Controlling the Complexity of Menu Networks

James W. Brown
Jet Propulsion Laboratory

SUMMARY: A common approach to the design of user interfaces for computer systems is the menu selection technique. Each menu frame can be considered a node in an information/action network. The set of nodes and the permissible transitions between them (menu selections) form a directed graph which, in a system of substantial size, can be large and enormously complex. The solution to this problem of unmanageable complexity is the same for menu networks as for programs: the disciplined use of a set of well-defined one-in-one-out structures. This paper defines a set of such structures and offers some guidelines for their use.

1. Introduction

There are many techniques commonly used for communication between humans and computer systems. They vary widely in their ease of learning and use, and their general applicability. One technique for human-computer interaction is menu selection. It makes the most of the computer's ability to find and display large quantities of information rapidly and the human's ability to make decisions in the context of a specific problem. It also recognizes the slowness of the input channel, i.e., fingers typing (or hunt and pecking) on a keyboard.

2. Menu Networks

The menu selection technique presents the user with a sequence of "frames" or "pages," each containing some text and a list of options. The text offers information to the user, and the options allow the user to choose what to do or where to go next, from a limited set of possibilities. The user indicates a choice by typing a single character, pointing, or other techniques [4]. The selection made determines which frame will be displayed next.

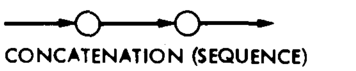
Since each frame has several options linking it to other frames, a frame can be thought of as a node in a network or graph, and the option links then correspond to "arcs" or "edges." Moreover, since the option selection represents a one-way transition from one node to the next, the menu system (the collection of frames and option links) forms a directed graph.[1]

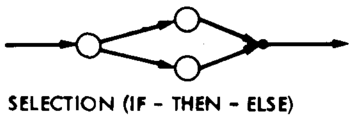
Menu systems are commonly used for two purposes: controlling the actions of computer applications systems and presenting information. Systems of the former type typically contain tens or hundreds of frames, with the latter possibly containing tens of thousands [7, 8]. If the links between frames allow for great richness of interconnection, the resulting graph can become so complex that understanding or modification becomes virtually impossible.

Such a situation is analogous to the unmanageable complexity which can occur with a large computer program containing many multiway branches; e.g., a program whose control flow consists of a hundred to ten thousand computed GOTOS, each with six to eight destinations, all interconnected, apparently at random. This kind of disaster has been averted by several modern techniques, especially top-down structured programming. In fact, that technique suggests a useful approach to the design of menu networks, which can be called "structured subgraphs."

CR Categories and Subject Descriptors: H.1.2 [**Models and Principles**]: User/Machine Systems–*human factors*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval–*query formulation*; H.3.5 [**Information Storage and Retrieval**]: On-Line Information Services.
General Terms: Design, Human Factors.
Additional Key Words and Phrases: Menu systems, graphs, networks, structured design tools.
This paper presents the results of one phase of research conducted at the Jet Propulsion Laboratory, California Institute of Technology, under Contract NAS7-100, sponsored by the Information Systems Office of the National Aeronautics and Space Administration, Office of Space Science and Applications.
Author's present address: J.W. Brown, Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, CA 91109.

[1] In the illustrations we have taken some liberties with the graph-theoretic notion of an arc, several arcs having been combined into one with the joining dot notation.





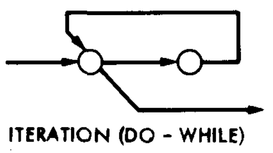


Fig. 1. Basic Structured Subgraphs.

3. Basic Structures

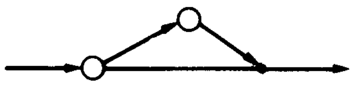
Following the example of structured programming, we define a basic set of one-in-one-out elementary graphs, which can be used as building blocks to form arbitrarily large networks. Like the control-flow graph of a structured program, any network constructed by appropriate concatenation or nesting of these structures will be representable as a planar graph. In yet another analogy to structured programming, we find that three basic structures, equivalent to sequence, selection, and iteration, are sufficient to deal with a wide variety of applications, but some orderly extensions to these provide much greater convenience and clarity.

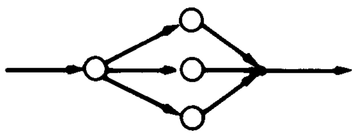
Figure 1 presents the three basic structures. The terms in parentheses relate the structures to commonly used programming constructs. We see that these do not represent very interesting or useful cases in the context of menu systems. The first, concatenation, simply leads the user from one frame to the next without the necessity or opportunity of making any decision. The two frames could have been combined into one. The second structure, selection, allows the user a choice between two options. This is useful in some instances, but rather restrictive. The third form, iteration, is the strangest. It is hard to imagine many uses for such a structure in a menu system. Soon we shall see that the value of the basic structures lies in using them as templates for combining more complex structures.

4. Extended Structures

As in structured programming, the basic structures are rather confining, and so we look for useful ways to generalize them. Figure 2 illustrates three generalizations which are very useful in real applications. The first is a modification of IF-THEN-ELSE having a null ELSE-branch. This is commonly the entry point to any kind of menu network. The first frame asks, "Do you know how to use the system, or do you need help?"

The second frame (the THEN-branch) provides an explanation of how to use the system and proceeds back (without the need for user action) to the "main-line." This approach can be observed throughout a network which provides shortcuts for experienced users. It is also useful for implementing other kinds of optional side paths.





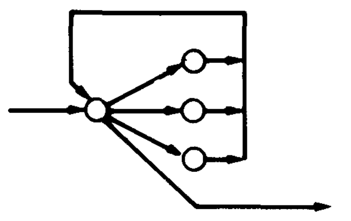


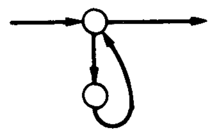
Fig. 2. Extended Structured Subgraphs.

The second extended structure, the "one-of-N," is the most common in most systems. It is the basis for the term "menu selection" and, as we shall see, generalizes into a tree structure, which is a very common form of a menu network.

The third extended structure, "M-of-N," is less commonly seen but is very useful. It permits the user to pick any number of entries (including zero) from a list, in any order. This is very important in applications with no obvious, natural order for presenting things. In such cases each user needs the freedom to make decisions in the order that seems appropriate at the time, given the user's specific knowledge, background, and orientation with respect to the problem at hand.

5. HELP Facilities

Menu systems are typically used in applications with little or no user training involved. A very well-designed menu network would be completely self-teaching, unambiguous,



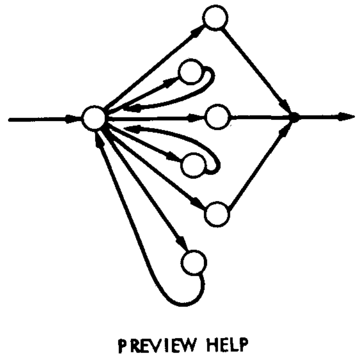


Fig. 3. HELP Structures.

and surprise-free. It may be possible to approach this ideal in application areas whose underlying subject matter has a well-defined and well-known structure, and the wording of the menu frames and options is carefully polished by incorporating feedback from many users over a long period of time. However, this degree of refinement cannot be achieved in many cases, and so it is necessary to provide HELP facilities which can be invoked when a user becomes confused.

Figure 3 illustrates one approach to providing such a facility. The help option is not explicitly included in the main frame's option list, but is invoked instead by some special, global mechanism. This mechanism may involve keying in "HELP", a "?", or simply "H", or pressing a special HELP key on terminals so equipped. This simple mechanism can only invoke a single help frame from any given main frame, although the help frame itself may generalize to a help network.

An extension of this technique is "preview help." This associates a different help frame with each option on the main frame. If an option is selected by entering a corresponding digit or letter, the help frame for that option can be selected by preceding or following the selection character with a question mark. Preview help is useful when a user does not understand the implications of making a particular selection and is afraid to "leap" before looking. The user can ask for clarification about any or all options before selecting one.

What is important to note about these help structures is that they are analogous to subroutine calls. They always return to the place from which they were called. No special action from the user, beyond perhaps indicating "ready," should be required to cause the return.

In order to keep drawings of the graphs uncluttered, help frames should not normally be included explicitly in the drawings. In practice, a well-designed network would include both types of help structures at every decision node where confusion might possibly arise. Single frames for each help node are usually best. Expanding a help node into a more complex subnetwork runs the risk of causing the user to forget why help was requested in the first place or that the current context is a help network rather than the "main" one.

6. Building Composite Structures

As in top-down structured programming, large networks can be constructed by combining the basic structures in two ways: nesting and concatenation. These are illustrated in Figure 4. Nesting is performed by replacing any *single-exit* node with a one-in-one-out structure. Concatenation simply connects the outgoing "half-arc" of one structure with the incoming half-arc of another. Where arcs join at joining dots, they can be redrawn to terminate properly on frame nodes. This appears to destroy the one-in-one-out property, but it actually does not. An alternative would be to define the joining dots as null nodes and keep them in the graph. This is simply a matter of the convenience of external representation versus mathematical rigor. The representation can be chosen to suit the purpose.

If the concatenation is done without crossing any lines, the graph always remains planar and well-structured. How important this is remains to be determined.

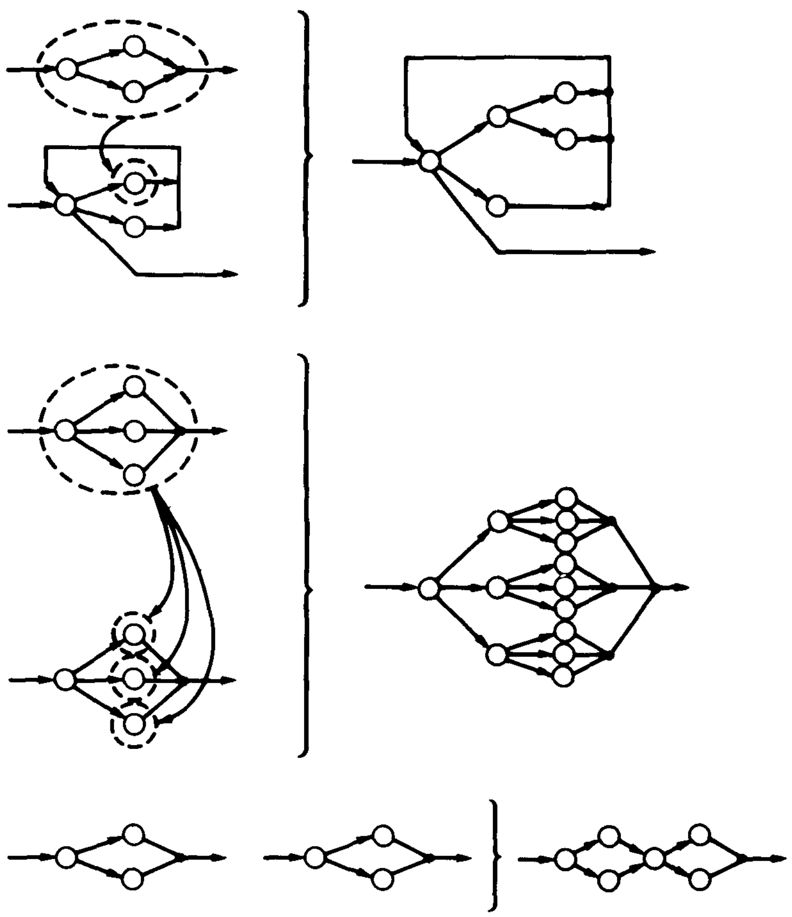


Fig. 4. Building Composite Structures.

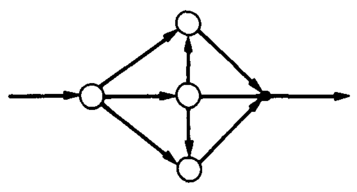


Fig. 5. A Nonstructured Planar Subgraph.

7. Noncanonical Forms

At this point one might well ask whether the structures and operations described above are sufficient for all interesting applications. In the case of programming, the answer would be a fairly emphatic yes (although with certain applications one must add provisions for dealing with interrupts and concurrent processes). In the case of menu systems, the answer might be no. There is at least one type of commonly encountered network, which is known to be useful although highly nonplanar. This is the cross-linked, multiple-tree structure often seen in multiply-indexed library databases, organization/person/task networks, and elsewhere. An important question is whether organizing information into network structures which seem "natural," although not "structured" in the sense defined above, leads to problems of unmanageable complexity when the networks become large. Our initial experience, discussed below, indicates that it may.

What are some other possibly useful forms? The subgraph in Figure 5 is simple-looking, planar, one-in-one-out, but note that it cannot be extended to more (or less) than three selection nodes without destroying some of its properties. (Observe that the 1-of-N and M-of-N structures extend uniformly for any N greater than zero.) A fourth selection node, with an arc leading from the existing center node, cannot be added without making the result nonplanar. Would such a structure be useful in a real menu system? Perhaps, but it might also be a warning signal that some redesign is in order.

Figure 6 presents a structure that, at first glance, seems the same as Figure 5. However, it is greatly different, both topologically and psychologically. It extends uniformly to any N, without becoming nonplanar. It is used to allow the user to choose many or all of the N selections without returning to the precedent node. Its successful use necessitates that the list of selections be either short and simple enough for the user to keep in short-term memory (since it will not be seen again) or sufficiently regular so individual items do not have to be remembered (e.g., the months of the year). It also requires that the user know in advance something about the local topology, and that each of the N selectable nodes be simple and nonconfusing.

The "next" option appears to be appropriate only with the 1-of-N structure. It can be generalized to the "circular next" at the cost of introducing a minor local nonplanarity. This does not appear to be too troublesome as long as the user is able to cope with the "next" concept in the first place.

8. Is the GOTO Harmful?

In considering the GOTO statement, we perhaps come to a place where the analogy between menu networks and computer programs weakens. A branch implies a decision, and in a menu network the decision-maker is a human being. The human user is present precisely because the decisions required *cannot* be preprogrammed. On the other hand, the user has a very limited ability to comprehend the static structure of an entire large network, whereas the computer can do so with ease. None of this tells us whether it is safe or desirable to use GOTO, only that the lessons learned in programming [2] may need to be modified.

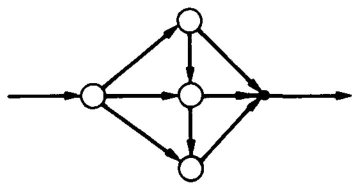
We should note that there are two types of GOTO in a menu system: a static or preprogrammed path in the network which violates the basic structuring rules, and a dynamic jump which a user may wish to construct from any node to *any* other. The first may be likened to an express highway allowing a large volume of traffic to bypass the local road network, or to a direct trunk group bypassing the basic hierarchy of a telephone network. The second type is more akin to teleportation, allowing the user to arrive instantly at any desired destination without having to pass the places between the jump-off point and the destination.

Direct paths are commonly found in the kinds of networks mentioned above, and they seem to be useful and manageable. Both the highway network and the telephone network are larger than any menu network we are soon likely to see, yet users can navigate them with ease. Maintaining and modifying them require massive efforts, but these efforts do not grow disproportionately with the size of the networks.

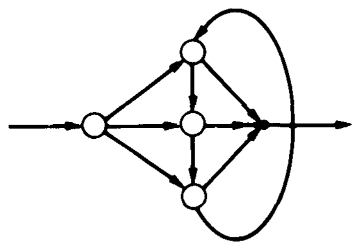
Conversely, the phenomenon of users becoming lost or disoriented in menu systems is common enough [7], except perhaps in networks which are simple trees. Without a great deal of difficult experimentation[2], it is impossible to relate the likelihood of getting lost with any measure of network complexity[3].

[2] [5] outlines some directions such experimentation might take.

[3] [1] discusses the relative merits of several complexity measures.



1 OF N WITH NEXT



1 OF N WITH CIRCULAR NEXT

Fig. 6. The "Next" Option.

9. Implementation Experience

The ideas described above were developed in the course of designing the menu-driven user interface for the NASA Oceanic Pilot System. This system is a pilot project intended to demonstrate techniques for providing convenient, rapid access to large volumes of satellite-derived data for the oceanographic research community. One of the requirements for the system is that infrequent, nonexpert users be able to easily use it. For this reason the menu selection technique was chosen as the primary user interface. Command language and DBMS query language interfaces supplement the menu interface for expert users.

The menu interface allows access to the following basic system capabilities:

(1) A bibliography of ocean remote sensing literature;
(2) A directory of available data sets, consisting of high-level "abstracts" and availability information;
(3) A detailed inventory of available data, including temporal and spatial location and access information;
(4) Extraction of user-specified subsets of the archival data sets;
(5) Formatting of extracted data into a variety of output products, such as tabulations, plots, maps, magnetic tapes, and disk files;
(6) Support functions such as saving and retrieving request specifications and files, status monitoring, accounting, etc.

In addition, the menu system includes a self-teaching option for new users. This feature guides the user through the various capabilities of the menu processor itself and provides brief overviews of the various system capabilities.

A prototype menu network was initially designed and implemented with the designers' view of a typical use scenario in mind. The prototype network consisted of about 50 nodes, organized without regard to the structuring ideas described above. When the prototype was exposed to trial users, who were in fact more knowledgeable about the system than real users were expected to be, several important points quickly became evident. First, users can easily get lost in even a relatively small network. Second, users demand more flexibility in the *order* in which they specify or receive information than designers are likely to think necessary. Third, modifying an unstructured network, even if small, is unacceptably difficult.

These observations led to the conclusion that a more disciplined methodology would be needed for the real network. The author's experience with top-down structured programming led naturally to the ideas discussed in this paper. A review of the literature on human factors in computer systems, starting with [6], indicated that no other work had been done in this area. Therefore, the concepts presented here may be considered tentative first steps in this direction.

Although our prototype menu network proved imperfect, the results of our application were encouraging. The version of the network delivered for initial operational use is nearly an order of magnitude larger than the prototype, as illustrated in Table I. It has undergone several modifications as features were added or relocated, but all changes have been made smoothly. Users still get lost occasionally, but not as frequently as in the prototype. It is not clear, though, that the structuring was responsible for the latter effect, since considerable effort also went into polishing the wording of those menu frames which caused confusion.

One area of the implementation which remains unsatisfactory is the bibliography. The initial version provides access only by subject and is organized as a basic tree with limited cross-indexing added. The effort required to organize the references into a hierarchy of subject categories and then implement the access frames was substantial. In addition, users sometimes want access by author, institution, title, report type, project, sensor, and key word. Creating access trees for each of these categories is a formidable task and still would not always provide the kinds of multiple selectors which users desire.

As a result, we plan to redesign the bibliography as a relational database, using the menu system to help users build queries. This is expected to yield much more flexible and selective access for users, while substantially reducing the effort of adding new entries to the database.

10. Design Aids

The initial version of the frame network was developed manually, assisted by a single program which analyzes the network and produces

Table I. Oceanic Pilot System Menu Network—Initial Version.

Category	Number of non-HELP Frames	Number of HELP Frames
Bibliography		1
access paths	50	
citations	203	
Data set directory		1
access paths	17	
citations	10	
Data inventory and extraction	37	13
Output product specification	34	12
Support (request manipulation, file saving, etc.)	21	21
Tutorial	20	3
Total	392	51

72 frames invoke 67 programs as subprocesses.

```
Oceanic Pilot System Menu Network
SECTION Construct_a_request
   LOOP until case 6 or 7 is selected
      'You will get a chance to review and modify your request'
      'before executing it.  You can return to this page as'
      'often as necessary to complete it.'
      SELECT menu option:
      CASE 1:
         DO Specify_region_of_interest------------------------------->(  4)
      CASE 2:
         DO Specify_time_span_of_interest---------------------------->(  5)
      CASE 3:
         DO Specify_platform_sensor_parameters_of_interest----------->(  5)
      CASE 4:
         DO Specify_processing_level--------------------------------->(  5)
      CASE 5:
         DO Check_completeness_of_request---------------------------->( 10)
      CASE 6:
         DO Construct_output_product_request------------------------->( 10)
      CASE 7:
         GOTO Modify_or_process_request------------------------------>(  3)
      ENDSELECT
   REPEAT until case 6 or 7 is selected
END
SECTION Modify_or_process_request
   LOOP until case 7 is selected
      CALL Check_request------------------------------------------->( 29)
      PAUSE
      SELECT menu option:
      CASE 1:
         DO Display_request------------------------------------------>( 19)
      CASE 2:
         DO Modify_request------------------------------------------->( 19)
      CASE 3:
         DO Save_request_for_future_use------------------------------>( 21)
      CASE 4:
         DO Execute_sample_of_request-------------------------------->( 22)
      CASE 5:
         DO Estimate_cost_for_full_request--------------------------->( 22)
      CASE 6:
         DO Execute_full_request------------------------------------->( 22)
      CASE 7:
         Exit to Begin_request_specification (Erase and restart)
      ENDSELECT
   REPEAT until case 7 is selected
END
```

Fig. 7. SDDL Printout of Two Menu Subnetworks.

```
PAGE      8 -- Construct a request
You will get a chance to review and modify your request before executing
it.  You can return to this page as often as necessary to complete it.
1-Specify region of interest
2-Specify time span of interest
3-Specify platform/sensor/parameter(s) of interest
4-Specify processing level
5-Check completeness of request
6-Construct output product request  (Do above steps first)
7-Modify or process request

 (?HELP -BACK +NEXT *TOP :REFRESH !GRIPE &TUTOR #GOTO >CMD $LOGOFF)
SELECT > 7

PAGE    100 -- Modify or process request
NOW CHECKING COMPLETENESS OF YOUR REQUEST...
REQUEST IS COMPLETE.  READY TO PROCEED.
PRESS <RETURN> TO CONTINUE
1-Display request
2-Modify request
3-Save request for future use
4-Execute sample of request  (results to terminal)
5-Estimate cost for full request
6-Execute full request
7-Begin request specification (erase and restart)

 (?HELP -BACK +NEXT *TOP :REFRESH !GRIPE &TUTOR #GOTO >CMD $LOGOFF)
SELECT >
```

Fig. 8. Menu Processor Output of Frames Defined in Figure 7.

lists of: frames which invoke subprocesses, frame titles, frame pointers, pointer mismatches, spare frame numbers, and syntactical errors in the control information embedded in the frames. This sort of tool is essential to weed out errors in manually created frame networks. Fortunately, it is easy to build—ours is less than 300 lines of code, including comments.

Because the network is expected to grow substantially, it has become necessary to look for more automated support for the design and introduction of new subnetworks. One existing tool that has been used for this application is the Software Design and Documentation Language (SDDL) [3]. This is a "structured English" processor which is usually used for designing and describing program code, but which lends itself easily to a great variety of novel applications. Sample frame definitions using SDDL are shown in Figure 7, with the corresponding frames displayed to the user depicted in Figure 8. At this time, no tool is available to translate the SDDL representation into the format actually used to drive the menu processor. The translation continues to be done manually. However, the SDDL processor provides enough support, in the form of "call trees," cross-references, and diagnostics, to eliminate all "topological" errors, such as mismatched or hanging pointers. The translation required is straightforward and can, for example, be automated by the modification of an existing structured Fortran preprocessor. Since errors introduced by manual translation are easily found and corrected, this additional automation does not appear to be warranted at this time.

11. Measurement

Because the system has not, at least as of this writing, been released to operational users, we have not collected use-related statistics. However, the menu processor does produce a detailed log of all user inputs, with each entry time-stamped to ten millisecond resolution. In addition, there is an easy-to-use facility which allows users to enter comments into the log. Such comments are also time-stamped and can be analyzed in the context of the session history. From this log we intend to extract such information as: frequency of use for each frame and link, distribution of residence times at each frame, use of the dynamic GOTO and related commands, use of the HELP facility, requests for help when none is available (not all frames have HELP frames), and use of the user comment facility. In addition, we will monitor the use of the command language interface to determine to what extent users migrate from the menu mode to the command mode.

12. Conclusions

Although data is still lacking, the following principles for the design of menu networks seem reasonable:

(1) If the underlying information has a stable, well-understood, and well-known structure, allow the network to reflect that structure.

(2) Lacking such a structure, use the principles of top-down structured network design described above.

(3) If a demand for highly travelled shortcuts arises (measure it!), install direct routes (static GOTOS) where they are needed. Remove those that are infrequently used. Keep track of which links violate the structuring canons and always be alert to opportunities for eliminating some.

(4) Provide a dynamic GOTO mechanism which users can either use or ignore. Monitor its use. Install static GOTOS for frequently used paths.

(5) Look for software tools (analogous to structured design and structured programming languages) to aid in the construction, analysis, and maintenance of structured networks.

Acknowledgments

A great deal of insight into the problems of users "getting lost" in a menu network resulted from discussions with the staff of the ZOG project at Carnegie-Mellon University. Also, the highway analogy and the "next" operation were inspired by ZOG.

References

1. Baker, A.L., and Zweben, S.H. A comparison of measures of control flow complexity. Proc. IEEE Third Internat. Comptr. Software and Applications Conf., Chicago, Ill., Nov. 6–8, 1979, pp. 695–701. Discusses the advantages and limitations of several complexity measures for program flow graphs.

2. Dijkstra, E.W. Go to statement considered harmful. *Comm. ACM 11*, 3 (March 1968), 147–148.

3. Kleine, H. *Software Design and Documentation Language*. Pub. 77-24, Revision 1, NASA, Jet Propulsion Lab., Calif. Inst. Tech., Pasadena, Calif., Aug. 1977. Language reference with examples of use. This design language processor has been successfully used for a wide variety of design and documentation tasks, from high-level functional design through detailed code design.

4. Martin, J. *Design of Man-Computer Dialogs*. Prentice-Hall, Englewood Cliffs, N. J., 1973. A survey and discussion of interactive man-machine interface techniques.

5. Newell, A. Notes for a model of human performance in ZOG. Tech. Rep., Dept. Comptr. Sci., Carnegie-Mellon Univ., Pittsburgh, Pa., Aug. 1977. An outline and motivation for an experiment for measuring some human performance parameters in the context of a large-network, rapid-response menu system.

6. Ramsey, H.R., Atwood, M.E., and Kirshbaum, P.J. A critically annotated bibliography of the literature on human factors in computer systems. Tech. Rep. SAI-78-070-DEN, Sci. Applications, Inc., Englewood, Co. (available from Defense Tech. Inform. Ctr., Alexandria, Va.). Abstracts and critiques of 564 papers selected from an initial set of 20,000 citations.

7. Robertson, G., McCracken, D., and Newell, A. The ZOG approach to man-machine communication. Tech. Rep. CMU-CS-79-148, Carnegie-Mellon Univ., Pittsburgh, Pa., Oct. 1979. Describes a system designed for research into the characteristics of menu-driven man-machine communications. Discusses the importance of large networks and some general issues concerning menu selection.

8. Schultz, J., Cantrill, S., and Morgan, K. An initial operational problem oriented medical record system—For storage, manipulation and retrieval of medical data. Proc. AFIPS 1971 Spring Joint Comptr. Conf., Vol. 38, AFIPS Press, Arlington, Va., 1971, pp. 239–264. Most of this paper is devoted to describing the way the system is used in its specific application domain. However, it does illustrate a large, successful, menu-driven application system.