NASSI-SHNEIDERMAN CHARTS
AN ALTERNATIVE TO FLOWCHARTS FOR DESIGN

by
Cornelia M. Yoder and Marilyn L. Schrag
IBM Corporation
System Products Division
Endicott, New York 13760

ABSTRACT

In recent years structured programming has emerged as an advanced programming technology. During this time, many tools have been developed for facilitating the programmer% use of 'structured programming. One of these tools, the Structured Flowcharts developed by I. Nassi and B. Shneiderman in 1972, is proving its value in both the design phase and the coding phase of program development.

Several programming groups in System Products Division, Endicott, New York, have used the Nassi-Shneiderman charts as replacements for conventional flowcharts in structuring programs. The charts have been used extensively on some projects for structured walk-throughs, design reviews, and education.

This paper describes the Nassi-Shneiderman charts and provides explanations of their use in programming, in development process control, in walk-throughs, and in testing. It includes an analysis of the value of Nassi-Shneiderman charts compared to other design and documentation methods such as pseudo-code, HIP0 charts, prose, and flowcharts, as well as the authors' experiences in using the Nassi-Shneiderman charts.

The paper is intended for a general data processing audience and although no special'knowledge is required, familiarity with structured programming concepts would be helpful. The reader should gain insight into the use of Nassi-Shneiderman charts as part of the total development process.

INTRODUCTION

The search for a good design method and a good documentation method to use in conjunction with structured programming has resulted in several widely varying yet very useful techniques. One of these is the 'Structured Flowcharts of I. Nassi and B. Shneiderman.

The Nassi-Shneiderman Charts have many features to recommend them for use in top-down structured programming environment, not the least of which is the difficulty of designing unstructured programs using the charts. The authors were introduced to this charting technique in 1973 and have been using it successfully on their respective projects since that time. The use of Nassi-Shneiderman Charts has spread to almost everyone who has been exposed to their use as the need' to express and verify the design and documentation of programs has increased.

Nassi-Shneiderman Charts or N-S Charts is not the only design language that has been developed recently; for example, pseudo-code is another excellent technique. Nor is N-S Charting the only documentation method; pseudo-code can be used for program documentation, and HIP0 charting also has many advocates,

One of the purposes of this paper will be to compare N-S Charts to other design and documentation methods in a constructive way - each of the different techniques has its advantages and disadvantages, and each is useful in certain situations. Before these comparisons, there is a brief introduction to Structured Programming followed by a description of the Nassi-Shneiderman Charts.

STRUCTURED PROGRAMMING CONCEPTS

Structured programming, contrary to some programmers' beliefs, is not a set of coding rules and restrictions. Structured programming is a style, an attitude toward programming that starts with fundamental goals of the programming process. Classically, programming goals were correctness, efficiency, and creativity. Of these, correctness is the only valid programming goal remaining today. Efficiency has become of minor importance with the advent of very high speed computers and virtual memories. Creativity, not bad itself, was classically directed toward cleverness and obscurity, with.frequently detrimental results.

In today's programming environment new goals have been set. Correctness remains of primary importance; however, maintainability (the ease of fixing errors), modifiability (the ease of making changes), and readability (the clarity of the program) have replaced program efficiency and abstruseness as desirable program characteristics. The programmer who sacrifices modifiability to save a few bytes or who gleefully hands over a program saying, I'll bet you can't guess what this one does!" is finally receiving the disdain so long deserved.

Once these new programming goals were set, it was inevitable that many programming techniques would be developed to achieve them. One of the best techniques has been structured code. (The distinction in terms is clear. Structured coding is the set of standard coding methods for accomplishing the goals of structured programming.) Structured coding is a set of program structures sufficient for writing any proper program (one entry point and one exit point) together with some rules for segmentation and indentation. The required set of program structures is not unique; one

Reprinted from *Proceedings, ACM SIGSOFT/BIGMETRICS Soft ware and* Assurance Workshop, November 1978. Published by 'ACM, reprinted by permission.

minimum set consists of structures titled SEQUENCE, IFTHENELSE, and DOWHILE. Frequently, other structures such as DOUNTIL and CASE are included. These structures are diagrammed using conventional flowchart symbols in Figure 1.

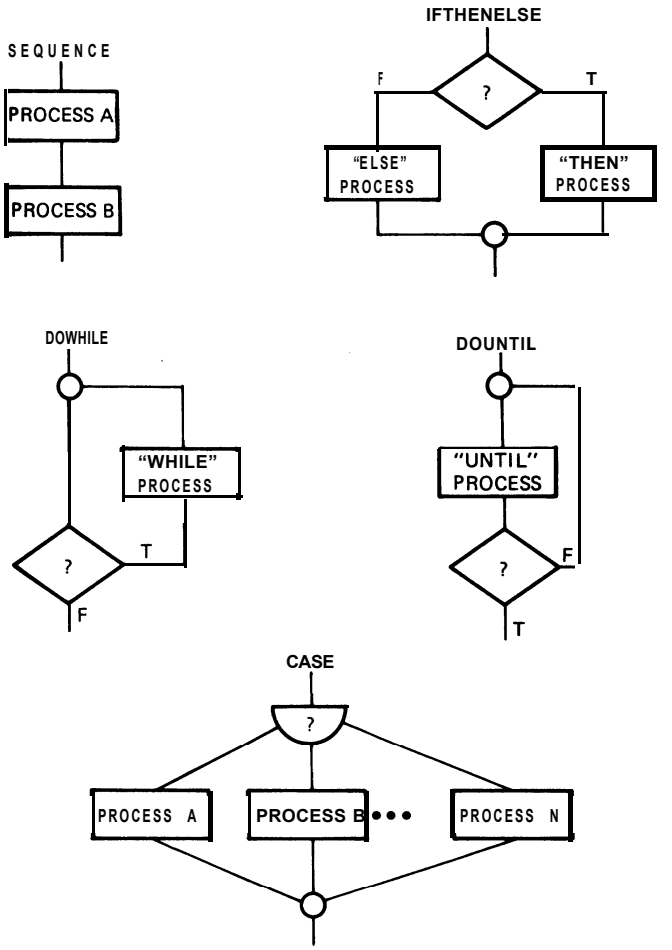

Figure 1. Required Set of Program Structures

NASSI-SHNEIDERMAN CHARTS

In SIGPLAN Notices of the ACM, August, 1973, Messrs. Nassi and Shneiderman published a new flowcharting language with a structure closely akin to that of structured code. The advantages they claimed for their charts have proven correct; they are as follows:

1. The scope of iteration is well-defined and visible.

2. The scope of IFTHENELSE clauses is well-defined and visible; moreover, the conditions or process boxes embedded within compound conditions can be seen easi ly from the diagram.

3. The scope of local and global variables is immediately obvious.

4, Arbitrary transfers of control are impossible.

5. Complete thought structures can and should fit on no more than one page (i .e. no off-page connectors) .

6. Recursion has a trivial representation.

The authors have added another advantage to those listed above:

7. These charts are adaptable to the peculiarities of the system or language they are used with.

By combining and nesting the basic structures, all of which are rectangular, a programmer can design a structured, branch-free program. Figure 2 shows

Figure 2. Process Symbol

the basic PROCESS symbol - a rectangle representing assignments, calls, input/output statements, or any other sequential operations. In addition, a PROCESS symbol may contain other symbols nested within it. The PROCESS symbol may be of any chosen dimensions provided the symbol fits on one page. The symbol used to represent a decision is shown in Figure 3.

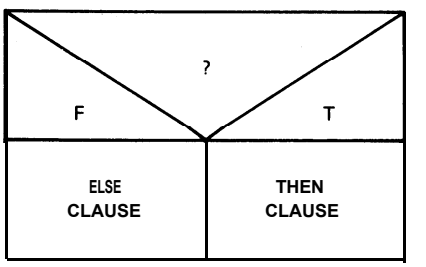

Figure 3. Decision Symbol

This IFTHENELSE symbol contains the test or decision in the upper triangle and the possible outcomes of the test in the lower triangles. "Yes" and "No" may be substituted for "True" and "False," and there is no particular objection to switching them right and left, although consistency is desirable. The rectangles contain the functions to be executed for each of the outcomes. Notice the ELSE and THEN clause boxes are actually PROCESS symbols and may contain any valid PROCESS statements or nested structures.

Repeating processes are represented by an iteration symbol. One of three symbols may be used depending on whether loop termination is at the beginning or the end of the loop. Figure 4 shows a DOWHILE symbol, used for loops which test a condition at the beginning.

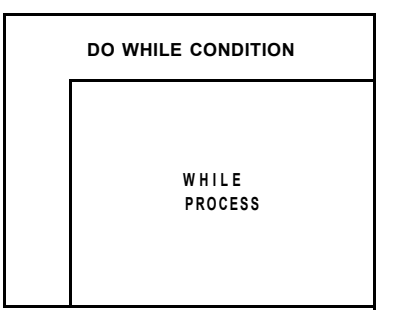

Figure 4. DOWHILE Symbol

Figure 5 is a DOUNTIL symbol, for loops which test for termination at the end. Figure 6 is a combination for loops with compound tests and may also be used for special constructs such as DO FOREVER or for setting off BEGIN/END blocks.

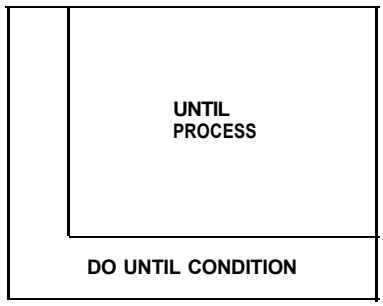

Figure 5. DOUNTIL Symbol

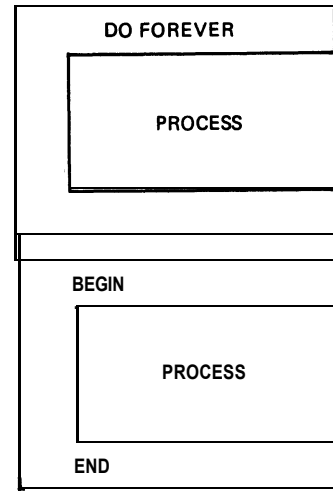

Figure 6. Other Acceptable Symbols

The CASE structure is represented by the symbol in Figure 7. This form of CASE requires the setting of a variable to an integer value, and the choice of path is based on the value of that variable. Figure 8 depicts a more powerful form of CASE, but one that requires the designer to be certain the conditions chosen are mutually exclusive and cover all necessary condition testing.

Nesting of structures to create programs should now be an obvious extension of the use of basic symbols. Figure 9 shows an N-S chart to calculate and print an FICA report in a style useful to designers. Figure 10 shows the same chart written in a style closer to the programming language, such as programmers might use.

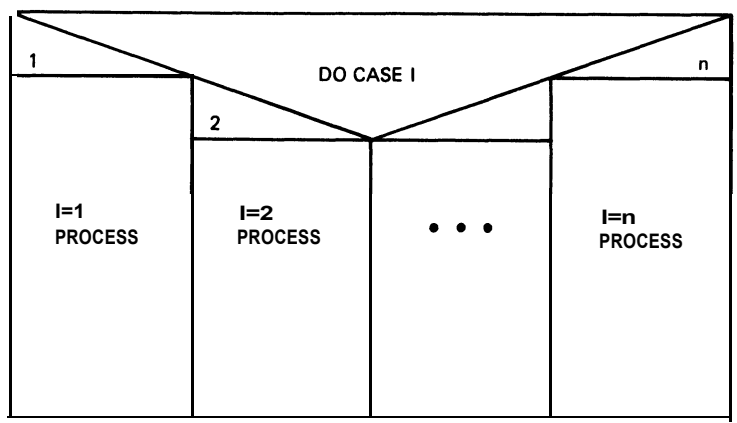

Figure 7, CASE Symbol

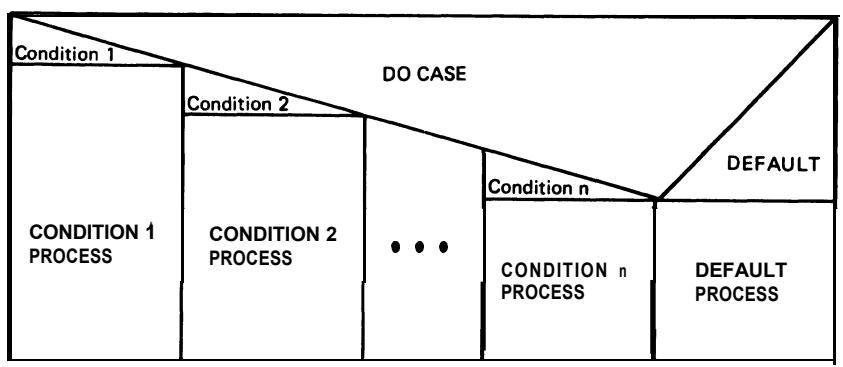

Figure 8. Alternative CASE Symbol

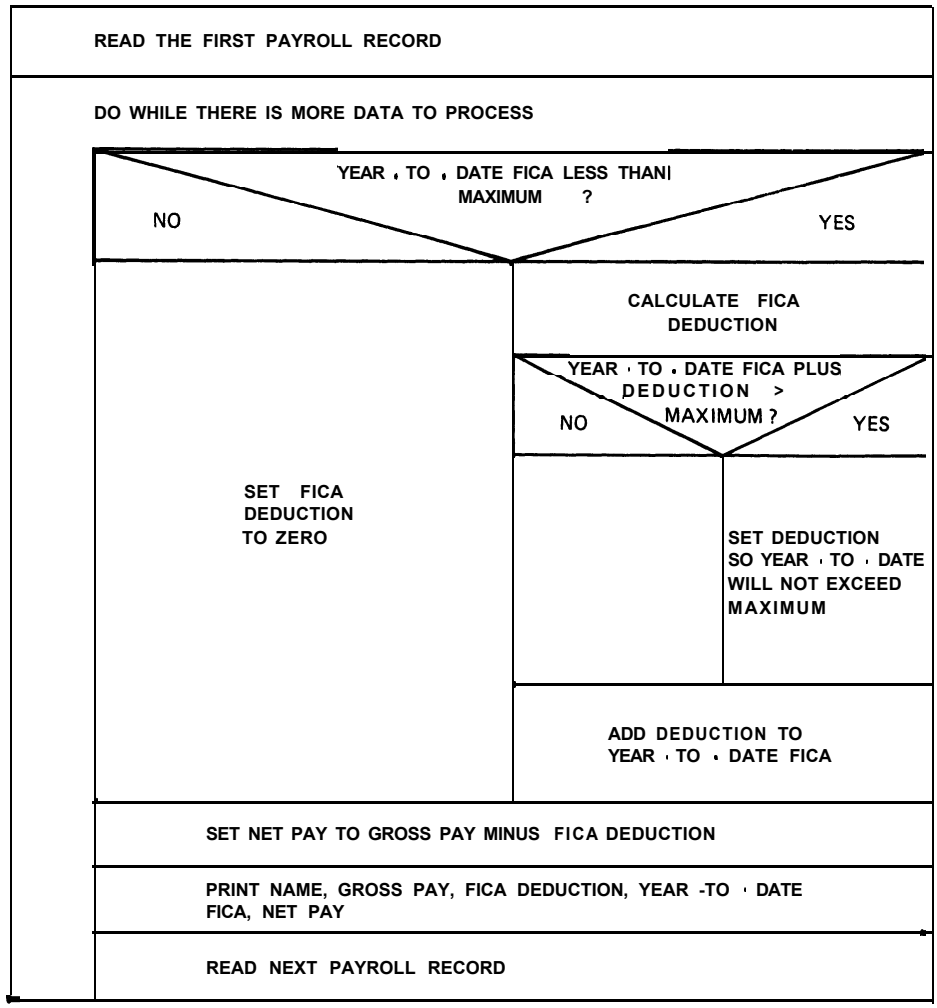

Figure 9. Example of N · S Chart Used For Design

USING NASSI-SHNEIDERMAN CHARTS

The practical use of N-S charts requires some basic techniques for optimum benefits. The major uses of N-S charts fall into three categories: Creating the logic design, programming from the charts, and writing program documentation. In addition, the N-S charts can be used for higher level design and procedural documentation; they have also been very well received for walkthroughs and design reviews.

I. CREATING THE LOGIC DESIGN

Nassi-Shneiderman charts were developed as a better way than traditional flow charts to describe the logic of a structured program. Drawing the chart and developing the logic go hand-in-hand, with the constraints of N-S charts (single page, no branch symbols) forcing the development of a structured design, that will in turn lead to structured code.

How to Start

Let us assume that functional design for a project has been completed, and that a modular design technique was used to determine function, input, and output for each module to be programmed. The programmer is now ready to design logic for structured coding.

The N-S chart starts with a rectangle drawn at the top of the page. This block might be any of the N-S symbols, depending on the module's function. If the module requires initialization of some variables, the first block is probably a processing symbol. If the module's function is performed repeatedly, a block with an iterative symbol will be close to the top of the page. If the function to be performed is conditional, a decision symbol will be used initially.

Arranging the N-S Structure

When a block is drawn symbolizing a decision, the programmer must make an actual decision about the assignment of processing paths on the chart (see Figure 11) . An effective technique is to locate on the right the path which in coding would be equivalent to the 'then' clause of an 'if' statement, and to locate on the left the path equivalent to the 'else' clause (see Figure 12). A consistent technique for path assignment makes the chart easier to draw and to read,

Suppose repeated decisions must be made. It is possible that much of the page would be taken up by blank paths (corresponding to null ELSE statements), or by paths with little processing. Very little room would then be left for describing the main processing path of the program (see Figure 13) .

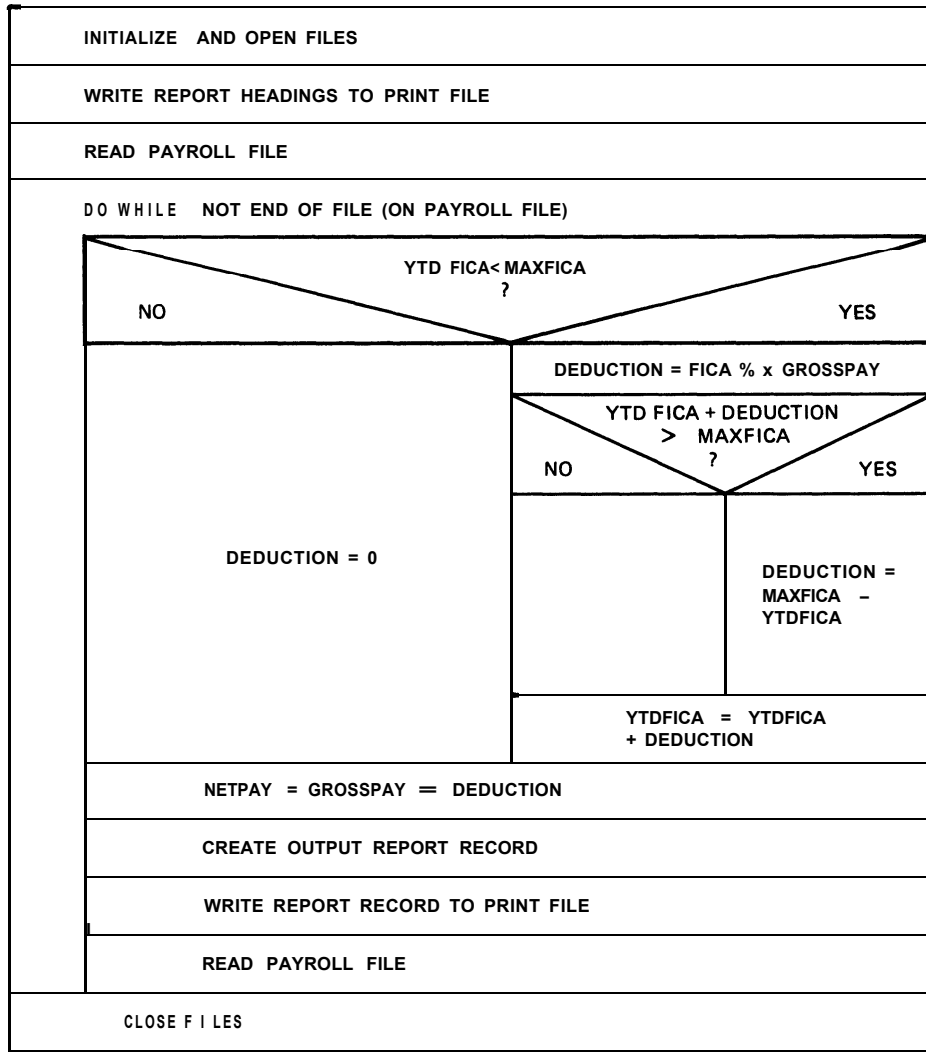

Figure 10. Example of N - S Chart Used For Coding

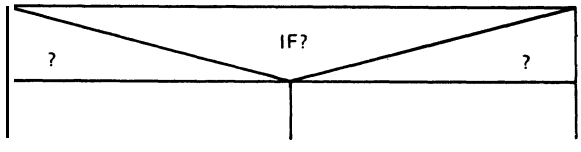

Figure 11. Choice of Path Assignment

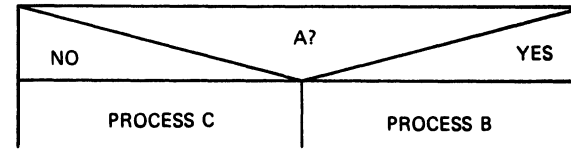

Figure 12. IF A THEN Process B ELSE Process C

When to Segment

As the programmer continues to draw the chart and develop the design, nested iteration and decision symbols will cause the blocks to get increasingly smaller. If the programmer did not give some thought initially to segmentation of the module, he or she may find that space has run out on the chart before the design is complete. Any rectangular position of an N-S chart can be removed from the main routine, replaced by a processing block, and made a separate segment or internal subroutine with its own N-S chart. Figure 15 shows in dark lines three of the possible segments which could be removed from the main routine. The choice should depend on the extent to which the portion that is removed constitutes a single function.

Differences in N-S Structure

The N-S chart for a module will visually reflect the design of the module. It may be large or small, complex or simple, depending on the function to be performed. A module at the top of the modular design hierarchy will consist mainly of calls to lower-level modules and evaluation of return codes. Its chart will probably have a diagonal look as can be seen in Figure 16.

A module at the lowest level will perform the actual processing of the data. Figure 17 shows such a module, with a single call to a service module to write an error message.

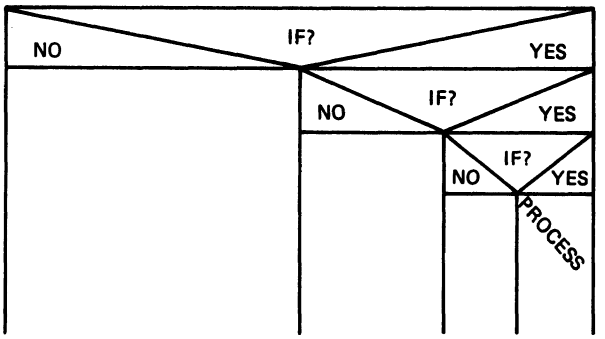

Figure 13. Unskewed N · S Chart

To allow more room on the chart for describing the processing paths, the decision triangle can be skewed, as in Figure 14, allocating space as it is needed.

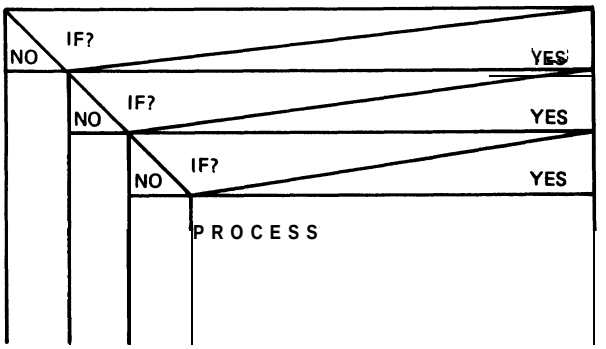

Figure 14. Skewed N · S Chart

II. PROGRAMMING FROM N-S CHARTS

Once the logic design for the module is completed, coding and testing of the module can begin. In both coding and testing, the N-S chart serves as a guide.

Coding

Translating from the N-S chart to code, especially in a high level language, is very easy; this ease is one reason why N-S charts have been accepted enthusiastically by programmers who have tried them.

The code will be structured; there is no possibility of a branch, and the coded segments wi ll be small. IFTHENELSE statements are well defined by the chart, as are the limits of DO structures.

Testing

The N-S chart can be used as a guide while testing the module. The number of test cases which will be required may be readily determined by counting decision blocks (count 2 per decision) and iteration blocks (count 2 or 3 per loop, depending on boundary conditions of the loop).

The precise test cases needed and data required may be developed directly from the charts, and the tested paths may be checked off on the charts as tests are executed.

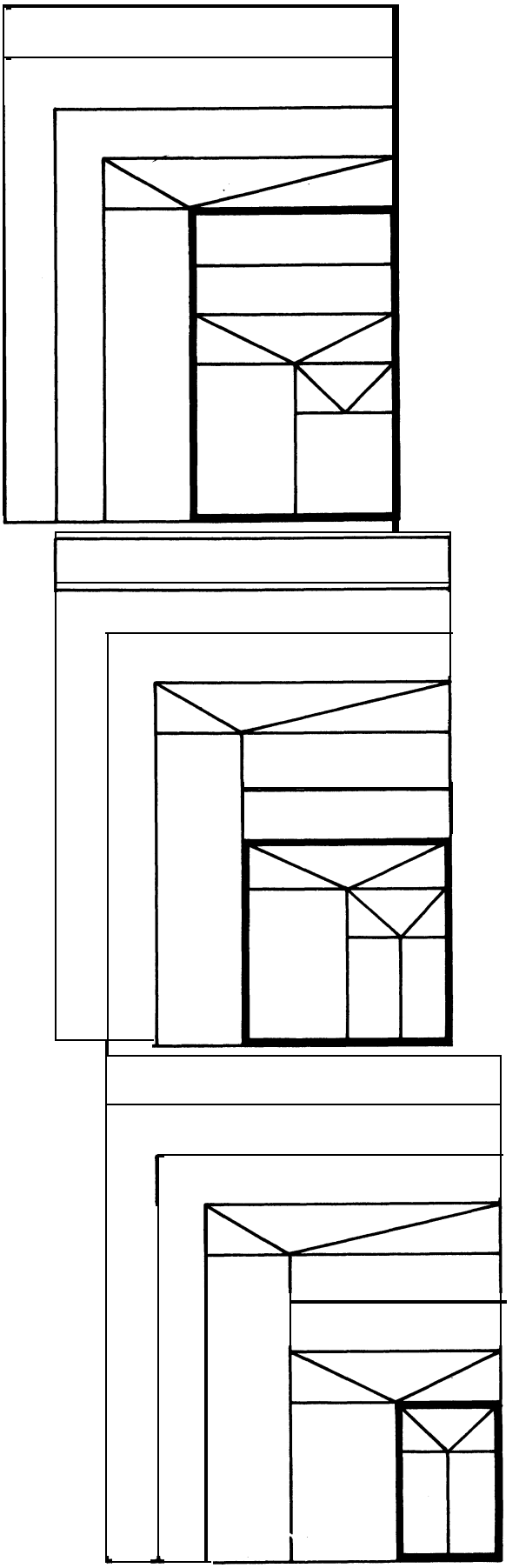

Figure 15. Possible Choices for Segmenting an N-S Chart

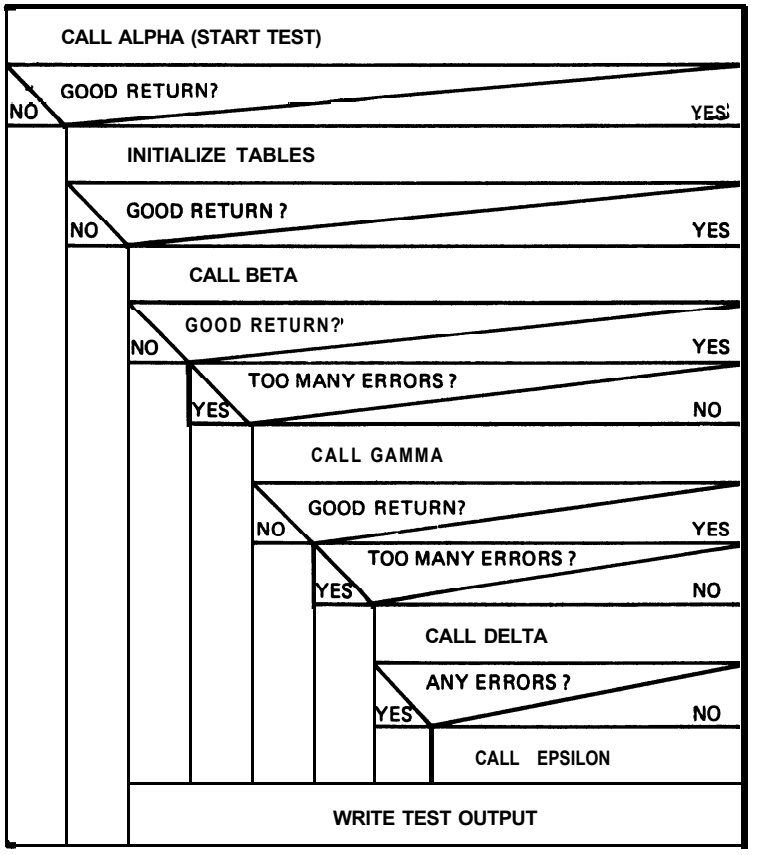

Figure 16. Diagonal N • S Chart

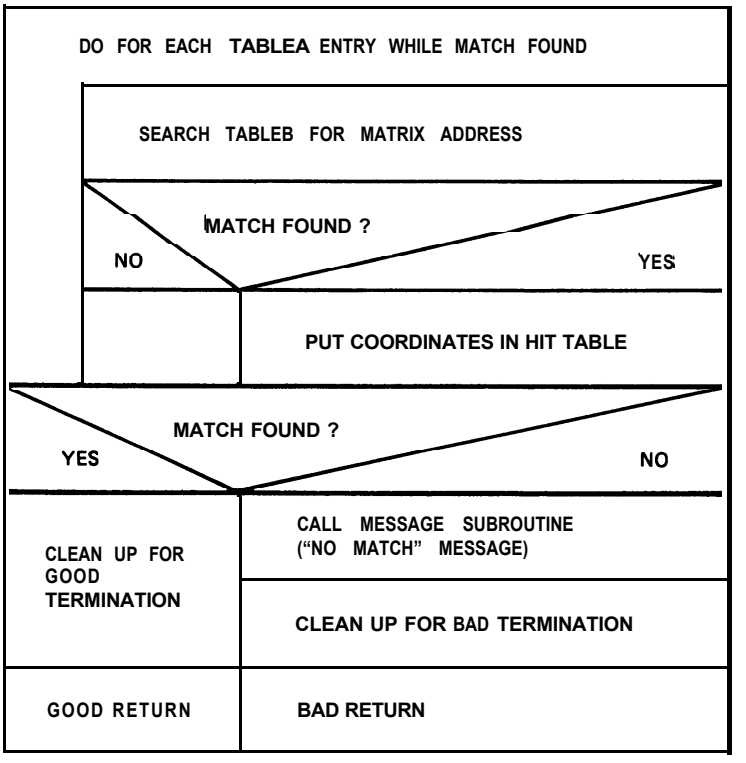

Figure 17. Nondiagonal N • S Chart

III. N-S CHARTS AS PROGRAM DOCUMENTATION

The Nassi-Shneiderman chart is a graphic representation of a module's logic design and a blueprint for the code. This makes it an excellent tool to use in educating other programmers on the function of the module. An N-S chart provides a maintenance programmer with a quick reference for finding the code performing any logical function.

IV. OTHER USES

Other parts of this paper described the use of N-S charts to design and program structured code. The high acceptance level of the charts by programmers who have used them indicates that use of the symbols may expand to other areas.

For example, a possible use may be in functional design. Process blocks can be described in general terms, rather than at the detailed level used for logic design.

As usage of N-S symbols extends beyond programmers to people in other technical areas, as have the symbols of traditional flowcharts, they can become a part of user's procedural documentation.

An area where use of N-S charts has already expanded beyond initial expectations is for presentations at walkthroughs and review. The graphic, visually descriptive qualities of the charts make them easy to use as presentation aids when describing program function to users and other nonprogramming people. Code inspections are significantly easier when a corresponding N-S chart is available to graphically depict the code being inspected. If N-S charts are used for design inspections, then code may be inspected directly against them.

CONCLUSIONS

Many design/documentation methods are in use today; some of these methods have existed for many years, and some methods were recently developed. Among the former are prose, the writing of specifications and documentation in English paragraphs, and conventional flowcharting. The latter includes N-S charts, pseudo-code, and HIPO Charts. None of these methods is bad in itself; for a particular use, one is often better than another.

For example, a program design at a high level, such as a functional specification, may lend itself to prose and to HIPO Charts. Yet, neither of these is much good for detailed logic specifications; prose is often ambiguous and seldom possible to use for coding. HIPO Charts have no facilities for structuring program logic and are also very difficult to use in coding.

Flowcharting has been the method to get from prose or HIPO Charts to code; however, flowcharting is quickly giving way to pseudo-code and Nassi-Shneiderman charts. The latter items have structuring ability built into the technique, and both can be easily translated directly into code. Pseudo-code has the advantage of depicting graphically the logic and also clearly and visually identifying processes within compound conditionals,

For education of users and for walkthroughs or reviews, a combination of HIPO Charts for input/function/output and N-S charts for logical flow has proven extremely useful. Flowcharts and pseudo-code are too strongly programming-oriented for use by nonprogrammers. Pseudo-code might also be useful for code inspections particularly if coded into the programs as comments.

Program documentation has traditionally been separate from programs. One of the hoped-for benefits of structured code was self-documenting programs, To some extent, this benefit has been realized; yet, in many cases supplementary documentation is required. Pseudo-code provides one excellent way of including this supplementary explanation of code within the program as comments. Modification of documentation then requires exactly the same mechanism as modification of the code, and as a result, it aids in maintaining the documentation at a current level.

However, if external program documentation is required, a graphical representation of the code (something impossible to code into the program) can be significantly better. The success of HIPO Charts has demonstrated this fact for overview and function documentation. For displaying logic, N-S charts are much better than HIPO Charts and far better than flowcharts and prose.

SUMMARY

Nassi-Shneiderman Charts have proven to be useful in nearly all phases of program development from early design through walk-throughs, coding, testing, and user education. An excellent graphic technique, the N-S charts provide a simple, yet elegant language that, intentionally, is compatible with structured programming goals and methods. As Nassi and Shneiderman wrote,

> "Programmers who first learn to design programs with these symbols never develop the bad habits which other flowchart notation systems permit.. Since no more than fifteen or twenty symbols can be drawn on a single sheet of paper, the programmer must modularize his program into meaningful sections. The temptation to use off-page connectors, which lead only to confusion, is eliminated. Finally, the ease with which a structured flowchart can be translated into a structured flowchart can be translated into a structured program is pleasantly surprlslng . "[1]

Because Nassi-Shneiderman charts are only now becoming known, the method has not been fully exploited. There is potential in many areas for expanding on the usage of such structured charts and if the logical thinking we are now insisting on in programming can be spread to other disciplines, we cannot lose.

BIBLIOGRAPHY

1. I. Nassi and B. Shneiderman, "Flowchart Techniques for Structured Programming," Notices of the ACM, v. 8, n. 8, 12-26 (August 1973).