represents the ability of the method to be used for the entire software life cycle. Column 5, automated support, depicts the availability of automated tools that facilitate the use of the methodology. Discriminators used in these columns are very limited, limited, and good. Very limited in the life cycle support column means that the method applies to only one phase of the software life cycle. Limited means that the methodology applies to two or more phases. Good denotes that the methodology applies to all phases of the software life cycle.

Methodologies noted in the automated support column as very limited are only supported by few automated tools. Limited means that some adequate automated support tools are available. A notation of good indicates effective and efficient automated tools are available to support the methodology.

A very promising tool is being developed by David Workman at the University of Central Florida (10). The Workman tool is called Graphical Interactive Programming (GRIP). This tool appears to support the entire software life cycle. The tool is used by developing several coordinated views of a software system. These views are a structure view, a control view, a data view, a computation view and a runtime view. The most significant feature of this tool is that changes made in the control view cause changes to be made automatically in the computation (code) view and vice versa.

Currently the tool generates C Code. Consideration is being given to enhancing the tool to generate Ada code.

CONCLUSION

Graphical design techniques and support tools described in this paper as well as others are currently evolving. Application of any of these techniques should increase software productivity.

Training system software developers should be aware of these tools and techniques and consider their use when developing trainer software. Some methodologies are best suited to a limited class of applications while others may be applied more broadly. For example, PAMELA is designed to support real time applications and probably would not be useful for a data base application. On the other hand, OOD is applicable to a wide range of applications. Careful consideration should be given to several methodologies prior to selecting one for a software project. For some applications it may be appropriate to use a combination of methodologies. For example, some developers are using Data Flow Diagrams during the analysis phase and OOD during the detailed design phase.

To standardize on any one technique at this time would be premature. Tools that document designs, generate code and aid in software maintenance should be given prime consideration. Ultimately the technique that provides the most comprehensive life cycle support will win in the market place.

REFERENCES

1. Raeder, Georg. "A Survey of Current Graphical Programming Techniques." IEEE Computer 18 (August 1985): 11-25.

2. Grafton, Robert B. and Ichikawa, Tadao. "Visual Programming." IEEE Computer 18 (August 1985): 6-9.

3. Ross, Douglas T. "Structured Analysis (SA): A Language for Communicating Ideas." IEEE Transactions on Software Engineering SE-3 (January 1977): 16-34.

4. Ross, Douglas T. "Applications and Extensions of SADT." IEEE Computer 18 (April 1985): 25-34.

5. DeMarco, Tom. Concise Notes on Software Engineering. New York: Yourdon Press, 1979.

6. Page-Jones, Meilir. The Practical Guide to Structured Systems Design. New York: Yourdon Press, 1980.

7. Cherry, George. "Real-Time Applications Design Using Ada" Seminar, January 1987.

8. Booch, Grady. "Object-Oriented Development." IEEE Transactions on Software Engineering SE-12 (February 1986): 211-221.

9. Booch, Grady. Software Engineering with Ada. Menlo Park, CA: The Benjamin/Cummings Publishing Company, Inc., 1987.

10. Workman, David A. "A C-Based Visual Programming Environment Supporting Multiple Views." Orlando, FL: University of Central Florida, Computer Science Department, 1988. Photocopied.

ABOUT THE AUTHORS

Pamela S. Woodard is a Computer Scientist in the Surface/Submarine Warfare Software Branch at the Naval Training Systems Center where she is the Software Engineer for several trainers that are being procured. She has over 14 years of software development experience, including 7 years within the Research Department of the Naval Training Systems Center. The software for one of her trainers, the LSD-41 Propulsion Control System Trainer, is being written in Ada. Mrs. Woodard holds a BS degree in Mathematics from George Mason University, and an MS degree in Computer Engineering from the University of Central Florida.

William F. Parrish, Jr. is a Supervisory Electronics Engineer in the Surface/Submarine Warfare Software Branch at the Naval Training Systems Center and is currently involved in procuring trainers with Ada software. He has over 21 years of software development experience. Prior to his employment with the Navy, he was a Senior Engineer with Sperry Rand Corporation. Mr. Parrish holds a BSE degree in Electrical Engineering and a MSE degree in Computer Engineering from the University of Alabama in Huntsville.