GRASP/Ada

Graphical Representations of Algorithms, Structures, and Processes for Ada

Update of GRASP/Ada
Reverse Engineering Tools For Ada

Final Report

Delivery Order No. 21
Basic NASA Contract No. NAS8-39131

Department of Computer Science and Engineering
Auburn University, AL 36849-5347

Contact: James H. Cross II, Ph.D.
Principal Investigator
(205) 844-4330
cross@eng.auburn.edu

December 14, 1993

Update of GRASP/Ada

Graphical Representations of Algorithms, Structures, and Processes for **Ada**

Reverse Engineering Tools For Ada
Final Report

Delivery Order No. 21
Basic NASA Contract No. NAS8-39131

James H. Cross II, Ph.D.
Principal Investigator

December 14, 1993

Abstract

The GRASP/Ada project (Graphical Representations of Algorithms, Structures, and Processes for Ada) has successfully created and prototyped a new algorithmic level graphical representation for Ada software, the Control Structure Diagram (CSD). The primary impetus for creation of the CSD was to improve the comprehension efficiency of Ada software and, as a result, improve reliability and reduce costs. The emphasis has been on the automatic generation of the CSD from Ada PDL or source code to support reverse engineering and maintenance. The CSD has the potential to replace traditional prettyprinted Ada source code. In Phase 1 of the GRASP/Ada project, the CSD graphical constructs were created and applied manually to several small Ada programs. A prototype CSD generator (Version 1) was designed and implemented using FLEX and BISON running under VMS on a VAX 11-780. In Phase 2, the prototype was improved and ported to the Sun 4 platform under UNIX. A user interface was designed and partially implemented using the HP widget toolkit and the X Windows System. In Phase 3, the user interface was extensively reworked using the Athena widget toolkit and X Windows. The prototype was applied successfully to numerous Ada programs ranging in size from several hundred to several thousand lines of source code. Following Phase 3,e two update phases were completed. Update'92 focused on the initial analysis of evaluation data collected from software engineering students at Auburn University and the addition of significant enhancements to the user interface. Update'93 (the current update) focused on the statistical analysis of the data collected in the previous update and preparation of Version 3.4 of the prototype for limited distribution to facilitate further evaluation. The current prototype provides the capability for the user to generate CSDs from Ada PDL or source code in a reverse engineering as well as forward engineering mode with a level of flexibility suitable for practical application. This report provides an overview of the GRASP/Ada project with an emphasis on the current update.

ACKNOWLEDGEMENTS

We appreciate the assistance provided by NASA personnel, especially Mr. Keith Shackelford whose guidance has been of great value. The following is an alphabetical listing of the team members who have participated in various phases of the project. An asterisk (*) indicates the team member worked on the Update of GRASP/Ada addressed in this report.

Principal Investigator:

* Dr. James H. Cross II, Associate Professor
  Computer Science and Engineering

Graduate Research Assistants:

  Richard A. Davis
  Charles H. May
* Kelly I. Morrison
  Timothy A. Plunkett
* Narayana S. Rekapalli
  Darren Tola

Statistician

* Dr. Saeed Magssoodloo, Professor
  Industrial Engineering

The following trademarks are referenced in the text of this report.

Ada is a trademark of the United States Government, Ada Joint Program Office.
AdaVision is a trademark of Sun Microsystems, Inc.
Apex is a trademark of Rational.
ObjectMaker is a trademark of Mark V Systems, Inc.
PostScript is a trademark of Adobe Systems, Inc.
Software through Pictures (StP), Ada Development Environment (ADE), and **IDE** are trademarks of Interactive Development Environments.
VAX and **VMS** are trademarks of Digital Equipment Corporation.
VERDIX and **VADS** are trademarks of Verdix Corporation.
UNIX is a trademark of AT&T.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1.0 Introduction

Computer professionals have long promoted the idea that graphical representations of software can be extremely useful as comprehension aids when used to supplement textual descriptions and specifications of software, especially for large complex systems [SHU88, AOY89, SCA89]. The general goal of this research has been the investigation, formulation and generation of *graphical representations of algorithms, structures, and processes for Ada* (GRASP/Ada). This specific task has focused on *reverse engineering* of control structure diagrams from Ada PDL or source code.

Reverse engineering normally includes the processing of source code to extract higher levels of abstraction for both data and processes. The primary motivation for reverse engineering is increased support for software reusability, verification, and software maintenance, all of which should be greatly facilitated by automatically generating a set of "formalized diagrams" to supplement the source code and other forms of existing documentation. The overall goal of the GRASP/Ada project is to provide the foundation for a CASE (computer-aided software engineering) environment in which reverse engineering and forward engineering (development) are tightly coupled. In this environment, the user may specify the software in a graphically-oriented language and then automatically generate the corresponding Ada code [ADA83]. Alternatively, the user may specify the software in Ada or Ada/PDL and then automatically generate the graphical representations either dynamically as the code is entered or as a form of post-processing.

The GRASP/Ada project was divided into three primary development phases followed by two update phases: Update'92 and Update'93, the current phase. Each of these phases is briefly described below.

1.1 Phase 1 - The Control Structure Diagram For Ada

Phase 1 focused on a survey of graphical notations for software with concentration on detailed level diagrams such as those found in [MAR85, TRI89], and the development of a new algorithmic or PDL/code level diagram for Ada. Tentative graphical control constructs for the *Control Structure Diagram* (CSD) were created and initially prototyped in a VAX/VMS environment. This included the development of special diagramming fonts for both the screen and printer and the development of parser and scanner using UNIX based tools such as LEX and YACC. The CSD is described in Section 2.0.

1.2 Phase 2 - The GRASP/Ada Prototype and User Interface

During Phase 2, the prototype was extended and ported to a Sun/UNIX environment. The development of a user interface based on the X Window System represented a major part of the extension effort. Verdix Ada and the Verdix DIANA interface were acquired as potential commercial tools upon which to base the GRASP/Ada prototype. Architectural diagrams for Ada were surveyed and the OOSD notation [WAS89] was identified as having good potential for accurately representing many of the varied architectural features of an Ada

software system. Phase 2 also included the preliminary design for an architectural CSD [DAV90]. The aspects of architectural CSD are expected to be integrated into the fully operational GRASP/Ada prototype during a future phase of the project.

1.3 Phase 3 - CSD Generation Prototype and Preliminary Object Diagram Prototype

Phase 3 has had two major thrusts: (1) completion of an operational GRASP/Ada prototype which generates CSDs and (2) the development of a preliminary prototype which generates object diagrams directly from Ada source code. Completion of the GRASP/Ada CSD prototype (CSDgen) included the addition of substantial functionality, via the User Interface, to make the prototype easier to use. The User Interface was reworked based on the Athena widget set. CSDgen was installed and demonstrated on a Sun workstation at Marshall Space Flight Center, Alabama.

The development of a preliminary prototype for generating architectural object diagrams (ODgen) for Ada source/PDL was an effort to determine feasibility rather than to deliver an operational prototype as was the case with CSD generator above. The preliminary prototype has indicated that the development of the components to recover the information to be included in the diagram, although a major effort, is relatively straightforward. However, the research has also indicated that the major obstacle for automatic object diagram generation is the automatic layout of the diagrams in a human readable and/or aesthetically pleasing format. A user extensible rule base, which automates the diagram layout task, is expected to be formulated during future GRASP research.

1.4 Update'92 - Preliminary Evaluation and User Interface Enhancements

Following Phase 3, the Version 3.1 prototype was used in several software engineering classes at Auburn University, evaluated, and enhanced to create Version 3.2. A preliminary analysis was done on data collected from students, and then changes were made to the User Interface to reflect the indicated usage patterns. The UNIX man-page was drafted to provide online documentation, and the installation guide was drafted to provide for limited distribution of the tool. And finally, the prototype was modified so that it could be invoked from IDE's CASE tool StP with a pspec or PDL file (see Appendix B).

1.5 Update'93 of the GRASP/Ada

Update'93 is the most recent phase of the GRASP/Ada project, and the one described in the remainder of this report. Since Update'92, the Version 3.2 prototype has undergone continual upgrades which have resulted in Version 3.4 of the prototype. The CSD evaluation data collected in the previous update was formally analyzed during Update'93 and the User Manual and Installation Guide have been completed. Each of these tasks is briefly described below.

(1) **The GRASP/Ada tool was evaluated.** As part of the ongoing evaluation of GRASP/Ada, the tool has been and continues to be used in CSE 422 (Introduction to

Software Engineering). An evaluation instrument was developed and administered to collect feedback from the students during the Fall 1992 quarter. Both the CSD and the GRASP/Ada tool were evaluated. A preliminary analysis of the data was done during Update'92. During Update'93 a formal statistical analysis was applied to the CSD evaluation data to determine statistical significance, if any, in the preference for the CSD over other common graphical representations such as the flowchart. The results were, in fact, statistically significant and are described in Section 6.

(2) **Modification of the GRASP/Ada tool.** As a result of the GRASP/Ada tool evaluation described above and in Section 6, numerous modifications and enhancements were made to the User Interface. The current User Interface is described in Section 4.

(3) **The Installation Guide, Man-Page, and online User Manual.** The make files were streamlined to make recompilation and installation more straightforward. The Installation Guide (see Appendix A) includes sections on the system requirements, installation procedure, and getting started. Although the window-based User Interface is relatively intuitive, one of items requested most by the students that evaluated the prototype was a User Manual. The UNIX *Man Page* has been updated to provide all necessary user information, and an online User Manual which allows the user to view individual topics via the User Interface has been included.

(4) **Investigation of Object Diagrams.** Existing CASE tools that utilize object diagrams in one form or another were reviewed. *ObjectMaker* by Mark V Systems was acquired for evaluation purposes as an example of currently available software tools that support object diagrams and other architectural level graphical representations. The results of the review indicate that the GRASP/Ada tool can play an important role as a natural extension to these existing object diagrams and their supporting CASE tools. The control structure diagram provides a detailed algorithmic level graphical representation which has statistically significant advantages over other graphical notations, PDL, and source code. The entire spectrum of re-engineering oriented CASE tools with respect to functionality and availability has recently been well-documented in [OLS93, SIT92] .

(5) **Presentation of Update'93.** Specifications for this update and the expected results of this update (Update'93) were presented informally during a panel session at the 5th International Conference on Software Engineering and Knowledge Engineering (SEKE'93), June 16-18, 1993 in San Francisco [CAL93]. The actual results are expected to be presented and published at appropriate professional meetings and/or in an appropriate journal following this update phase.

The following sections describe the control structure diagram, the GRASP/Ada system model, the user interface, the control structure diagram generator, evaluation of the CSD and prototype, and future requirements. The overall rationale for the development of the CSD is described in [CRO90a, CRO90b], which were written during Phase 1. A taxonomy and extensive literature review of reverse engineering can be found in [CHI90, CRO92], which were written during Phases 2 and 3.

2.0 The Control Structure Diagram

Advances in hardware and software, particularly high-density bit-mapped monitors and window-based user interfaces, have led to a renewed interest in graphical representation of software. Although much of the research activity in the area of software visualization and computer-aided software engineering (CASE) tools has focused on architectural-level charts and diagrams, the complex nature of the control constructs and control flow defined by programming languages such as Ada and C and their associated PDLs, makes source code and detailed design specifications attractive candidates for graphical representation. In particular, source code should benefit from the use of an appropriate graphical notation since it must be read many times during the course of initial development, testing and maintenance. The control structure diagram (CSD) is a notation intended specifically for the graphical representation of algorithms in detailed designs as well as actual source code. The primary purpose of the CSD is to reduce the time required to comprehend software by clearly depicting the control constructs and control flow at all relevant levels of abstraction. The CSD is a natural extension to existing architectural graphical representations such as data flow diagrams, structure charts, and object diagrams.

The CSD, which was initially created for Pascal/PDL [CRO88], has been extended significantly so that the graphical constructs of the CSD map directly to the constructs of Ada. The rich set of control constructs in Ada (e.g. task rendezvous) and the wide acceptance of Ada/PDL by the software engineering community as a detailed design language made Ada a natural choice for the basis of a graphical notation. A major objective in the philosophy that guided the development of the CSD was that the graphical constructs should supplement the code and/or PDL without disrupting their familiar appearance. That is, the CSD should appear to be a natural extension to the Ada constructs and, similarly, the Ada source code should appear to be a natural extension of the diagram. This has resulted in a concise, compact graphical notation which attempts to combine the best features of diagraming with those of well-indented PDL or source code.

2.1 Background

Graphical representations have been recognized as having an important impact in communicating from the perspective of both the "writer" and the "reader." For software, this includes communicating requirements between users and designers and communicating design specifications between designers and implementors. However, there are additional areas where the potential of graphical notations have not been fully exploited. These include communicating the semantics of the actual implementation represented by the source code to personnel for the purposes of testing and maintenance, each of which are major resource sinks in the software life cycle. In particular, Selby [SEL85] found that code reading was the most cost effective method of detecting errors during the verification process when compared to functional testing and structural testing. And Standish [STA85] reported that program understanding may represent as much as 90% of the cost of maintenance. Hence, improved comprehension efficiency resulting from the integration of graphical notations and source code could have a significant impact on the overall cost of software production.

Since the flowchart was introduced in the mid-50's, numerous notations for representing algorithms have been proposed and utilized. Several authors have published notable books and papers that address the details of many of these [MAR85, TRI88, SHN77]. Tripp, for example, describes 18 distinct notations that have been introduced since 1977 and Aoyama et.al. describes the popular diagrams used in Japan. In general, these diagrams have been strongly influenced by structured programming and thus contain control constructs for sequence, selection, and iteration. In addition, several contain explicit EXIT structures to allow single entry / multiple exit control flow through a block of code, as well as PARALLEL or concurrency constructs. However, none the diagrams cited explicitly contains all of the control constructs found in Ada.

Graphical notations for representing software at the algorithmic level have been neglected, for the most part, by business and industry in the U.S. in favor of non-graphical PDL. A lack of automated support and the results of several studies conducted in the seventies which found no significant difference in the comprehension of algorithms represented by flowcharts and pseudo-code [SHN77] have been a major factors in this underutilization. However, automation is now available in the form of numerous CASE tools and recent empirical studies reported by Aoyami [AOY89] and Scanlan [SCA89] have concluded that graphical notations may indeed improve the comprehensibility and overall productivity of software. Scanlan's study involved a well-controlled experiment in which deeply nested if-then-else constructs, represented in structured flowcharts and pseudo-code, were read by intermediate-level students. Scores for the flowchart were significantly higher than those of the PDL. The statistical studies reported by Aoyami et.al. involved several tree-structured diagrams (e.g., PAD, YACC II, and SPD) widely used in Japan which, in combination with their environments, have led to significant gains in productivity. The results of these recent studies suggest that the use of a graphical notation with appropriate automated support for Ada/PDL and Ada should provide significant increases productivity over current non-graphical approaches.

2.2 The Control Structure Diagram Illustrated

Two examples are presented below to illustrate the CSD. The first shows the basic control constructs of sequence, selection and iteration in Ada. These three control constructs are common to all structured procedural languages such as Ada, C, and Pascal. The second example illustrates a more complex control construct, the task rendezvous in Ada.

Figure 1 contains an Ada procedure called SearchArray that searches an array A of elements and counts the number of elements above, below, and/or equal to a specified element. Figure 2 contains the CSD for SearchArray which includes the three basic control constructs sequence, selection, and iteration. Although this is a very simple example, the CSD clearly indicates the levels of control inherent in the nesting of control statements. For example, at level 1 there are four statements executed in sequence - the three assignment statements and the *for* loop. The *for* loop defines a second level of control which contains a single statement, the *if* statement, which in turn defines three separate level 3 sequences, each of which contains one assignment statement. It is noteworthy that even the CSDs for most production strength procedures rarely contain more than ten statements at level 1 or in any of the subsequences defined by control constructs for selection and iteration. This graphical chunking on the basis of functionality and level of control appears to have a

```
procedure SearchArray (A :  in ArrayType;
    Element:  in ElementType;
    Above,Below, EqualTo: out integer)   is

begin
   Above := 0;
   Below := 0;
   EqualTo := 0;
   for index in A'first..A'last loop
      if Element > A(index) then
         Below := Below + 1;

      elsif Element < A(index) then
         Above := Above + 1;

      else
         EqualTo := EqualTo + 1;

      end if;
   end loop;
end SearchArray;
```

Figure 1. Ada Source for SearchArray.

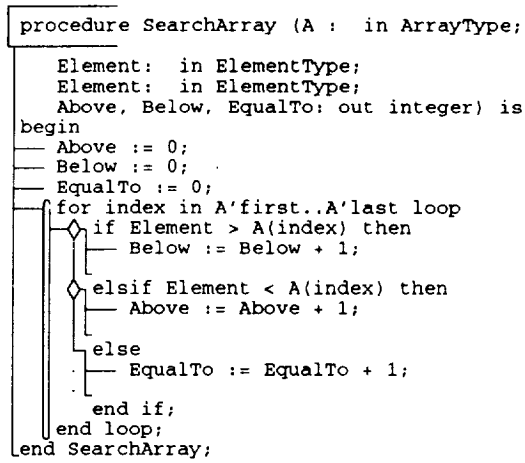

Figure 3. CSD for SearchArray.

substantial positive effect on detailed comprehension of the software.

Figures 3 and 4 contain an Ada task body CONTROLLER adapted from [BAR84], which loops through a priority list attempting to accept selectively a REQUEST with priority P. Upon on acceptance, some action is taken, followed by an exit from the priority list loop to restart the loop with the first priority. In typical Ada task fashion, the priority list loop is contained in an outer infinite loop. This short example contains two threads of control: the rendezvous, which enters and exists at the accept statement, and the thread within the task body. In addition, the priority list loop contains two exits: the normal exit at the beginning of the loop when the priority list has been exhausted, and an explicit exit invoked within the

```
task body TASK_NAME is

begin
   loop
      for p in PRIOITY loop
         select


            accept REQUEST(p) (D: DATA) do


               ACTION(D);

            end;
            exit;


         else
            null;

         end select;
      end loop;
   end loop;
end TASK_NAME;
```

Figure 4. Ada Source for Task Body Controller.

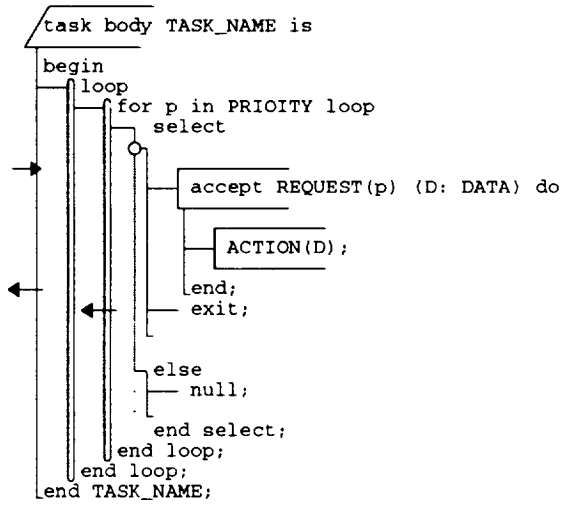

Figure 2. CSD for Ada Task Body Controller.

select statement. While the concurrency and multiple exits are useful in modeling the solution, they do increase the effort required of the reader to comprehend the code.

The CSD in Figure 4 uses intuitive graphical constructs to depict the point of rendezvous, the two nested loops, the select statement guarding the accept statement for the task, the unconditional exit from the inner loop, and the overall control flow of the task. When reading the code without the diagram, as shown in Figure 3, the control constructs and control paths are much less visible although the same structural and control information is available. With additional levels of nesting and increased physical separation of sequential components, the visibility of control constructs and control paths becomes increasingly obscure, and the effort required of the reader dramatically increases in the absence of the CSD. Now that the CSD has been briefly introduced, the various CSD constructs for Ada are presented in Figure 5. Each of the CSD constructs should be relatively self-explanatory since the CSD is designed to supplement the semantics of the underlying Ada.

2.3 Observations

The control structure diagram is a new graphical tool which maps directly to Ada and Ada PDL. The CSD offers advantages over previously available diagrams in that it is combines the best features of PDL and code with simple intuitive graphical constructs. The potential of the CSD can be best realized during detailed design, implementation, verification and maintenance. The CSD can be used as a natural extension to popular architectural level representations such as data flow diagrams, object diagrams, and structure charts.

The GASP/Ada prototype, described in Sections 4 and 5, provides for the automatic generation of the CSD from Ada or Ada PDL. A preliminary statistical evaluation of the CSD is presented in Section 6.

2.4 Control Structure Diagram - Future Directions

The CSD constructs shown in Figure 5 are expected to continue to evolve, especially with Ada 9X on the horizon. Suggestions for improvements to the individual CSD graphical constructs are continually solicited from users. While most of these suggested changes appear to be minor when considered individually, their aggregate implementation in the current prototype represents a major rework. Theoretically, the CSD and its individual constructs are a separate issue from the automatic generation of the diagrams in a production environment. However, in practice unless CSDs (or any other diagrams) can be automatically generated, they will not be utilized. Additional future considerations concerning the overall system model, the user interface, and the automatic generation of the CSD can be found at the end of Sections 3, 4, and 5 respectively.

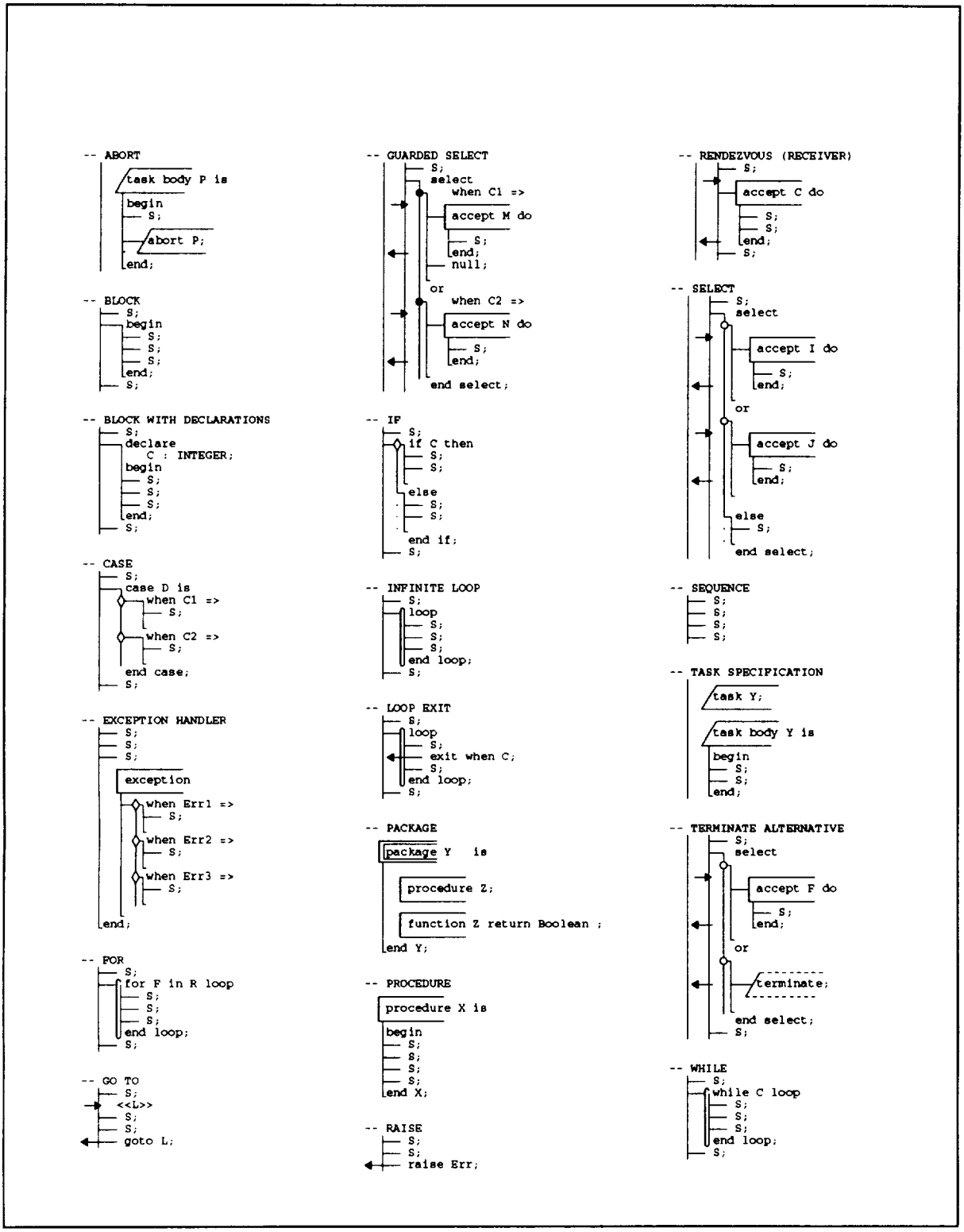

Figure 5. Control Structure Diagram Constructs for Ada.

3.0 The GRASP/Ada System Model

3.1 Overview

The major system components of the GRASP/Ada system are shown in the block diagram in Figure 6. Currently, the entire prototype is written in the programming language C. The **User Interface** was built using the X Window System and includes a special CSD window (modified text editor) and provides general control and coordination among the other components.

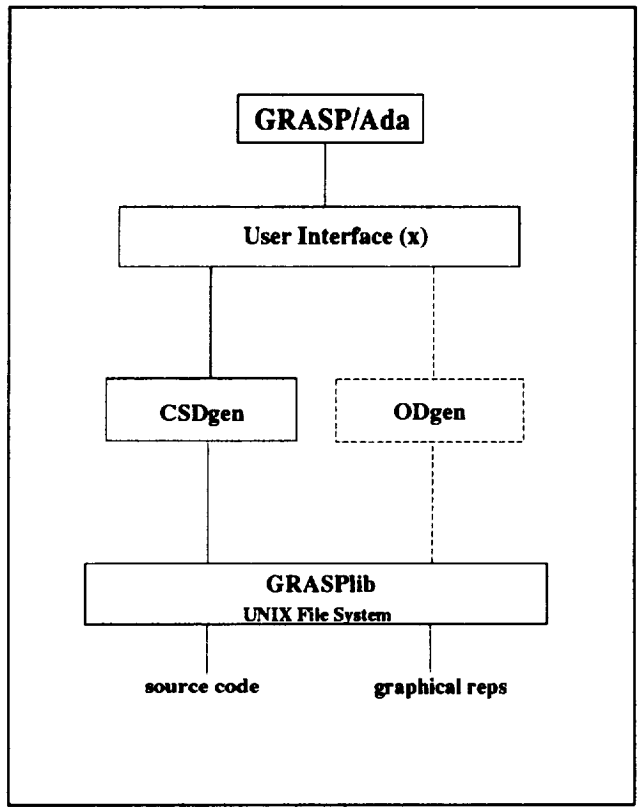

Figure 6. GRASP/Ada System Block Diagram.

The control structure diagram generator, **CSDgen**, inputs Ada PDL or source code and produces a CSD. CSDgen has its own parser/scanner built using FLEX and BISON, successors of LEX and YACC. It also includes its own printer utilities. As such, CSDgen is a self-sufficient component which can be executed from the user interface or the command line without the commercial components. When changes are made to the Ada PDL or source code, the entire file must be reparsed to produce an updated CSD. A CSD editor, which will provide for dynamic incremental modification of the CSD, is currently in the planning stages.

The object diagram generation component, **ODgen**, is in the analysis phase and has been implemented as a separate preliminary prototype. The dashed lines indicate future integration. The feasibility of automatic diagram layout remains under investigation. Beyond automatic diagram layout, several design alternatives have been identified. The major alternatives include the decision of whether to attempt to integrate GRASP/Ada directly with commercial components. For example, the Verdix Ada development system (VADS) and DIANA interface could be used for extraction of diagram information and (2) IDE's Software through Pictures, Ada Development Environment (IDE/StP/ADE) for the display of the object diagrams. *ObjectMaker* by Mark V Systems has also been reviewed and is a strong contender as a basis for generating and displaying object diagrams. ObjectMaker was recently ported to UNIX from a PC platform, and unfortunately it appeared to be somewhat unstable in its current state.

The GRASP/Ada library component, **GRASPlib**, allows for coordination of all generated items with their associated source code. The current file organization uses standard UNIX directory conventions as well as default naming conventions. For example, all Ada source files end in ***.a***, the corresponding CSD files end in ***.a.csd***, and the corresponding print files end in ***.a.csd.ps***. In the present prototype, library complexity has been keep to a

minimum by relying on the UNIX directory organization. Its purpose is to facilitate navigation among the diagrams and the production of sets of diagrams.

3.2 System Model - Future Directions

The GRASP/Ada tool was conceived to be self-sufficient reverse engineering tool that would generate control structure diagrams primarily, and then architectural level diagrams (e.g., object diagrams) secondarily, all from Ada source code or PDL. An alternative to the model presented in Figure 6 would be to concentrate on issues that improve the integration capabilities of GRASP/Ada with commercially available CASE tools and programming environments. The following are potential tasks.

(1) A CSD editor/generator should include full syntax checking with appropriate error messages. Appropriate hooks to and from the compiler and debugger should be considered.

(2) Appropriate hooks to/from the GRASP/Ada tool are necessary to facilitate integration with commercially available CASE tools and programming environments.

(3) GRASP/Ada system model should include the capability of running in a stand-alone mode in which several CSD windows are to be coordinated are coordinated by a main window, similar to the current Version 4.3 prototype.

(4) GRASP/Ada system model should include the capability of running in a single window mode, similar to opening an X Windows textedit application.

(5) Finally, GRASP/Ada system model should include the capability of running in a single window mode, as an extension of a commercial CASE tool. For example, when clicking on an object or module in an architectural diagram, a CSD window could be opened with the PDL or source code represented in a CSD rather than simply text. Many commercial tools are competing in this market. One of particular interest is Rational's Ada programming development environment for UNIX (*Apex*), which has recently been made available to universities. Apex is a state-of-the-art environment which supports the Ada Semantic Interface Specification (ASIS) to facilitate tool integration. The GRASP/Ada tool with a CSD editor/generator could play an integral role in software development, maintenance, and reengineering when integrated with an environment such as *Apex*.

4.0 User Interface

GRASP/Ada user interface was developed using the X Window System, Version 11 Release 4 (X11R4). The X Window System, or simply X, meets the GRASP/Ada user interface requirements of an industry-standard window based environment which supports portable graphical user interfaces for application software. Some of the key features which make X attractive for this application are its availability on a wide variety of platforms, unique device independent architecture, adaptability to various user interface styles, support from a consortium of major hardware and software vendors, and low acquisition cost. With its unique device independent architecture, X allows programs to display windows on any hardware that supports the X Protocol. X does not define any particular user interface style or policy, but provides mechanisms to support many various interface styles.

The specifications and figures that follow are intended to define the look and feel of the GRASP/Ada User Interface as well as indicate much of the current and planned functionality of the CSD generator. The Man Page provides additional information.

4.1 System Window

The System window, shown in Figure 7, provides the user with the overall organization and structure of the GRASP/Ada tool. Option buttons include: General and Control Structure Diagram. These are briefly described below. A future button is planned for Object Diagram.

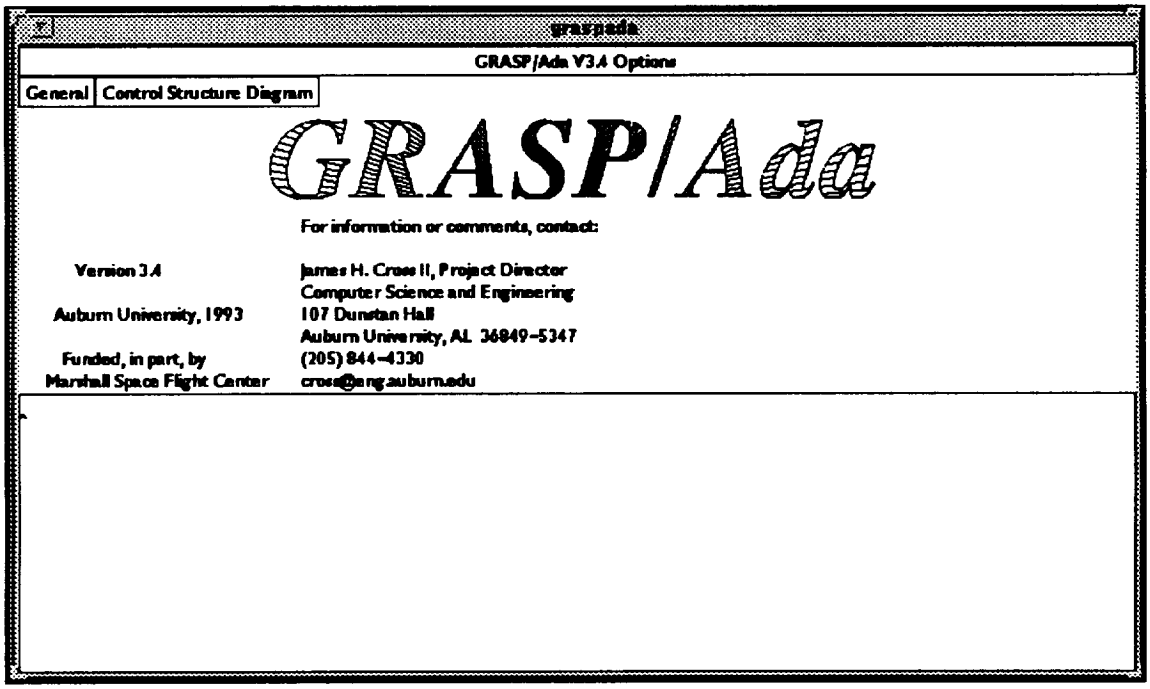

Figure 7. GRASP/Ada System Window.

General - Provides for selection of a printer access to the user manual (see Figure 8).

Print ... - Allows the user to select among several printers or the user may enter a printer name. The list of printers is contained in the fileGRASP.printers located in the diectory $GRASP_HOME/lib.

User Manual - Allows the user to open the GRASP/Ada online User Manual which provides an interactive index of topics. This information is also contained in the UNIX man page.

Quit - Allows the user to exit the GRASP/Ada system.

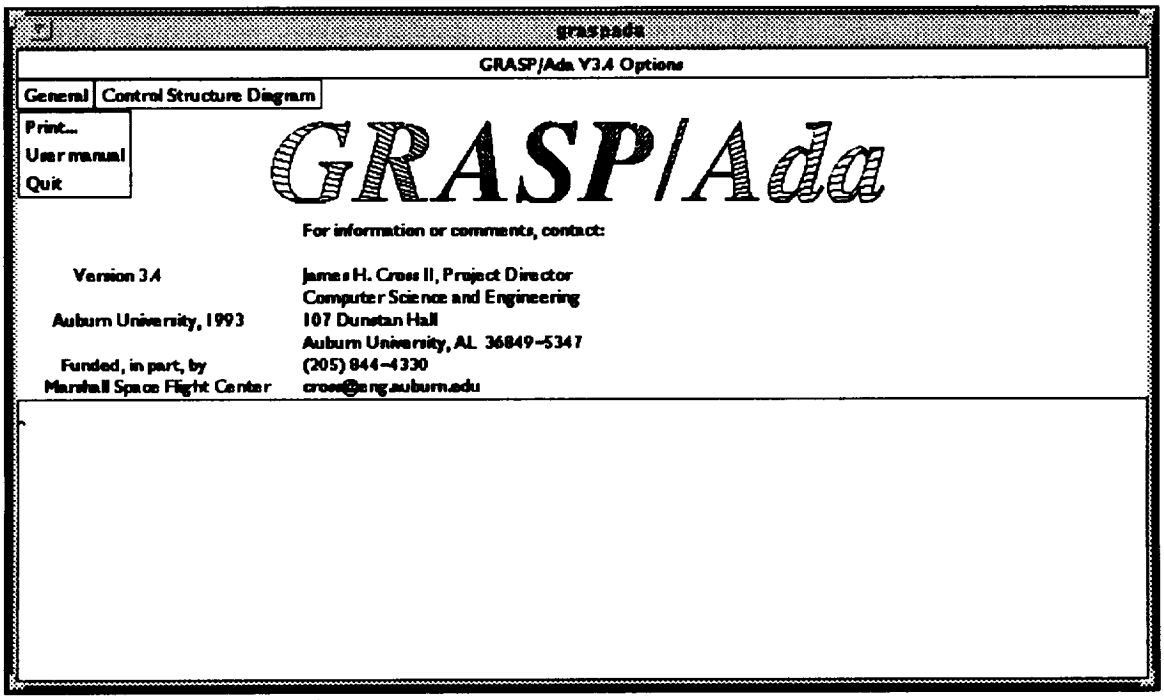

Figure 8. General Options.

Control Structure Diagram - Allows the user to open one or more CSD windows, close all CSD windows that are currently open, and generate CSDs in a batch mode (see Figure 9). In addition, a list of all CSD windows currently open is presented to the user.

Open CSD window - Opens a CSD window and adds it to the list of open CSD windows. This list is displayed at the end of the options pop-up menu by appending the name of the respective file in each CSD window. In Figure 9, the file /dev/null has been appended which indicates that a CSD window has been opened but no file has been loaded or saved. This list allows the user to see quickly how many CSD windows are active and what file is associated with each.

Close all CSD windows - Closes all CSD windows. Currently, this "quits" each CSD window without querying the user regarding unsaved changes. When a CSD window is "quit" from the CSD window file option menu, the user is queried if changes have been made since the last save.

Generate CSD ... - Allows the user to generate a set of CSDs by entering the file name using standard wildcard notation. For example, entering **.a* would generate a CSD for each file with a ".a" extension. A CSD generation **summary window** displays the progress of the generation by listing each file as it is being processed and any resulting error messages. The summary concludes with number of files processed and the number of errors encountered. The default for each CSD file name is the source file name with **.csd** appended. If an error is encountered, an extension of **.err** is used. As the CSDs are generated, the GRASP library is updated, which currently consists of populating a specified directory with file images of the CSDs. Since GRASP/Ada is expected to be used to process and analyze large existing Ada

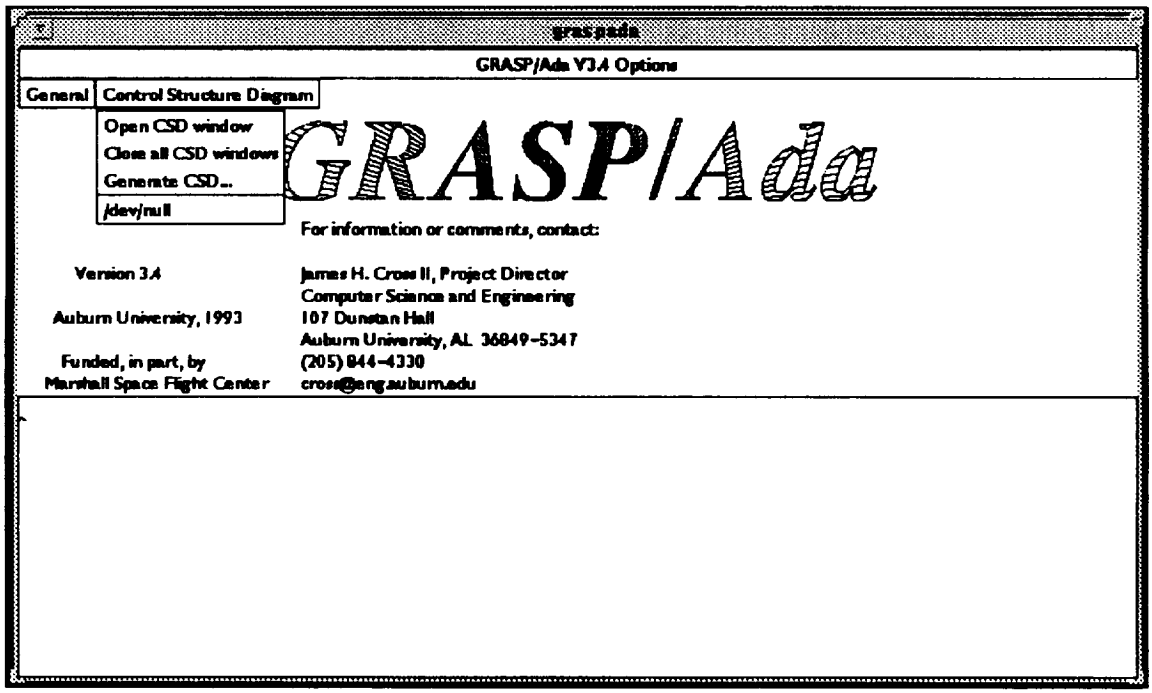

Figure 9. CSD Options.

software systems consisting of perhaps hundreds of files, the option to generate a set of CSDs in batch mode is particularly useful.

4.2 Control Structure Diagram Window

The CSD window, shown in Figure 10 with file options displayed, provides the user with capabilities for generating and viewing a CSD for an Ada PDL or source file. Multiple CSD windows may be opened to access several CSD files at once. CSD file names and their associated directory paths are selected under the File option and displayed at the top of each window. Figure 11 shows a CSD window after a procedure provided by NASA has been loaded and the CSD generated by clicking **Regenerate CSD** on the menu selection bar or by clicking **Generate CSD** under **File** options. In the current version of GRASP/Ada, generation of the CSD is done on a file-level basis where each file contains one or more units. When changes are made to the source code, the entire CSD for the file involved is regenerated. Future versions of GRASP/Ada will address incremental regeneration of the CSD in conjunction with editing capabilities in the CSD window.

The CSD window Options

File - Allows the user to select from numerous options (see Figure 10) including the following.

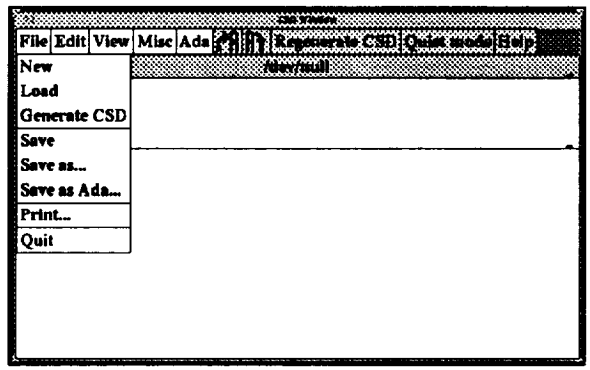

Figure 10. CSD Window File Options.

> **New**- Removes existing file, if any, from CSD window. Currently, this option is disabled, and the user must load a new file or explicitly delete the contents of the CSD window to accomplish the function of **new**.

Load - Loads a CSD file. A window is presented that allows the user to navigate among directories and select a file.

Generate CSD - Generates a CSD from source code and/or regenerates a CSD after modification. When the CSD window is opened and loaded with a source file without a *.csd* extension, a separate CSD window is automatically opened to display the CSD when it is generated. Note that CSD graphics characters, if any, are filtered prior to the parse or reparse. Currently, this option is the same as **Regenerate CSD** on the menu selection bar of the CSD window (described below). Figure 11 shows an Ada procedure provided by NASA after the CSD has been generated.

Save - Saves the CSD file with the same name that was loaded.

Save as ... - Saves the CSD file with a new name. This options allows the user to selected the directory and name from a list.

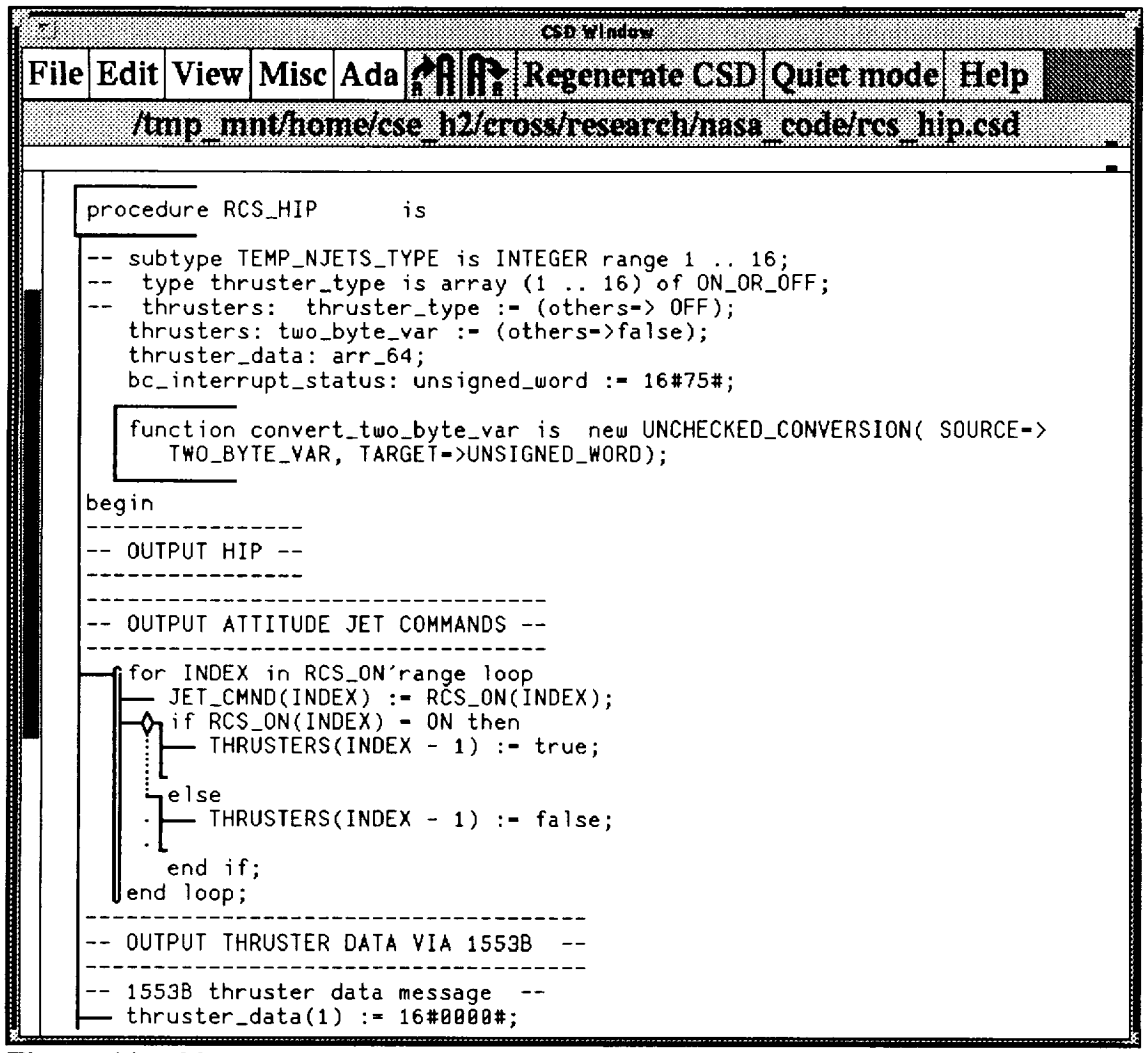

Figure 11. CSD Window with Procedure Provided by NASA after CSD Generation.

Save as Ada - Filters the CSD characters from the CSD file and writes to a file with a *.a* extension.

Print - Presents a window which allows the user to select various print options such as point size, page numbers, and header, and then generates a PostScript file (.ps) from the .csd file and sends it to the selected printer.

Quit - Closes the CSD window.

Edit - Allows the user to do traditional text editing functions of cut,copy, paste, and search (see Figure 12).

Cut ... - Allows the user to deleted highlighted text. This option is currently disabled, and the user must press CTRL-W to accomplish a **cut**.

Copy ... - Copies the highlighted text to buffer. This option is currently disabled, and the user must click the right mouse button to accomplish a **copy**.

Paste ... - Inserts the previously "cut" or "copied" text from the buffer. This option is currently disabled, and the user must click the middle mouse button to accomplish a **copy**.

Search ... - Opens a textedit search window. Currently, this option is disabled, and the user must press CTL-s to activate the search window.

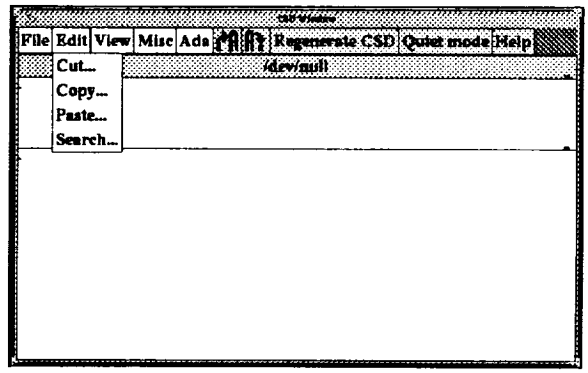

Figure 12. CSD Window Edit Options.

View - Currently allows the user to select one of several window font sizes ranging from 9 points to 24 points (see Figure 13). The default font size for the CSD window is 13 points. Option AA-AA described below also facilitates the selection of font sizes.

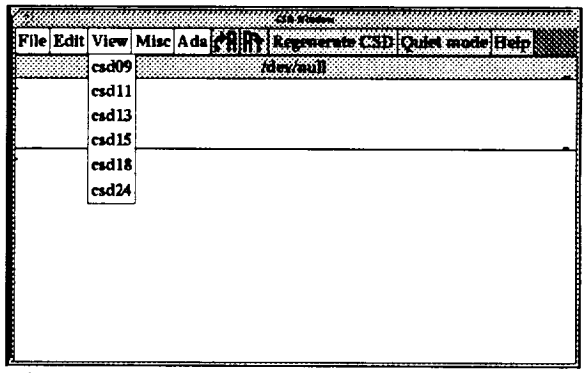

Figure 13. CSD Window View Options.

Misc - Allows the user to *show* or *hide* The CSD character panel, set compiler (future option), invoke interactive User Manual, and load window fonts. With the CSD character panel visible, CSD characters can be inserted directly into the current window, primarily for the purpose of experimentation (see Figure 14).

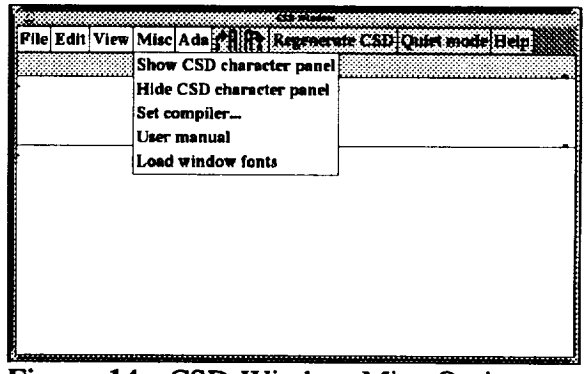

Figure 14. CSD Window Misc Options.

Show CSD character panel - Displays the CSD character set in a panel in the lower section of the CSD Window. This option allows the user to position the curser in the edit window and then click on a CSD character in the panel to insert it at the cursor position. This is useful for experimenting with alternative graphical constructs and diagram layouts. Regenerating the CSD results in the removal of all added CSD characters.

Hide CSD character panel - Removes the panel of CSD characters if it is currently dispalayed.

Compile - Allows the user to enter the name of an Ada compiler to be called from the CSD window. Currently, the CSD window **file** options do not support the invocation of a compiler. Experience has indicated that it is more practical to invoke

the compiler from a separate window. In particular, GRASP/Ada can be called as the Ada editor from an Ada development tool such as Sun's AdaVision. This allows the CSD to be utilized for viewing and editing the Ada source code while taking full advantage of the Ada development environment.

Ada - Displays Ada control constructs and enables the user to insert them directly into the current window at the location of the curser (see Figure 15). A syntactically correct program can be constructed quickly using this option. Figure 16 shows a program structure resulting from four clicks on the Ada constructs: *procedure body*, *loop*, *if/then/else*, *for loop*, and *while loop*. The template placeholders can be modified or replaced as necessary.

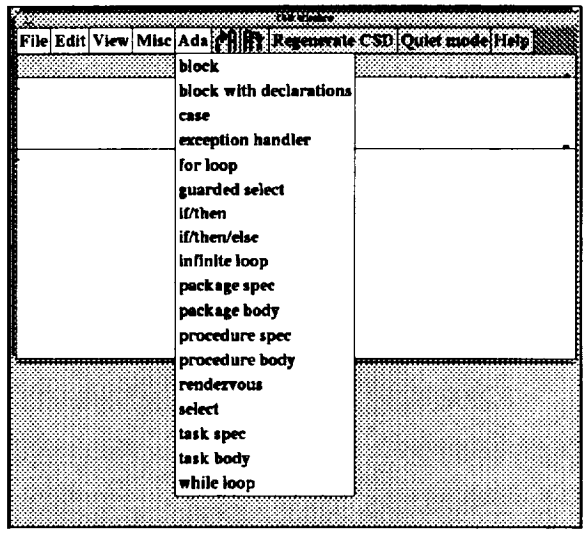

Figure 15. CSD Window with Ada Constructs.

AA AA - Allows the user to increase or decrease the font size for the current window, thus shrinking or expanding the overall size of the CSD.

Regenerate CSD - Allows the user to quickly regenerate a CSD after making changes. Currently, this option is the same as **Generate CSD** on the **File** submenu.

Quiet mode / Verify mode - Allows the user to toggle between to modes of regeneration. Quiet mode assumes that the existing files should be overwritten during generation or regeneration, and Verify mode queries the user before continuing.

4.3 User Interface - Future Directions

The User Interface is expected to continue to evolve, especially as new functionality is added. In particular, implementation based on alternate widget sets is under consideration as well as utilization with other window manager. The requirements definition and design of the current version were done in a learning mode under a schedule that required an operational prototype to be implemented quickly. As a result, many of the features, such as placement of options, are expected to be streamlined considerably. However, the current prototype is suitable for limited practical application, and information collected from current users is expected to have a positive effect on the overall evolution of the prototype. Two rather immediate future tasks are described below.

(1) **Move to the Motif widget set** - Currently, the user interface is implemented using the Athena widget set which is provided with the MIT distribution tape of the X Windows System. Recently, the Motif widget set was adopted as an industry

standard. By moving to Motif, the GRASP/Ada user interface will have the same "look and feel" as most commercial CASE tools in the near future.

(2) **Simplification of the User Interface** - In order to streamline the integration of GRASP/Ada with commercial CASE tools, the user interface should be simplified. The appropriate time to do this would be during the move to Motif described above.

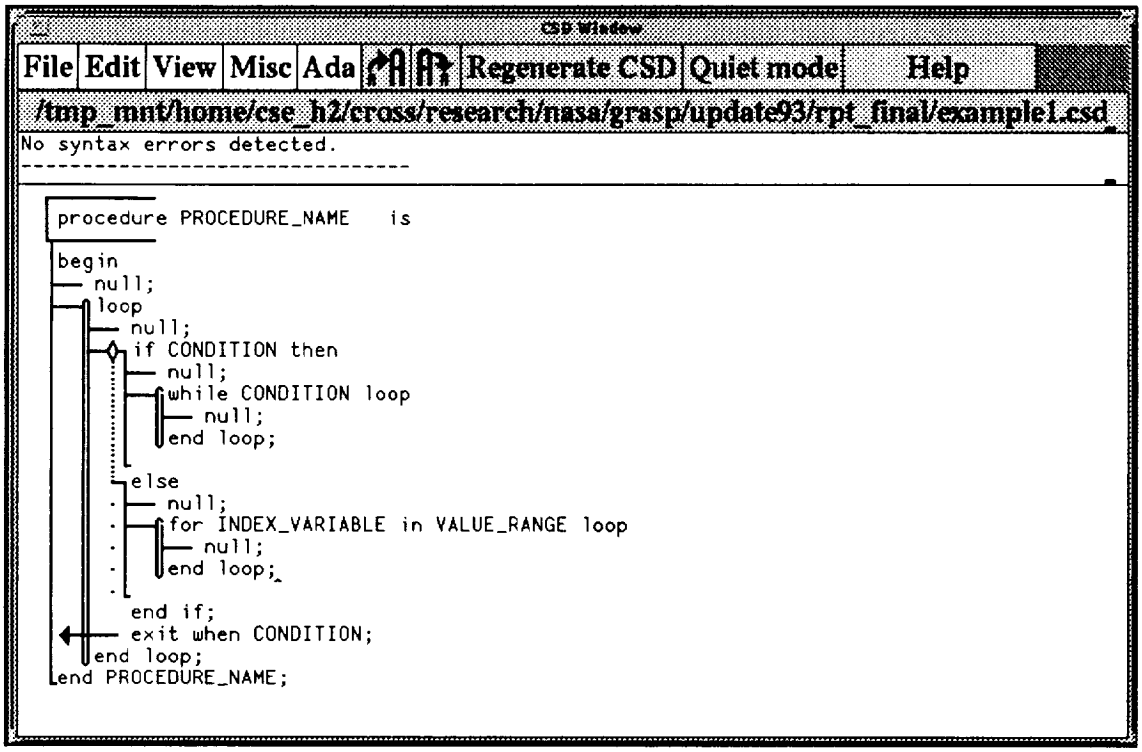

Figure 16. CSD Window with Program Structure Resulting from Clicking on *procedure body*, *if/then/else*, *for loop*, and *while loop*.

5.0 Control Structure Diagram Generator

The GRASP/Ada control structure diagram generator (CSDgen) is described in this section from a technical and developmental perspective. Since display mechanisms for both the screen and printer are an integral part of the CSDgen application, these are included in this section as well. A more complete history and rationale for the development of the CSD is contained in [CRO90a, CRO90b]. The graphical constructs produced by CSDgen are summarized in Figure 5 (Section 2.0). Examples of the CSD are presented in conjunction with the User Interface in Section 4.0.

5.1 Generating the CSD

The primary function of CSDgen is to produce a CSD for a corresponding Ada source or PDL file. Although a complete parse is done during CSD generation, CSDgen assumes the Ada source code has been previously compiled and thus is syntactically correct. Currently, little error recovery and error reporting are attempted when a syntax error is encountered. The diagram is simply generated down to the point of the error. In the case of Ada PDL, non-Ada statements must be valid Ada identifiers so that they are treated like procedure calls. For example, the PDL for "search array for largest element" could be represented as "Search_array_for _largest_element" so that the phrase becomes a single identifier.

As indicated in Section 3, CSDgen was constructed using the UNIX tools LEX to build the lexical analyzer and YACC to build the parser for Ada. An Ada grammar was converted to YACC syntax and seeded with calls to action routines which generate the CSD as tokens in the source code are recognized as individual productions in the grammar. The current CSDgen prototype builds the diagram directly during the parse by inserting CSD graphics characters into a file along with text. To increase efficiency and improve extensibility, future versions of the CSDgen prototype may use a more abstract intermediate representation.

5.2 Displaying the CSD - Screen and Printer

Basic display capabilities to the screen and printer were implemented during Phase 2. Screen display is facilitated by sending the CSD file to a CSD window opened under an X Window manager. Printing is accomplished by converting the CSD file to a PostScript file and then sending it to a printer. Moving to a more abstract intermediate representation in future versions would necessitate the development of a new set of display routines which will be X Window System based.

CSD Screen Fonts. The default CSD screen font is a bitmap 13 point Courier to which the CSD graphic characters have been added. The font was defined as a bitmap distribution font (BDF) then converted to SNF format required by the X Window System. Four additional screen fonts ranging from 9 to 24 point are user selectable. These fonts were

later converted to OpenWindows fonts which has since become the version supported in the distribution tar file.

CSD Printer Fonts. CSD Printer fonts were initially developed for the HP LaserJet series. These were then implemented as PostScript type 3 fonts and all subsequent font development has been directed towards the PostScript font. The PostScript font provides the most flexibility since its size is user selectable from 1 to 300 points.

5.3 CSD Generator - Future Considerations

As indicated above, the actual generation of the CSD and the subsequent display of it on the screen or printer are in some ways inseparable. For example, margins, line spacing, and indentation of the CSD constructs could be part of the actual generation of the CSD or they could be a function of the display mechanism tightly coupled with a CSD editor. Hence, there is some degree of overlap in the discussion below regarding generating and displaying the CSD.

5.3.1 Generating the CSD - Future Considerations

Ada 9X. The Ada 9X specification will impact the CSD generator in at least two important ways.

> **(1) New 9X control constructs** - Additional CSD graphical constructs must be created as appropriate. Addressing the new 9X control constructs should be relatively straightforward. Numerous examples will need to be diagrammed and evaluated for comprehensibility and ease of implementation and integration into the current set of constructs.
>
> **(2) The Ada 9X specification allows all 256 ASCII character codes** - These are allowed to facilitate international character sets. The use of all 256 character codes presents somewhat more of a challenge. In the present prototype, ASCII codes above 128 have been used for the CSD graphics characters. Several operations in the user interface have freely filtered these character codes during the regeneration process as well as the "save as Ada" operation. Obviously, this approach will not be acceptable if the source code itself contains character codes in this range.

Internal Representation of the CSD - Alternatives. Several alternatives were considered for the internal representation of the CSD in the Version 3 prototype. Each has its own merits with respect to processing and storage efficiency and is briefly described below. These alternatives remain under consideration for future versions.

> **(1) Single ASCII File with CSD Characters and Text Combined.** This is the most direct approach and is currently used in the Version 3 prototype. The primary advantage of this approach is that combining the CSD characters with text in a single file eliminates the need for elaborate transformation and thus enables the rapid

implementation of prototypes as was the case in the previous phases of this project. The major disadvantages of this approach are that it does not lend itself to incremental changes during editing and the CSD characters have to be filtered if the user wants to regenerate the CSD or send the file to a compiler.

(2) **Separate ASCII Files for CSD Characters and Text.** In this approach, the file containing the CSD characters along with placement information would be "merged" with the prettyprinted source file. The primary advantage of the this approach is that the CSD characters would not have to be stripped out if the user wants to send the file to a compiler. The major disadvantage of this approach is that coordinating the two files would add complexity to generation and editing routines with little or no benefit. As a result, this approach would be more difficult to implement than the single file approach and not provide the advantages of the next alternative.

(3) **Single ASCII File Without Hard-coded CSD Characters.** This approach represents a compromise between the previous two. While it uses a single file, only "begin construct" and "end construct" codes are actually required for each CSD graphical construct in the CSD file rather than all CSD graphics characters that compose the diagram. In particular, no continuation characters would be included in the file. These would be generated by the screen display and print routines as required. The advantages of this approach would be most beneficial in an editing mode since the diagram could grow and shrink automatically as additional text/source code is inserted into the diagram. The extent of required modifications to text edit windows must be considered with this alternative.

Ada Comments. Currently, the location of Ada comments is not preserved by the CSD generator. Future CSD generators should attempt to preserve the original location to the degree possible.

Ada Coding Standards. Future CSD generators should provide the user with the capability to generate CSDs (together with prettyprinted source code) according to a prescribed standard. This may include conventions for keywords, identifiers, indentation, layout for compound statements, placement of comments, etc.

Syntax Error Messages. Currently, the CSD generator assumes the Ada source code is syntactically correct and makes little attempt to recover from encountered errors or to display meaningful error messages. This has been a major source of complaints from users who have used the CSD window to develop software in a forward direction rather than simply reverse engineering existing software. Future versions of the CSD generator could address this issue in one of two ways.

(1) CSD Generator could generate messages during the parse.

(2) Capabilities of current Ada development environments could be utilized to return the line number and descriptive message whenever an error is encountered during syntax checking and/or compiling. These could be sent to the CSD window opened

on the file or unit in question, and the offending element could be highlighted awaiting corrective action from the user.

Direct Generation Using the Ada Semantic Interface Specification (ASIS). If tight coupling and integration with a commercial Ada development system such as *Rational Apex* or *Verdix VADS* is desired, then the ASIS may provide for the direct generation of the CSD from the DIANA net or other underlying intermediate representation produced as a result of compilation. This would require a layer of software which traverses the DIANA net and calls the appropriate CSD primitives as control nodes are encountered. This approach would eliminate the possibility of directly editing the CSD since the DIANA interface does not support modifying the net, only reading it. In practice, it may prove more efficient to allow the CSD generator to simply reparse the entire compilable unit being edited.

Incremental Changes to the CSD. In the present prototype, there is no capability for incrementally modifying the CSD. When the CSD or source code is modified in the CSD window, the CSD must be regenerated by reparsing the entire file. While this has been sufficient for prototyping, especially for small programs, editing capabilities with incremental modification of the CSD may be desirable in an operational setting. The ASIS cited above may offer a bridge for these incremental changes to the CSD.

5.3.2 Displaying the CSD - Future Considerations

Layout/Spacing. The general layout of the CSD is highly structured by design. However, the user should have control over such attributes as horizontal and vertical spacing and the optional use of some diagramming symbols. In the current Version 3 CSDgen prototype, horizontal and vertical spacing are not user selectable. They are a part of the CSD file generation and are defaulted to single spacing with 80 characters per line. In order to change these, e.g., from single to double spacing, the CSD file would have to be regenerated. In future versions of the prototype, these options are expected to be handled by the new display routines and, as such, can be modified dynamically without regenerating the CSD file.

Vertical spacing options will include single, double, and triple spacing (default is single). Margins will be roughly controlled by the character line length selected, either 80 or 132 characters per line (default is 80). Indentation of the CSD constructs has been a constant three blank characters. Support for variable margins and indentation is being investigated in conjunction with the new display routines. In addition, several display options involving CSD graphical constructs are under consideration. For example, the boxes drawn around procedure and task entry calls may be optionally suppressed to make the diagram more compact.

Collapsing the CSD. The CSD window should provide the user with the capability to collapse the CSD based on all control constructs as well as complete diagram entities (e.g., procedures, functions, tasks and packages). This capability directly combines the ideas of chunking with control flow which are major aids to comprehension of software. An *architectural CSD* (ArchCSD) [DAV90] can be facilitated by collapsing the CSD based on procedure, function, and task entry calls, and the control constructs that directly affect these

calls. In future versions of the prototype, the ArchCSD will be generated by the display routines from the single intermediate representation of the CSD.

Color. Although general color options such as background and foreground may be selected via the X Windows system, color options within a specific diagram were only briefly investigated for both the screen and printer. It was decided that these will not be pursued in the Version 3 prototype.

Printing An Entire Set of CSDs. Printing an entire set of CSDs in an organized and efficient manner is an important capability when considering the typically large size of Ada software systems. A book format is under consideration which would include a table of contents and/or index. In the event GRASP/Ada is fully integrated with IDE/StP/ADE, the StP Document Preparation System may be utilized for this function.

Navigating among CSDs and Object Diagrams. A GRASP library is required to provide the overall organization of the generated diagrams. The current file organization uses standard UNIX directory conventions as well as default naming conventions. For example, all Ada source files end in ***.a*** or ***.ada***, the corresponding CSD files end in ***.a.csd***, and the corresponding print files end in ***.a.csd.ps***. In the present prototype, library complexity has been keep to a minimum by relying on the UNIX directory organization. In future versions, a GRASP library entry will be generated for each procedure, function, package, task, task entry, and label. The library entry will contain minimally the following fields.

identifier - note: unique key should be composed of the identifier + scoping.

scoping/visibility

type (procedure, function, etc.)

parameter list - to aid in overload resolution.

source file (file name, line number) - note: the page number can be computed from the line number.

CSD file (file name, line number)

OD file (file name)

"Referenced by" list

"References to" list

Alternatives for generation and updating of the library entries include the following.

(1) During CSD generation, the library entry is established and the entry is updated on subsequent CSD generations.

(2) During the processing via the ASIS of DIANA nets.

Alternatives for implementing the GRASP library include the following.

(1) Developing an Ada package or equivalent C module which is called by the CSD generation routines during the parse of the Ada source.

(2) Using the Rational Apex or Verdix VADS library system along with ASIS.

(3) Using the relational database system of a commercial CASE tool such as StP TROLL/USE.

Of these alternatives, the first one may be the most direct approach since it would be the easiest to control. The Apex, VADS, and StP library approaches may be more useful with the addition of object diagram generation and also with future integration of GRASP with commercial CASE tools.

6.0 Evaluation of the Control Structure Diagram and GRASP/Ada

An important aspect of any research project is the evaluation of the results. In the GRASP/Ada project the two primary results were (1) the development of the Control Structure Diagram (CSD) as a new algorithmic level graphical representation for Ada software and (2) the development of a prototype that automatically generates the CSD from existing Ada PDL or source code. Formal statistically-based controlled experiments dealing with the comprehensibility of graphical representations of software are difficult to design and conduct. Similar difficulties are encountered when attempting design controlled experiments to evaluate CASE tools with respect to improvements in productivity that result from their use. The primary difficulty arises from the learning curve that users/subjects must overcome. For example, a year or more may be required to become proficient enough with a software tool to actually realize gains in productivity. Thus, it may be difficult to compare two CASE tools in a "controlled" experiment without introducing bias based on familiarity or in many cases the lack of familiarity. As a result, most evaluation of CASE tools is based on preference surveys in which the user/subject is asked to make mental assessments or comparisons of various aspects of the tool(s) under study.

This section describes the design of the experiment, the subjects that participated in the evaluation of the CSD and GRASP/Ada, the preference survey instrument that was developed and administered, and the results of the analysis of the data collected.

6.1 The Design of the Experiment

The primary objective of the evaluation was to determine preference, if any, for the CSD over other similar graphical representations for algorithms. The ANSI flowchart (FC), the Nassi-Shneiderman diagram (NS), the Warnier-Orr diagram (WO), the action diagram (AD), were selected for comparison [MAR84]. The experiment was set up as a block design in which each subject was considered a block and each diagram type (FC, NS, WO, AD, CSD) represented a treatment. The subjects were to compare the treatments with respect to eleven performance characteristics (PCHs). A secondary objective of the evaluation was to collect constructive criticisms of the GRASP/Ada prototype to set priorities for future enhancements.

The evaluation instrument was divided into two parts: (1) the evaluation of graphical representations of algorithms and (2) the evaluation of GRASP/Ada (see Appendix C). In the first part, the first three items solicited background information with respect to familiarity with the five diagram types. The next eleven items indicated PCHs by which the subjects were asked to compare the diagrams with respect to (a) how well each represented sequence, selection, and iteration, (b) overall readability, (c) improvement in readability as an extension to pseudo-code, (d) ease of coding from, (e) ease of manual use, (f) overall preference if drawn manually, (g) overall economy, (h) overall preference with equivalent automated support, and finally (h) overall preference all assumptions aside. These eleven items are described in more detail blow in the discussion of results. The first part of the instrument concluded with an open ended question soliciting suggestions on how to improve any of the diagrams compared.

The second part of the evaluation instrument was directed specifically at the GRASP/Ada prototype. Questions were designed to solicit information regarding the User Interface, major problems encountered, modifications/enhancements desired, and the level of coverage provided for Ada during the presentation of GRASP/Ada.

6.2 The Subjects

The evaluation instrument was administered to 33 junior/senior computer science and engineering students at Auburn University in the course CSE 422 - Introduction to Software Engineering, during the Fall 1992 quarter. These students all had experience with FORTRAN, Pascal, and C in previous courses. None had formal training in Ada for which the GRASP/Ada tool was designed. Since participation in the evaluation was optional, five bonus points to be added to the final exam score were offered as an incentive. All students present took part in the evaluation.

Each of the graphical representations included in the first part of the evaluation instrument was presented briefly in class, and exercises were assigned involving the Nassi-Shneiderman diagram (NS) and the control structure diagram (CSD). Most students were familiar with the flowchart (FC) from prior classes.

The GRASP/Ada prototype was presented during a laboratory session and used in conjunction with the commercial CASE tool, *Software through Pictures* (StP). The primary focus of the CSE 422 lab is the development of a software specification using CASE tools such as StP.

6.3 The Evaluation Results

An item analysis was performed on the data collected for the eleven PCHs in the first part of the evaluation instrument (i.e., all items except the three background items at the beginning and the last item which asked for suggested improvements to the diagrams). Following the item analysis, an analysis of variance was performed to determine if differences in the preferences were statistically significance. The results item analysis and the tests for significance are presented below, followed by a general summary of the responses from the second part of the evaluation instrument.

6.3.1 Item Analysis of Comparison of Graphical Representations

The subjects were given the following instructions regarding the eleven performance characteristics.

> Based on the experience you have gained by using these diagramming tools to represent algorithms, you are asked to assign a rating to each of the diagrams with respect to a specific comparison among the diagrams. You may assign the same rating to more than one diagram for a given comparison. Select your ratings from the following scale and enter them as indicated below.

5 - best / most / first choice
4 -
3 - moderate
2 -
1 - worst / least / last choice

For <u>each</u> of the eleven items below, the subjects used the rating scale above to complete the following.

FC	NS	WO	AD	CSD
—	—	—	—	—

The eleven items describing performance characteristics were:

1. Compare the diagrams with respect to how well each shows **sequence**.

2. Compare the diagrams with respect to how well each shows **selection**.

3. Compare the diagrams with respect to how well each shows **iteration**.

4. Compare these diagrams with respect to overall **readability** (consider reading someone else's code).

5. Each of these tools can be used with informal pseudocode as opposed to actual statements in a programming language and, as such, can be thought of as a graphical extension to pseudocode (with possibly some spatial rearrangement). Rate the diagrams on the extent to which they increase readability over non-graphical pseudocode.

6. Suppose as a programmer you are given a design specification in which the program logic has been documented using one of the graphical representations below. Compare the diagrams with respect to which would best facilitate your task of coding from the design specification.

7. Compare the diagrams with respect to **ease of manual use**; consider the initial drawing and subsequent modifications.

8. Assuming you have to manually draw the diagrams (in the sense that they are <u>not</u> automatically generated), indicate your **overall preference** for each diagram where: 5 - first choice, . . ., 1 - last choice.

9. Compare the diagrams with respect to their **overall economy** (i.e., increases in comprehension versus effort to draw them manually).

10. Assuming you have equivalent **automated support** to draw each of the diagrams in the sense that the diagrams are automatically generated either by selecting constructs from a menu or by recognizing key words in the code,

indicate your overall preference for each diagram where: 5 - first choice, . . ., 1 - last choice.

11. All assumptions aside, indicate your **overall preference** for each diagram where: 5 - first choice, . . ., 1 - last choice.

The results of the item analysis for these eleven items are summarized in Tables 1 and 2 below. Table 1 shows the number of responses from the 33 students for each PCH and diagram type (FC, NS, WO, AD, CSD). A number less than 33 indicates the performance characteristic for that particular diagram type was left blank. Students were advised orally to leave an item blank if they were unfamiliar with the notation or a particular construct. Note that 20 students responded to all performance characteristics for all five diagrams, and 32 students responded to all performance characteristics for three of the diagrams (FC, NS, CSD). Table 2 contains the averages for the responses computed on the basis of only those items completed.

Table 1. Item Response Frequencies

	PCH	N:items	FC	NS	WO	AD	CSD
1.	SEQ	33	33	33	25	22	33
2.	SEL	33	33	33	24	21	33
3.	ITR	33	33	33	24	21	33
4.	GEN READ	33	33	32	24	20	33
5.	EXT P-COD	33	33	33	24	21	33
6.	CODE-FROM	33	33	32	25	22	32
7.	MANUAL	33	33	32	23	24	33
8.	PREF/MANL	33	33	33	26	25	33
9.	ECONOMY	33	33	33	25	24	33
10.	PREF/AUTO	33	33	33	25	23	33
11.	PREF/GEN	33	33	33	26	25	33

Table 2. Item Analysis for Graphical Representations

	PCH	N:items	FC	NS	WO	AD	CSD
1.	SEQ	33	3.21	3.64	2.64	2.32	3.94
2.	SEL	33	3.52	4.06	2.46	2.05	3.64
3.	ITR	33	3.45	3.48	2.58	2.14	3.91
4.	GEN READ	33	3.03	3.38	2.67	2.10	4.24
5.	EXT P-COD	33	2.85	3.76	2.38	2.48	3.94
6.	CODE-FROM	33	2.82	3.53	2.60	2.14	4.31
7.	MANUAL	33	3.09	3.16	2.61	2.38	3.91
8.	PREF/MANL	33	3.00	3.30	2.42	2.16	4.15
9.	ECONOMY	33	2.70	3.27	2.52	2.00	4.52
10.	PREF/AUTO	33	3.03	3.52	2.36	2.09	4.55
11.	PREF/GEN	33	3.00	3.33	2.54	1.96	4.55
ITEM AVG		363	3.06	3.49	2.52	2.16	4.15

The results in Table 2 clearly indicate the overall trend in preference for the CSD. For an additional perspective, the averages in Table 2 were converted into percentages and are shown in Table 3. Then the differences between the percentages in Table 3 for the CSD and the other diagram types is shown in Table 4.

Table 3. Percentage Scores For Graphical Representations

	PCH	N:items	FC	NS	WO	AD	CSD
1.	SEQ	33	64.24	72.73	52.80	46.36	78.79
2.	SEL	33	70.30	81.21	49.17	40.95	72.73
3.	ITR	33	69.09	69.70	51.67	42.86	78.18
4.	GEN READ	33	60.61	67.50	53.33	42.00	84.85
5.	EXT P-COD	33	56.97	75.15	47.50	49.52	78.79
6.	CODE-FROM	33	56.36	70.62	52.00	42.73	86.25
7.	MANUAL	33	61.82	63.12	52.17	47.50	78.18
8.	PREF/MANL	33	60.00	66.06	48.46	43.20	83.03
9.	ECONOMY	33	53.94	65.45	50.40	40.00	90.30
10.	PREF/AUTO	33	60.61	70.30	47.20	41.74	90.91
11.	PREF/GEN	33	60.00	66.67	50.77	39.20	90.91
ITEM AVG		363	61.27	69.89	50.48	43.23	82.98

Table 4. Percentage Score Differences - CSD Compared to Others

	PCH	N:items	FC - CSD	NS - CSD	WO - CSD	AD - CSD	CSD
1.	SEQ	33	-14.55	-6.06	-25.99	-32.42	
2.	SEL	33	-2.42	8.48	-23.56	-31.77	
3.	ITR	33	-9.09	-8.48	-26.52	-35.32	
4.	GEN READ	33	-24.24	-17.35	-31.52	-42.85	
5.	EXT P-COD	33	-21.82	-3.64	-31.29	-29.26	
6.	CODE-FROM	33	-29.89	-15.62	-34.25	-43.52	
7.	MANUAL	33	-16.36	-15.06	-26.01	-30.68	
8.	PREF/MANL	33	-23.03	-16.97	-34.57	-39.83	
9.	ECONOMY	33	-36.36	-24.85	-39.90	-50.30	
10.	PREF/AUTO	33	-30.30	-20.61	-43.71	-49.17	
11.	PREF/GEN	33	-30.91	-24.24	-40.14	-51.71	
ITEM AVG		363	-21.72	-13.09	-32.50	-39.76	

The Table 4 shows the difference between the control structure diagram (CSD) percentage scores and each of the other percentage scores. Negative values indicate a lack of preference for the indicated diagram type, and a preference for the CSD. Note that the NS SELection construct was the only item for which the CSD construct was not preferred on average. Item 5 is of particular interest in that it attempts to determine perceived improvements in readability over non-graphical pseudo-code.

Tables 1 through 4 provide useful insight and suggest potential significant differences among the preferences. However, an analysis of variance is required to determine the presence or absence of actual statistical significance.

6.3.2 Statistical Analysis for Significance of Preference of Graphical Representations

Since the data were taken on a scale of 1 to 5 and thus were not continuous, the ordinary Analysis of Variance (ANOVA) could not be applied. Note that ANOVA generally assumes that the data originates from a normal population - an assumption that would not be tenable in this case. Therefore, a nonparametric (or distribution-free) test was used to determine if there were statistically significant differences among the five treatments FC, NS, WO, AD, and CSD. The determination of significant differences would be made with respect to all eleven PCHs.

Each student responded to at least 3 of 5 treatments (i.e., each student ranked at least 3 of the treatments FC, NS, WO, AD, and CSD). Therefore, each student is considered as

a block and the relevant model is the randomized complete block design. The appropriate ANOVA nonparametric test is the Friedman's Test [CON80]. This test analyzes ranked data for a complete randomized block design. Since only 20 students responded to all 5 treatments, initially the test for significance was done for significant differences among all five treatments. The null hypothesis is as follows:

H_0 : There are no significant differences among the 5 treatments;

versus the alternative:

H_1 : There are significant differences.

For the sake of illustration, the data for the PCH SEQ is provided in Table 5 below.

Table 5. Data for Performance Characteristic SEQ (PCH #1)

		Student Responses					Assigned Ranks				
Record#	STU_NO	FC	NS	WO	AD	CSD	FC_R	NS_R	WO_R	AD_R	CSD_R
1	2	1	2	5	4	3	1.0	2.0	5.0	4.0	3.0
12	3	5	1	2	4	3	5.0	1.0	2.0	4.0	3.0
23	7	5	4	4	2	3	5.0	3.5	3.5	1.0	2.0
34	8	5	3	2	1	4	5.0	3.0	2.0	1.0	4.0
45	11	5	3	3	3	5	4.5	2.0	2.0	2.0	4.5
56	14	3	3	1	2	4	3.5	3.5	1.0	2.0	5.0
67	15	5	4	3	3	3	5.0	4.0	2.0	2.0	2.0
78	16	4	2	3	1	5	4.0	2.0	3.0	1.0	5.0
89	17	3	4	2	2	3	3.5	5.0	1.5	1.5	3.5
100	19	1	4	2	2	5	1.0	4.0	2.5	2.5	5.0
111	20	1	5	4	2	3	1.0	5.0	4.0	2.0	3.0
122	21	3	3	3	3	3	3.0	3.0	3.0	3.0	3.0
133	23	4	2	3	3	5	4.0	1.0	2.5	2.5	5.0
144	25	1	5	3	2	4	1.0	5.0	3.0	2.0	4.0
155	27	3	4	1	2	5	3.0	4.0	1.0	2.0	5.0
166	29	4	5	1	2	3	4.0	5.0	1.0	2.0	3.0
177	30	1	2	3	4	5	1.0	2.0	3.0	4.0	5.0
188	31	5	4	2	1	3	5.0	4.0	2.0	1.0	3.0
199	32	2	5	3	1	4	2.0	5.0	3.0	1.0	4.0
210	33	2	5	1	3	4	2.0	5.0	1.0	3.0	4.0
					Totals		63.5	69.0	48.0	43.5	76.0

Table 5 shows that, for example, student number 23 ranked FC, NS, WO, AD, CSD as 4, 2, 3, 3, and 5, respectively. The Friedman's Test requires that the responses for all treatments be ranked within each block from 1 to 5 and therefore, that the sum of the ranks within each block is 15 as shown below.

$$\sum_{j=1}^{5} R_{ij} = \sum_{j=1}^{5} j = 15$$

The tied ranks receive average ranks. The last five columns (FC_R, NS_R, WO_R, AD_R, and CSD_R) of Table 9 give the ranks for each of the 20 students for the performance characteristic SEQ after tied ranks have been averaged. Now the ranks assigned by student number 23 become 4.0, 1.0, 2.5, 2.5, and 5.0 respectively. Since the student ranked NS the lowest, the treatment NS_R received a rank of 1.0. The student ranked WO and AD equally

and next lowest, therefore ranks 2 and 3 were averaged and WO_R and AD_R each received 2.5. Similarly, FC_R received a rank of 4.0 and CSD a rank of 5.0. Thus, we have

$$\sum_{j=1}^{5} R_{13j}=4.0+1.0+2.5+2.5+5=15$$

as expected for student number 23 in row 13. The 2-way (treatments and blocks) ANOVA is conducted on ranks as illustrated below.

Let SS (Total), SS (Treatments), SS (Blocks), and SS (Residuals) represent total sum of squares, treatment sum of squares, block sum of squares, and residual sum of squares, respectively. Then

$$SS(Total)=\sum_{i=1}^{20}\sum_{j=1}^{5} R_{ij}^2 - \frac{300^2}{100}=1^2+2^2+5^2+\ldots+4^2-900=182.50$$

where the correction factor is

$$CF=\frac{(\sum_{i=1}^{20}\sum_{j=1}^{5} R_{ij})^2}{100}=\frac{[\sum_{i=1}^{20}(15)]^2}{100}=\frac{300^2}{100}$$

$$SS(Treatments)=\sum_{j=1}^{5}(R_{ij}^2/20) - 900=\frac{63.5^2+69.0^2+48.0^2+43.5^2+76.0^2}{20}-900=38.275$$

$$SS(Blocks)=\sum_{i=1}^{20}\frac{(R_{ij})^2}{5} - \frac{300^2}{100}=\frac{15^2+15^2+\ldots+15^2}{5}-900=\frac{20(225)}{5}-900=0$$

Note that, since each block subtotal is

$$SS(Blocks)=SS(Students)$$

is identically zero. Hence, we have the following $R_i=\sum_{j=1}^{5} R_{ij}=15$, then

$$SS(Residual)=SS(Total)-SS(Treatments)-SS(Blocks)=144.225$$

Table 6. Results of ANOVA

Source	df Degrees of Freedom	SS Sum of Squares	MS Mean Squares = SS/df	F_0 $(SS_{tr}/4)$ / $(SS_{res}/76)$
Total	99 =#Tr (#Stu) - 1	182.50		
Treatments	4 = #Tr - 1	38.275	9.56875	5.0423
Blocks	19 = #Stu - 1	0.0	0.0	
Residuals	76 =Totl-Tr-Blk = 99 - 4 - 19	144.225	1.8977	

The results of the ANOVA are shown in Table 6. Having computed the F statistic, we must now determine if it is sufficiently large to reject the null hypothesis. Since the 1 percentage point of the F distribution with 4 and 76 degrees of freedom (*df*) is $F_{.01}(4, 76) = 3.577$ which is less than $F_0 = 5.0423$, we reject the null hypothesis that there were no significant differences among the FC, NS, WO, AD, and CSD with respect to the performance characteristic "Sequence." Now that the null hypothesis has been rejected, we have either made a correct decision or we have committed a Type I error (note, a Type I error is committed when an experimenter rejects a true hypothesis). The probability of committing a Type I error, or the level of significance of the test is determined from $\hat{\alpha} = P(F_{4,76} \geq F_0)$. The probability (or critical) level of the test in this case is $\hat{\alpha} = 0.00117$ which indicates there is very little probability of a Type I error. Note that the smaller the critical level is, the more strongly H_0 can be rejected and the more significant are the differences among the treatments.

The Friedman's Test was conducted on the complete data set of 20 students for all five treatments and the results are summarized in Table 7. The Friedman's statistic, F_0, showed that, except in the case of performance characteristic MANUAL, the differences among the 5 treatments were highly significant ($\hat{\alpha} << 0.01$). In the case of MANUAL, the differences among the five treatments were significant at the level $\hat{\alpha} = 0.0133$.

Table 2 clearly shows that the average rating that the treatments WO and AD received were lower than the other three, FC, NS, and CSD. Furthermore, 32 students rated FC, NS, and CSD as opposed to only 20 who rated all five treatments. As a result, the Friedman's Test was also applied to determine if the three treatments FC, NS, CSD differed significantly. Again each student's rating of FC, NS, and CSD were ranked as 1, 2, 3 (average ranks were assigned to equal ratings as before) and the Friedman's Test was applied for each of the eleven PCHs. The results are summarized in Table 8.

Table 7. ANOVA Summary for Treatments FC, NS, WO, AD, CSD

Totals for Assigned Ranks of 20 Students for 5 Treatments

	PCH	FC_R	NS_R	WO_R	AD_R	CSD_R
1.	SEQ	63.5	69.0	48.0	43.5	76.0
2.	SEL	70.5	85.5	39.0	38.0	67.0
3.	ITR	74.0	68.0	50.5	40.5	67.0
4.	GEN READ	63.5	67.0	47.0	39.5	83.0
5.	EXT P-COD	57.0	79.5	41.5	46.5	75.5
6.	CODE-FROM	57.0	70.5	45.5	43.0	84.0
7.	MANUAL	65.0	68.0	48.5	46.0	72.5
8.	PREF/MANL	63.5	69.5	42.5	45.0	79.5
9.	ECONOMY	55.0	71.5	49.0	38.5	86.0
10.	PREF/AUTO	55.0	70.0	42.0	40.5	92.5
11.	PREF/GEN	57.0	70.0	46.0	38.0	89.0

Statistics for Treatments FC, NS, WO, AD, and CSD $\hat{\alpha} = P(F_{4,76} \geq F_0)$

	PCH	SS_TOTL	SS_TR	MS_TR	SS_RESID	MS_RESID	F_0	$\hat{\alpha}$
1.	SEQ	182.50	38.2750	9.5688	144.2250	1.8977	5.0423	.00117
2.	SEL	188.00	86.7250	21.6813	101.2750	1.3326	16.2703	$.0^{8}115$
3.	ITR	182.50	38.9750	9.7438	143.5250	1.8885	5.1596	.00099
4.	GEN READ	191.50	58.9750	14.7438	132.5250	1.7437	8.4552	.000011
5.	EXT P-COD	185.00	57.7000	14.4250	127.3000	1.6750	8.6119	.00000871
6.	CODE-FROM	194.50	59.7250	14.9313	134.7750	1.7734	8.4198	.0000112
7.	MANUAL	189.50	28.6750	7.1688	160.8250	2.1161	3.3877	.013231
8.	PREF/MANL	196.50	50.7000	12.6750	145.8000	1.9184	6.6070	.000129
9.	ECONOMY	187.00	70.8250	17.7063	116.1750	1.5286	11.5832	.000000215
10.	PREF/AUTO	194.00	94.2750	23.5688	99.7250	1.3122	17.9616	$.0^{9}202$
11.	PREF/GEN	197.00	81.5000	20.3750	115.5000	1.5197	13.4069	$.0^{7}258$

Table 8. ANOVA Summary for Treatments FC, NS, CSD

Totals for Assigned Ranks of 32 Students for 3 Treatments

	PCH	FC_R	NS_R	CSD_R
1.	SEQ	57.0	66.0	69.0
2.	SEL	60.0	73.5	58.5
3.	ITR	64.0	58.0	70.0
4.	GEN READ	54.0	57.0	81.0
5.	EXT P-COD	51.0	68.5	72.5
6.	CODE-FROM	46.5	63.0	82.5
7.	MANUAL	60.0	58.0	74.0
8.	PREF/MANL	53.5	61.0	77.5
9.	ECONOMY	46.0	59.0	87.0
10.	PREF/AUTO	48.5	57.0	86.5
11.	PREF/GEN	50.0	58.0	84.0

Statistics for Treatments - FC, NS, and CSD $\hat{\alpha} = P(F_{2,62} \geq F_0)$

	PCH	SS_TOTL	SS_TR	MS_TR	SS_RESID	MS_RESID	F_0	$\hat{\alpha}$
1.	SEQ	59.00	2.4375	1.2188	56.5625	0.9123	1.3359	.270
2.	SEL	59.00	4.2656	2.1328	54.7344	0.8828	2.4159	.098
3.	ITR	59.50	2.2500	1.1250	57.2500	0.9234	1.2183	.303
4.	GEN READ	61.00	13.6875	6.8438	47.3125	0.7631	8.9683	.00004
5.	EXT P-COD	54.00	8.1719	4.0859	45.8281	0.7392	5.5278	.0062
6.	CODE-FROM	61.50	20.2969	10.1484	41.2031	0.6646	15.2708	.000004
7.	MANUAL	58.00	4.7500	2.3750	53.2500	0.8589	2.7653	.071
8.	PREF/MANL	62.00	9.4219	4.7109	52.5781	0.8480	5.5551	.006
9.	ECONOMY	60.50	27.4375	13.7188	33.0625	0.5333	25.7259	.000000007
10.	PREF/AUTO	61.50	24.8594	12.4297	36.6406	0.5910	21.0324	.0000001
11.	PREF/GEN	64.00	19.7500	9.8750	44.2500	0.7137	13.8362	.000011

Table 8 shows that the differences among the three treatments FC, NS, CSD were not significant (at the .05 level) for the four performance characteristics SEQ, SEL, ITR, and MANUAL, but the three treatments differed very significantly with respect to each of the other seven performance characteristics. Furthermore, the average ranks for CSD far exceeded those for FC and NS in the case of the seven significantly different performance characteristics.

6.3.3 Summary of Responses For Evaluation of GRASP/Ada

The second part of the evaluation instrument was specifically directed at GRASP/Ada. The items are presented below with a summary of the responses in *italics*.

1. Was the User Interface intuitive?

Most subjects felt comfortable with the User Interface after several sessions. However, many expressed the desire for a User Manual.

2. What changes would you make to the User Interface?

Most subjects stated the User Interface was acceptable as is. Several expressed a desire to have "stickable" subwindows from which options are selected. These were not available through Athena widgets from which the User Interface was constructed.

3. What were the major problems you encountered when using GRASP/Ada.

As one might expect, a variety of responses were given for this item. Most were as a result of several known bugs which have since been removed. Some simply indicated improper use of the prototype and/or a lack of expertise in Ada. Again, many expressed the desire for a User Manual.

4. Rank the following items in order of importance in the prototype. Note, some of these items are available in the current version and others are under consideration as modifications/enhancements. Also, feel free to comment on each in the space provided. (1 - least important, ..., 7 - most important)

   The overall rank of the items is indicated.

   a. **4.69** Integration of CSD generation/editing capabilities with a CASE tool such as StP to facilitate development of process pspecs and/or module PDL.

   b. **4.84** GRASP/Ada User's Manual.

   c. **4.84** Error messages resulting from CSD generation.

d. **4.47** Integration of CSD editing/generation with automatic generation of object diagrams to show software architectural design (i.e., the object diagrams indicate the dependencies among a set of CSDs).

e. **2.81** Spatial options (line spacing, amount of indentation, etc.).

f. **4.22** Direct access to a compiler from the User Interface to facilitate use of the CSD during implementation.

g. **5.19** Extension of the CSD editor and generator to handle other languages such as C and Pascal.

5. Rate your knowledge of Ada.

_____ excellent _____ good _____ moderate _____ very little _____ virtually none

1.76 *indicates knowledge of Ada was between virtually none and very little.*

6. How useful was the Ada template feature in the CSD Window in producing Ada/PDL CSDs?

_____ extremely _____ very _____ moderately _____ not very _____ not useful
3.53 *indicates usefulness of the Ada template was between moderately and very useful.*

What modifications/improvements should be made to this feature?

Many subjects indicated that additional Ada construct templates were needed. Only control structures are included presently.

7. The time in class spent on Ada and/or AdaPDL

_____ should have been increased. _____ was about right. _____ should have been decreased.

2.64 *indicates the class time spent on Ada was between about right and should have been increased.*

Comments? *Some subjects indicated that the course (CSE 422) should have a more formal emphasis on Ada. Other indicated an emphasis on Pascal or C was preferred since prior required courses cover these languages.*

8. CSD editors and generators are planned for C and Pascal. If these tools were available on the network, how useful would they be to you with respect to improving the readability of your source code in future software development projects?

C:

_____ extremely _____ very _____ moderately _____ not very _____ not useful

4.09 *indicates a CSD editor/generator for C would be between very and extremely useful.*

Pascal:

_____ extremely _____ very _____ moderately _____ not very _____ not useful

3.36 *indicates a CSD editor/generator for Pascal would be between moderately and very useful.*

6.4 Future Directions for Evaluation

The current preference survey instrument should be refined with respect to the performance characteristics. The diagram types (FC, NS, WO, AD, CSD) that were compared to the CSD should be reassessed as to whether they are currently used widely in practice. Non-graphical PDL should also be considered as a treatment in the next evaluation of preference.

While the analysis of preference data in this research clearly indicated statistically significant differences which heavily favored the CSD, a controlled experiment should be done to evaluate actual increases and/or decreases in comprehension due to the use of a particular graphical notation or PDL. In fact, since PDL is in such widespread use as a detailed design language, an experiment comparing the comprehensibility of PDL versus the CSD would be perhaps even more appropriate than attempting to compare numerous graphical notations.

7.0 Conclusions

The GRASP/Ada project has provided a strong foundation for the automatic generation of graphical representations from existing Ada software. The current prototype provides the capability for the user to generate the Control Structure Diagram (CSD) from syntactically correct Ada PDL or source code in a reverse engineering mode with a level of flexibility suitable for practical application. The prototype is being used in two software engineering courses at Auburn University on student projects in conjunction with other CASE tools. The feedback provided by the students has been very useful, especially with respect to the user interface. The prototype has been prepared for limited distribution (GRASP/Ada V3.4).

An important issue for all software tools in general, and graphical representations in particular, is evaluation. An evaluation based on preference was conducted to provide information on user perceptions of the CSD. An experiment was designed and data was collected from software engineering students. Statistical analysis indicated highly significant differences among five graphical notations when compared with respect to eleven performance characteristics. There was a clear preference for the CSD for seven of the eleven performance characteristics. Experience indicates that empirical evaluation of the comprehensibility (rather than preference) of graphical notations such as data flow diagrams, object diagrams, structure charts, and flowgraphs is difficult. However, such an evaluation for the CSD and GRASP/Ada tool would provide further insight into the role that graphical notations play in the comprehension of software and, as a result, their potential impact on the overall cost of software.

The CSD generation component of GRASP/Ada has been loosely integrated with IDE's Software though Pictures to replace non-graphical process specifications (pspecs) for data flow diagrams and module PDL for structure charts and object diagrams (see Appendix B). In fact, the CSD becomes a natural detailed-level graphical extension for these system and architectural level diagrams. In this capacity, the CSD has the potential to replace traditional non-graphical pspecs/PDL used in software design and textual source code listings used in implementation, testing, and maintenance.

The primary impact of reverse engineering graphical representations will be improved comprehension of software in the form of visual verification and validation (V&V). To move the results of this research in the direction of visualizations to facilitate the processes of V&V, numerous additional capabilities must be explored and developed. A set of graphical representations that directly support V&V of software at the architectural and system levels of abstraction must be formulated. For example, the Object Diagram generator (ODgen) prototype described earlier is one the components of the GRASP/Ada project which would address architectural and system levels of abstraction. This task must include an on-going investigation of visualizations reported in the literature as currently in use or in the experimental stages of research and development. In particular, specific applications of visualizations to support V&V procedures must be investigated and classified. Prototype software tools which generate visualizations at various levels of abstraction from source code and PDL, as well as other intermediate representations, must be designed and implemented. Graphically-oriented editors must provide capabilities for dynamic reconstruction of the diagrams as changes are made to other diagrams at various levels. These graphical representations should provide immediate visual feedback to the user in an incremental

fashion as individual structural and control constructs are completed. Future directions and specific tasks for the GRASP/Ada project have been been described at the end of each of the sections above.

The current prototype of the CSD generator, while only one of set of required visualization tools, has clearly indicated the utility of the CSD. Future enhancements will only increase its effectiveness as a tool for improving the comprehensibility of software.

REFERENCES

ADA83 *The Programming Language Ada Reference Manual.* ANSI/MIL-STD-1815A-1983. (Approved 17 February 1983). In *Lecture Notes in Computer Science*, Vol. 155. (G. Goos and J. Hartmanis, eds) Berlin : Springer-Verlag.

AOY89 M. Aoyama, et.al., "Design Specification in Japan: Tree-Structured Charts," *IEEE Software*, Mar. 1989, 31-37.

BAR84 J. G. P. Barnes, *Programming in Ada*, Second Edition, Addison-Wesley Publishing Co., Menlo Park, CA, 1984.

CHI90 E. J. Chikofsky and J. H. Cross, "Reverse Engineering and Design Recovery - A Taxonomy," *IEEE Software*, Jan. 1990, 13-17.

CON80 W. J. Conover, Practical Nonparametric Statistics, Joh Wiley and Sons, New York, 1980.

CRO88 J. H. Cross and S. V. Sheppard, "The Control Structure Diagram: An Automated Graphical Representation For Software," *Proceedings of the 21st Hawaii International Conference on Systems Sciences* (Kailui-Kona, HA, Jan. 5-8). IEEE Computer Society Press, Washington, D. C., 1988, Vol. 2, pp. 446-454.

CRO90a J. H. Cross, K. I. Morrison, C. H. May, "Generation of Graphical Representations From Source Code," *Proceedings of the Southeast Regional ACM Computer Science Conference*, April 18-20, 1990, Greenville, South Carolina, 54-62.

CRO90b J. H. Cross, S. V. Sheppard and W. H. Carlisle, "Control Structure Diagrams for Ada," *Journal of Pascal, Ada, and Modula 2*, Vol. 9, No. 5, Sep./Oct. 1990.

CRO92 J. H. Cross, E. J. Chikofsky and C. H. May, "Reverse Engineering," *Advances in Computers*, Vol. 35, 1992, 199-254.

CAL93 F. W. Calliss, J. H. Cross, V. Rajlich, Panel on "Reverse Engineering," *Proceedings of 5th International Conference on Software Engineering and Knowledge Engineering* (San Francisco, CA June 16-18, 1993), 544-545.

DAV90 R. A. Davis, "A Reverse Engineering Architectural Level Control Structure Diagram," *M.S. Thesis*, Auburn University, December 14, 1990.

MAR85 J. Martin and C. McClure, *Diagramming Techniques for Analysts and Programmers.* Englewood Cliffs, NJ : Prentice-Hall, 1985.

OLS93 M. R. Olsem and C. Sittenauer, "Reengineeing Technology Report, Volume 1," Software Technology Support Center, Hill Air Force Base, UT 840556, August 1993.

SCA89 D. A. Scanlan, "Structured Flowcharts Outperform Pseudocode: An Experimental Comparison," IEEE Software, Sep. 1989, 28-36.

SEL85 R. Selby, et. al., "A Comparison of Software Verification Techniques," NASA *Software Engineering Laboratory Series* (SEL-85-001), Goddard Space Flight Center, Greenbelt, Maryland, 1985.

SHN77 B. Shneiderman, et. al., "Experimental Investigations of the Utility of Detailed Flowcharts in Programming," *Communications of the ACM*, No. 20 (1977), pp. 373-381.

SHU88 Nan C. Shu, *Visual Programming*, New York, NY, Van Norstrand Reinhold Company, Inc., 1988.

SIT92 C. Sittenauer and M. R. Olsem, "Re-engineeing Tools Report," Software Technology Support Center, Hill Air Force Base, UT 840556, July 1992.

STA85 T. Standish, "An Essay on Software Reuse," *IEEE Transactions on Software Engineering*, SE-10 (9), 494-497, 1985.

TRI89 L. L. Tripp, "A Survey of Graphical Notations for Program Design -An Update," *ACM Software Engineering Notes*, Vol. 13, No. 4, 1989, 39-44.

WAS89 A. I. Wasserman, P. A. Pircher and R. J. Muller, "An Object Oriented Structured Design Method for Code Generation," ACM SIGSOFT *Software Engineering Notes*, Vol. 14, No. 1, January 1989, 32-52.

APPENDICES

A. Installation Guide

B. Integrating GRASP/Ada with *Software through Pictures* (StP)

C. Evaluation Instrument

Appendix A

GRASP/Ada

Installation Guide

GRASP/Ada 3.4 Installation Notes
$(GRASP_HOME)/sup/doc/README.install.3.4
First compiled: 10/31/91 by CHM2
Additions:
12/10/91 by CHM2
01/17/92 by CHM2
01/27/92 by CHM2
12/12/93 by JHC2

##

Purpose of this file:

To give the installer the proper directions for installing GRASP/Ada and for instructing users in adjusting their environments as necessary.

##

Installation Requirements

1. GRASP/Ada was developed on Sun SPARCstations running SunOS (a derivative of UNIX BSD ?) and was intended to run on such systems. At the time of this writing, it is UNKNOWN whether GRASP/Ada will run under any other UNIX workstation environment.

2. The file hierarchy, when unarchived, consumes approximately 1.5 MB of disk space (not including the tar file itself).

   The process of "making" the program and associated object files should require approximately 3 MB more disk space.

   All intermediate products of the "make" process will be removed almost as soon as their usefulness has expired. They will not linger on until the entire "make" process is complete; this will reduce disk space requirements.

   The final GRASP/Ada system (source, executable, other needed files) will require about 1.5 MB of disk space.

3. As stands to reason, the installer must have read, write, and execute (search) permission in the directory in which he installs GRASP/Ada.

4. X-related requirements:

   a. The system on which GRASP/Ada runs must have X11R4 or OpenWindows.

   b. The system must have the following link libraries:

      -- Athena Widget Set (Xaw)
      -- (Xmu)
      -- (Xt)
      -- (X11)

##

Installation Directions

---

1. The installer should have an archive file named "grasp.3.4.tar" or possibly "grasp.3.4.tar.Z" (the compressed version).

   He should move this archive file to the directory in which he intends to install GRASP/Ada (called, logically enough, the "installation directory"). A typical example directory would be "/usr/local"; thus, the archive file will be in the "/usr/local" directory.

2. The installer then should unarchive the archive file in this way:

   (if compressed version)

   ```
   $ uncompress grasp.3.4.tar.Z
   $ tar xvBhmof grasp.3.4.tar
   ```

   (if uncompressed version)

   ```
   $ tar xvBhmof grasp.3.4.tar
   ```

   This will unarchive an hierarchy of files whose root is a directory called "graspada". "graspada" will be a subdirectory of the installation directory (i.e. /usr/local in the above example--ergo, /usr/local/graspada).

   The installer will be the owner of all the files and directories in the hierarchy. The permissions mode of the files and directories will be set according to the installer's umask code (see umask(1)).

3. At this point, the installer needs to define an environment variable known as "GRASP_HOME". This variable is necessary to the installation and use of GRASP/Ada. In his .cshrc file, the installer should have the following line:

   ```
   setenv GRASP_HOME [name of installation directory]/graspada
   ```

4. Check LD_LIBRARY_PATH environment variable to ensure it contains the path for the X Athena Widgets libraries (i.e., Xaw, Xmu). The default is /usr/lib. There should be a link directory in /usr/lib to the X libraries if they are not in the default library.

5. (OPTIONAL) There is an applications default file in $GRASP_HOME/lib called "XGrasp" which determines things like background color, button font, etc. To activate these defaults set the XENVIRONMENT variable as follows:

```
setenv XENVIRONMENT $GRASP_HOME/lib/XGrasp
```

6. At this point, the installer should "source" the .cshrc file, initializing the environment variables set above:

```
$ source ~/.cshrc
```

7. To ensure that others will not tamper with various source and other files, the installer should change the modes of the files and directories in this way:

```
$ chmod -R 755 $GRASP_HOME
```

or, if installer is especially trusting of those in his group,

```
$ chmod -R 775 $GRASP_HOME.
```

8. The installer can now proceed to install GRASP/Ada. The installation is done via a hierarchy of makefiles, so there will be a lot of fast-moving messages. The installer need not be disconcerted by this.

Make and install GRASP/Ada in this way:

(Note: currently only configured for OpenWindows)

```
$ cd $GRASP_HOME
$ make -e all "GRASP_VERSION=3.4" "WINDOWING_SYSTEM=1"
```

The following actions are performed in the "make" process:

- the program executable is constructed,
- the man page(s) is (are) constructed,
- the appropriate screen CSD character fonts are constructed, and
- the printer definitions file is constructed.

The whole process should take about 5-10 minutes.

##

User Instructions

Once installation is complete (no error messages), the installer should inform potential users of the following environmental changes:

The following should appear in the .cshrc files of all GRASP users in this order:

1. An environment variable "GRASP_HOME" which is where GRASP (the current version) is installed, namely,

```
      setenv GRASP_HOME [name of installation directory]/graspada

  2.  The path wherein is located the GRASP executable, namely

            $GRASP_HOME/bin

      in their $PATH list.

  3.  The path containing the GRASP/Ada man page(s):

            $GRASP_HOME/man

      in their $MANPATH list.

  4.  If users wish to override the X11R4 application defaults for
      GRASP/Ada (or whatever defaults that might be in effect),
      then they should set the environment variable

            XENVIRONMENT

      to their own custom application defaults file.

Invocation is by the command "graspada" or "graspada <filename>".


##############################################################################
```

Appendix B

GRASP/Ada

Integrating GRASP/Ada with *Software through Pictures* (StP)

Integrating GRASP/Ada with *Software through Pictures* (StP)

Introduction

Software through Pictures (StP) is a commercially available CASE tool from Interactive Development Environments, Inc. (IDE). StP provides automated support for software development methods which allow the user to build a comprehensive model of the system. This system model helps ensures the integrity of the design before starting the production of the code. In particular, StP can be used to build the graphical representations such as data flow diagrams, structure charts, entity relationship diagrams, and object diagrams. The information generated from the design is compiled into the underlying database called Data Dictionary to ensure consistency across the entire project. Once the design is complete, the system can be developed according to the model. StP includes a set of editors for graphically modeling programs and the data used in the programs. Among these graphical editors are the Data Flow Diagram Editor (DFE) and Structure Chart Editor (SCE). GRASP/Ada can be used to generate a control structure diagram (CSD) from (or in place of) process specifications (pspecs) in the DFE and module specifications (PDL) in the SCE. Using CSDs for pspecs and PDL provides a natural graphical extension to data flow diagrams and structure charts. The figure below shows a snapshot of the screen with a pspec represented by a CSD.

Data Flow Diagram Editor

The Data Flow Diagram Editor (DFE) is an interactive graphical tool for drawing data flow diagrams in support of the Structured Analysis method. Data flow diagrams provide a view of the system from a functional perspective. The top level context diagram shows the system's overall purpose and how it interacts with external objects. Lower level diagrams show the system subdivided into components or processes using decomposition techniques. The DFE models the data flow of a system by using symbols that represent processes, data flows, data stores, and external data sources and sinks. The status of a process is represented by the presence or absence of a status marker next to the index number. Ifs is undefined. An undefined process is one that is neither decomposed nor has a pspec. All processes must be defined before the diagram is entered into the Data Dictionary. If there is an asterisk (*) next to the process index number, the process is decomposed. If a Pspec has been generated for the process, a small p appears next to the process index.

GRASP/Ada can be used to generate CSDs from syntactically correct Ada programs or Ada PDL. Since the Pspec editor of the DFE does not have the graphic capability, it can be replaced by GRASP/Ada.

Structure Chart Editor

The Structure Chart Editor (SCE) is an interactive graphical editor for drawing structure charts. The purpose of the structure chart is to show the interconnections between identifiable program modules by graphically representing the modules hierarchy and indicating the data that is passed between the modules. Each module has an associated non-graphical PDL module specification. As with the pspec above, GRASP/Ada can be used to generate a CSD for each PDL spec by replacing the StP PDL editor with GRASP/Ada.

Integration Procedure

Before integrating GRASP/Ada with StP, the environment variables and path variables required to execute GRASP/Ada as a stand alone tool should be set in .cshrc file. Tool Information files are the principal means by which the user customizes StP to suit the needs of individual user and environment.

Variables in the Tool Information file are used to customize the environment in which StP runs, specify the commands StP executes when running, and create the look and feel of the graphical editors. The Tool Information file is a ASCII text file, and the variables can be changed by commenting them out, changing their values, or setting their values in another file.

When StP is first invoked from the command line, the system uses routines in the Tools Library to find the appropriate Tool Information file to read. First, the system looks for an environment variable called **ToolInfo** defined in the user's .cshrc file. This variable gives the complete pathname to the Tool Information file to be read. A ToolInfo variable can have a value which is a number or a character string, the pathname for a file or directory, or a command with optional parameters. For example, the variable that specifies the location of the file used to set up the Main Menu may appear in the file as follows:

STPMenuSpec=/usr/local/lib/STPmenu.spec

The appearance of the Main Menu and the tools and commands available through it are controlled by certain ToolInfo variables and Main Menu Specification file.

The Main Menu Specification File. The main menu specification file determines choices available in various areas of the Main Menu Window. It specifies the icons and labels that can appear in these areas, and it determines what combinations of commands or list of choices are available and how they are displayed.

Structure of the Specification File. The following excerpt from STPmenu.spec gives the specification for CSD icon.

```
/GRASP
      label [GRASP]
      image   { \
/* Format_version=1, Width=64, Height=64, Depth=1, Valid_bits_per_item=16   \
 */   \
    0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF, \
    0xC000,0x0000,0x0000,0x0003,0xC000,0x0000,0x0000,0x0003, \
    0xC000,0x0FFF,0xFFFF,0xFC03,0xC000,0x0FFF,0xFFFF,0xFC03, \
    0xC000,0x0C00,0x0000,0x0003,0xC000,0x0C00,0x0000,0x0003, \
    0xC000,0x0C00,0x0000,0x0003,0xC000,0x0FFF,0xFFFF,0xFC03, \
    0xC000,0x0FFF,0xFFFF,0xFC03,0xC000,0x0001,0x8000,0x0003, \
    0xC000,0x0001,0x8000,0x0003,0xC000,0x0001,0x8000,0x0003, \
    0xC000,0x0001,0x8000,0x0003,0xC3E0,0x0001,0x8000,0x0003, \
    0xC7F0,0x0001,0x8060,0x0003,0xCC30,0x0001,0x80F0,0x0003, \
    0xCC01,0xE001,0xFF80,0x0003,0xCC03,0xF001,0xFF98,0x0003, \
    0xCC07,0x3801,0x8198,0x0003,0xCC07,0x1BE1,0x819F,0xE003, \
    0xCC03,0x83F1,0x819F,0xE003,0xCC01,0xC339,0x8198,0x0003, \
    0xCC30,0xE319,0x8198,0x0003,0xC7F0,0x7319,0x819F,0xE003, \
    0xC3E6,0x3B19,0x819F,0xE003,0xC007,0x3B19,0x8198,0x0003, \
    0xC003,0xF319,0x8198,0x0003,0xC001,0xE319,0x819F,0xE003, \
    0xC000,0x0339,0x819F,0xE003,0xC000,0x03F1,0x8198,0x0003, \
    0xC000,0x03E1,0x80F0,0x0003,0xC000,0x0001,0x8060,0x0003, \
    0xC000,0x0001,0x8000,0x0003,0xC000,0x0001,0x8000,0x0003, \
    0xC000,0x0001,0x8000,0x0003,0xC000,0x0001,0x8000,0x0003, \
    0xC000,0x0001,0x8000,0x0003,0xC000,0x0001,0x8000,0x0003, \
    0xC000,0x0001,0x8000,0x0003,0xC1F8,0x0E01,0xFFE0,0x0003, \
    0xC3FC,0x1E01,0xFFE0,0x0003,0xC606,0x1601,0x8000,0x0003, \
    0xC606,0x0601,0x8000,0x0003,0xC006,0x0601,0x8000,0x0003, \
    0xC006,0x0601,0x8000,0x0003,0xC006,0x0601,0x8000,0x0003, \
    0xC006,0x0601,0x8000,0x0003,0xC07C,0x0601,0x8000,0x0003, \
    0xC07C,0x0601,0x8000,0x0003,0xC006,0x0601,0x8000,0x0003, \
    0xC006,0x0601,0x8000,0x0003,0xC006,0x0601,0x8000,0x0003, \
    0xC006,0xC601,0x8000,0x0003,0xC606,0xC601,0x8000,0x0003, \
    0xC606,0x0601,0x8000,0x0003,0xC3FC,0x1F81,0x8000,0x0003, \
    0xC1F8,0x1F81,0x8000,0x0003,0xC000,0x0000,0x0000,0x0003, \
    0xC000,0x0000,0x0000,0x0003,0xC000,0x0000,0x0000,0x0003, \
    0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF  \
}

      help  Graphical Representation of Algorithms, Structures and Processes
      row   0
      col   40

//Edit_Diagrams
      label       [Edit Diagrams]
      cmd         graspada ${Diagram_Name}
      msg         Control Structure Diagram Editor

///Diagram_Name
      label Diagram Name(s):
      text        (,,)
```

All the Hex numbers are the bitmap representation of the CSD icon. The line **cmd graspada ${Diagram_Name}** will specify what command to execute when the user selects CSD icon and clicks on execute.

The **dfe_pspec_edit** variable in ToolInfo file should be set as follows so that it invokes GRASP/Ada instead of standard Pspec editor.

dfe_pspec_edit=graspada&

The **sce_pdl_edit** variable in ToolInfo file should be set as follows so that it invokes GRASP/Ada instead of standard PDL editor.

sce_pdl_edit=graspada&

SUMMARY OF INTEGRATION STEPS

1. Copy the ToolInfo and STPmenu.spec files from StP library to user's home directory. Set the **ToolInfo** environment variable in .cshrc file to refer to the ToolInfo file in user's home directory.

2. Load the ToolInfo file (which is copied into user's home directory) into an editor and modify the ToolInfo variable **STPMenuSpec** as follows:

   STPMenuSpec=~/STPmenu.spec

3. To invoke GRASP/Ada in place of Pspec editor, replace the ToolInfo variable **dfe_pspec_edit** variable as follows:

   dfe_pspec_edit=graspada

4. To invoke GRASP/Ada in place of PDL editor,replace the ToolInfo variable **sce_pdl_edit** variable as follows:

   sce_pdl_edit=graspada

5. To invoke GRASP/Ada as an independent application (like DFE, SCE, etc.,) copy the information provided above in the **Structure of the Specification File** to STPmenu.spec file in user's home directory.

Appendix C

GRASP/Ada

Evaluation Instrument

Name _____ Date _____
(optional)

Evaluation of Graphical Representations for Algorithms

Several of the following graphical representations were briefly presented in class: flowcharts (FC), Nassi-Shneiderman diagrams (NS), Warnier-Orr diagrams (WO), action diagrams (AD), and control structure diagrams (CSD).

During this course, which of the above diagrams were presented? Check the appropriate responses.

FC	NS	WO	AD	CSD
—	—	—	—	—

Prior to this course, which of these diagrams had you used? Check the appropriate responses.

FC	NS	WO	AD	CSD
—	—	—	—	—

Were any of the diagrams used in a professional setting? Check the appropriate responses.

FC	NS	WO	AD	CSD
—	—	—	—	—

Based on the experience you have gained by using these diagramming tools to represent algorithms, you are asked to assign a rating to each of the diagrams with respect to a specific comparison among the diagrams. You may assign the same rating to more than one diagram for a given comparison. Select your ratings from the following scale and enter them as indicated below.

5 - best / most / first choice
4 -
3 - moderate
2 -
1 - worst / least / last choice

1. Compare the diagrams with respect to how well each shows **sequence**.

FC	NS	WO	AD	CSD
—	—	—	—	—

2. Compare the diagrams with respect to how well each shows **selection**.

FC	NS	WO	AD	CSD
—	—	—	—	—

3. Compare the diagrams with respect to how well each shows **iteration**.

FC	NS	WO	AD	CSD
—	—	—	—	—

4. Compare these diagrams with respect to overall **readability** (consider reading someone else's code).

FC	NS	WO	AD	CSD
—	—	—	—	—

5. Each of these tools can be used with informal pseudocode as opposed to actual statements in a programming language and, as such, can be thought of as a graphical extension to pseudocode (with possibly some spatial rearrangement). Rate the diagrams on the extent to which they increase readability over non-graphical pseudocode.

FC	NS	WO	AD	CSD
—	—	—	—	—

6. Suppose as a programmer you are given a design specification in which the program logic has been documented using one of the graphical representations below. Compare the diagrams with respect to which would best facilitate your task of coding from the design specification.

FC	NS	WO	AD	CSD
—	—	—	—	—

7. Compare the diagrams with respect to **ease of manual use**; consider the initial drawing and subsequent modifications.

FC	NS	WO	AD	CSD
—	—	—	—	—

8. Assuming you have to manually draw the diagrams (in the sense that they are not automatically generated), indicate your **overall preference** for each diagram where:
5 - first choice, . . ., 1 - last choice

FC	NS	WO	AD	CSD
—	—	—	—	—

9. Compare the diagrams with respect to their **overall economy** (i.e., increases in comprehension versus effort to draw them manually).

FC	NS	WO	AD	CSD
—	—	—	—	—

10. Assuming you have equivalent **automated support** to draw each of the diagrams in the sense that the diagrams are automatically generated either by selecting constructs from a menu or by recognizing key words in the code, indicate your overall preference for each diagram where:
5 - first choice, . . ., 1 - last choice

FC	NS	WO	AD	CSD
—	—	—	—	—

11. All assumptions aside, indicate your **overall preference** for each diagram where:
5 - first choice, . . ., 1 - last choice

FC	NS	WO	AD	CSD
—	—	—	—	—

12. It is not uncommon for individuals and organizations to introduce modifications (which they consider to be improvements) to "standard" diagramming tools. These changes may be to improve readability, to make the diagrams easier to work with manually, to make them easier to automate, to provide for control flow other than sequence, selection, iteration, etc. What **improvements** can you suggest for any of the diagrams we used in this class?

Name _____ Date _____
(optional)

Evaluation of GRASP/Ada

In CSE 422 lab you were provided the opportunity to work briefly with GRASP/Ada, a prototype reverse engineering tool for software written in Ada or AdaPDL. The prototype is currently being evaluated prior to widespread release via the network. As a prototype, GRASP/Ada is expected to undergo continual modification over the next year, especially with respect to integration with commercially available CASE tools.

The current GRASP/Ada prototype includes automatic generation of Control Structure Diagrams (CSDs). Future releases will include the generation of object diagrams. Your responses to the items below are intended to provide the developers with directions for enhancements to the prototype, including additional user interface requirements and overall functionality.

The GRASP/Ada Project is supported, in part, by funding from George C. Marshall Space Flight Center, NASA, Alabama 35821 (Contract Number NASA-NCC8-14).

1. Was the User Interface intuitive?

2. What changes would you make to the User Interface?

3. What were the major problems you encountered when using GRASP/Ada.

4. Rank the following items in order of importance in the prototype. Note, some of these items are available in the current version and others are under consideration as modifications/enhancements. Also, feel free to comment on each in the space provided. (1 - least important, ..., 7 - most important)

a. _____ Integration of CSD generation/editing capabilities with a CASE tool such as StP to facilitate development of process pspecs and/or module PDL.

b. _____ GRASP/Ada User's Manual.

c. _____ Error messages resulting from CSD generation.

d. _____ Integration of CSD editing/generation with automatic generation of object diagrams to show software architectural design (i.e., the object diagrams indicate the dependencies among a set of CSDs).

e. _____ Spatial options (line spacing, amount of indentation, etc.).

f. _____ Direct access to a compiler from the User Interface to facilitate use of the CSD during implementation.

g. _____ Extension of the CSD editor and generator to handle other languages such as C and Pascal.

5. Rate your knowledge of Ada.

_____ excellent _____ good _____ moderate _____ very little _____ virtually none

6. How useful was the Ada template feature in the CSD Window in producing Ada/PDL CSDs?

_____ extremely _____ very _____ moderately _____ not very _____ not useful

What modifications/improvements should be made to this feature?

7. The time in class spent on Ada and/or AdaPDL

_____ should have been increased. _____ was about right. _____ should have been decreased.

Comments?

8. CSD editors and generators are planned for C and Pascal. If these tools were available on the network, how useful would they be to you with respect to improving the readability of your source code in future software development projects?

C:

_____ extremely _____ very _____ moderately _____ not very _____ not useful

Pascal:

_____ extremely _____ very _____ moderately _____ not very _____ not useful