Experimental Investigations of the Utility of Detailed Flowcharts in Programming

Ben Shneiderman, Richard Mayer, Don McKay, and Peter Heller
Indiana University

This paper describes previous research on flowcharts and a series of controlled experiments to test the utility of detailed flowcharts as an aid to program composition, comprehension, debugging, and modification. No statistically significant difference between flowchart and nonflowchart groups has been shown, thereby calling into question the utility of detailed flowcharting. A program of further research is suggested.

Key Words and Phrases: flowcharts, program composition, program comprehension, debugging, modification, experimental testing, human factors
CR Categories: 1.5, 4.0

Authors' present addresses: B. Shneiderman, Department of Information Systems Management, University of Maryland, College Park, MD 20742; D. McKay and P. Heller, Computer Science Department, Indiana University, Bloomington, Indiana 74701; R. Mayer, Department of Psychology, University of California, Santa Barbara CA 93106.

The experimental materials can be obtained from B. Shneiderman.

Introduction

Flowcharts have been a part of computer programming since the introduction of computers in the 1940s. In 1947 Goldstein and von Neumann [7] presented a system of describing processes using operation, assertion, and alternative boxes. They felt that "coding begins with the drawing of flow diagram." Prior to coding, the algorithm had been identified and understood. The flowchart represented a high level definition of the solution to be implemented on a machine. Although they were working only with numerical algorithms, they proposed a programming methodology which has since become standard practice in the computer programming field.

Approximately a dozen texts are entirely dedicated to teaching flowcharting. Farina, in his book *Flowcharting* [6], expresses the opinion that flowcharting is an art requiring practice and that a flowchart should be developed before a program is coded. This opinion is practiced in many professional and educational institutions.

In *Flowcharting Techniques* [3], Bohl holds that flowcharting helps "distinguish between the procedure a computer program is written to express and the syntactical details of the language in which the program is written." She feels the flowchart is "an essential tool in problem solving" and states, "The person who cannot flowchart cannot anticipate a problem, analyze the problem, plan the solution, or solve the problem."

The acceptance of flowcharts has been so widespread that a national standard was proposed in 1963. However, the development of more powerful programming language features necessitated revisions in the original flowcharting schemas. For example, the Fortran DO statement has caused many textbook authors and programmers to create their own set of conventions for this construct. Other flowcharting schemes have

developed, including the "structured flowchart" [12] and its variants [5].

Programming language texts reflect differing opinions about flowcharting. An examination of 45 Fortran texts showed that 14 of them employed flowcharts extensively and 19 texts used them occasionally. The remaining 12 used no flowcharts. Another teaching philosophy is seen in introductory "computer science" texts. Being language-independent they teach only the principles of programming. In these texts flowcharts are the main vehicle with which ideas are expressed. This nonsyntactic approach suggests that flowcharting and programming can be separable independent tasks.

The introduction of computer-drawn flowcharts produced from *completed* programs was intended to greatly aid a future programmer's comprehension of the program when attempting debugging or modification. However, the usefulness of such ex post facto flowcharts is hotly disputed. In *The Program Development Process* [2], Aron maintains that such flowcharts are useless to a programmer when diagnosing errors. In such cases "the most helpful data is the program listing itself." Concurring with this opinion is Weinberg [19], who states, "We find no evidence that the original coding plus flow diagrams is any easier to understand than the original coding itself—except to the original programmer."

Brooks [4] is especially vehement in his criticism of the flowchart as documentation, referring to it as "a curse," "a space-hogging exercise in drafting" and "a most thoroughly oversold piece of program documentation." Ledgard and Chmura [9] believe that "program flowcharts can easily suppress much useful information in favor of highlighting sequential control flow, something which distracts the programmer from the important functional relationship in the overall design."

Resolving the question of whether or not flowcharts are an aid in programming could significantly affect the manner in which programming is taught, documented and practiced.

Previous Experimental Research

In a series of experiments with naive programmers, Mayer [11] demonstrated that providing a conceptual hardware model of a computer and its operations during instruction aided performance on postinstructional test items which measured comprehension of programs. Other subjects who utilized a flowchart or a flowchart with the model performed well on test items which required program composition but not as well as the model group on "comprehension" items. This experiment indicated that the use of flowcharts may assist program composition but may hinder learning and performance on comprehension tasks.

Flowcharts are not specific to computer programming. Considerable research and investigation of flowcharts as a tool for communication has been conducted in the human factors field. Lewis, Horabin, and Gane [10] discussed the utility of an "algorithmic approach" to official rules and regulations. In most of their examples, the "algorithm" for a process was presented as a flowchart which they claimed was less susceptible to misinterpretation and less time consuming to execute than a prose description.

Kammann [8] displayed remarkable results for a telephone dialing task that required reference to a set of instructions describing options for the use of internal codes, tie-line codes, area codes, etc. Housewives and Bell Telephone Laboratory professionals committed fewer errors and the housewives expended less time when using a flowchart diagramming the dialing process than when supplied with the normal prose description from a phone book. Kammann gave the following reasons for advocating the use of flowcharts for this type of problem:

—They move major decision criteria forward in the information sequence
—They reduce the complexity of the prose needed to describe the contingent relations
—They distinguish more clearly between relevant and irrelevant information for a given problem
—They reduce the amount of actual information to a reasonable load.

Wright and Reid [20] reported similar success with flowcharts over prose descriptions for an unfamiliar "algorithm" to chose among several hypothetical mechanisms for space travel. Decisions about costs, time, and distance were encoded into a flowchart. When subjects were required to solve the space travel problems from memory, performance for a flowchart group decayed over time while the performance for a prose group increased. Wright and Reid suggested that the flowchart representation of the "algorithm" was encoded visually and subject to "becoming less distinct over successive trials." The prose was encoded verbally and it was possible for the subjects to continue memorizing the material after it was removed [13, 18].

This human factors research, which produced interesting results for flowcharts, is not directly applicable to computer programming. First, programs are not expressed in vague prose but are algorithms represented in a precisely defined language. Second, previous research only measured performance on simple comprehension tasks, ignoring composition, debugging and modification. Third, most of the flowcharts used were limited to one page. Finally, subjects were not given redundant information; they received either the flowchart or the prose, not both.

The basic tasks of programming have been delineated as program composition, comprehension, debugging, and modification [16, 19]. For these tasks, detailed flowcharts are often advocated as a useful pictorial representation of the program logic or flow of

control. Our goal was to determine the utility of detailed flowcharts in these computer programming tasks.

Experiment I (Composition)

This first experiment was designed to study how the creation of a detailed flowchart assisted the subjects in composing a program. Much of the literature on flowcharting claims that it is most helpful as a program design aid which clarifies the problem to the programmer.

Method

Subjects. The subjects were students in an introductory computer programming course using Fortran. The textbook used flowcharts to illustrate program development and flowcharts were used by the instructor. The experiment was conducted by including the materials as part of the second of three in-class examinations which constituted the major part of the course grade.

Procedure and materials. Thirty-four subjects (flowchart group), received test instructions which indicated that they were to write a flowchart and then a program for a given problem. The flowchart counted for 15 points, the program 25 points. Twenty-eight subjects (nonflowchart group) were instructed to merely write the program, which counted for the full 40 points. The subjects were given as much time as they wanted to complete the test. The grading was done by a graduate student with much experience in grading programs and in consulting with students about programs. The results were returned to the students at the next meeting of the class.

Results

The scores on the program composition task were normalized to 100 percent. The flowchart group mean score was 94 while the nonflowchart group mean was 95. A *t*-test showed no significant difference between the two groups. The flowchart group had a mean score of 13.1 on the flowchart or 87.3 out of 100.

Discussion

The requirement to produce a flowchart seemed to have no benefit or harm on the subjects' ability to prepare a program described by that flowchart. This was in spite of the fact that the problem had been chosen to favor the flowchart group by having a relatively complicated branching pattern. The relatively good scores indicate that all the subjects found the task to be straightforward.

An earlier pilot study with the same design and procedures had produced similar results. In the pilot flowchart group two subjects (9 percent) achieved perfect scores and 14 (61 percent) out of 23 subjects scored 50 percent or above. In the pilot nonflowchart group four subjects (15 percent) achieved a perfect score and 17 (65 percent) out of 26 subjects scored 50 percent or above. The pilot study was run earlier in the term with a simpler problem, but one which the subjects found more challenging, given their shallower background.

Experiment II (Comprehension)

A second often cited beneficial aspect of detailed flowcharts is as an aid to comprehension of programs. Flowchart proponents argue that if a detailed flowchart is studied in conjunction with a program, comprehension can be improved. This experiment was designed to test this hypothesis. Since experiment I had not shown flowcharts to be useful for composition in a simple program with novices, this experiment drew on more complex program structures, but still with novice programmers as subjects.

Method

Subjects. The subjects were again students in an introductory computer programming course using Fortran and similar to the previous subjects. The experiment was conducted by including materials as part of the third of five in-class examinations which constituted the major part of the course grade.

Procedure and materials. Sixty of the 100 points on the examination were related to the experiment. Two forms of the examination were prepared. The first form contained two programs (27 and 24 Fortran statements) with a flowchart for the first program only, while the second form contained the same two programs but a flowchart for the second program only. Twenty-five subjects received the first form, 28 received the second form. The comprehension questions, which were the same on both forms of the test, required the subjects to determine the values printed for various inputs and to trace the flow of execution. The subjects were given as much time as they needed to complete the test. Grading was simplified since the answers were clearly correct or not correct. The results were returned to the students at the next class meeting.

Results

An analysis of variance for the scores indicated that the only significant result (at the 0.05 level) was that problem 2 was more difficult than problem 1. Subjects who had flowcharts did not perform differently than those who did not have flowcharts. The mean percent correct for each group and problem appear as Table I (all tables appear on pp. 378–379).

Discussion

The availability of a flowchart neither benefited nor harmed the subjects in the comprehension task. This

was surprising since the programs had been designed with a large number of transfers of control; precisely the situation which is purported to be most advantageous to the flowchart groups. Information observation during the exam showed that most subjects were rarely referring to the available flowchart but preferred to study the program directly.

Experiment III (Comprehension and Debugging)

This experiment, which used subjects from an intermediate experience level, measured the effect of flowcharts in debugging and comprehension of programs.

Method

Subjects. The subjects selected were basically from one programming experience level. They were all in intermediate Fortran programming courses but were divided into two groups based on the flowcharting environment they were learning in. The NFC group consisted of 43 subjects who had had some previous instruction about flowcharts and presently were using a book employing flowcharts but were under no obligation to use or turn in flowcharts with their assigned programs. The FC group consisted of 27 subjects who were required to write a flowchart before doing the actual programming and to hand it in with their programs. The data from each group was analyzed separately and later all data was combined and analyzed. All subjects knew the experiment would have no bearing on their course grade.

Design. The experimental design consisted of a control subgroup and two experimental subgroups for both NFC and FC. The members of each subgroup were randomly selected, each subgroup containing approximately equal numbers of subjects. The control subgroups received no flowchart. The first experimental subgroups received a detailed four-page (micro) flowchart while the second experimental subgroups received a more abstract single-page (macro) flowchart.

Materials. All the subjects received the same compiled listing (with line numbers) and its corresponding output. Depending upon which subgroup (experimental factor) they were being tested in, a subject would also receive either a micro flowchart, a macro flowchart, or no flowchart at all. The program used was a moderately difficult tic-tac-toe playing program written in Fortran. The program consisted of an 81 line main program with 43 and 23 line subroutines. The main program was commented and had several DO loops as well as a controlling IF loop. The program had three nonsyntactic bugs placed in it. It produced incorrect output which only reflected one of these bugs. Both the program and the flowcharts were taken from Sturgul and Merchant [17].

Procedures. The experiment took the form of a three-part test. In the first part, once the subjects had received a program listing and one of the three flowchart possibilities each subject was given a combination instruction/answer sheet. In the instructions subjects were told:

(1) They had been given a listing of a tic-tac-toe playing program which was not supposed to lose.
(2) The program had at least one bug in it which was apparent in the output.

Next they were instructed to:

(1) Study the listing and output in conjunction with any flowchart they had received (or on its own if one was received).
(2) Find the bug(s) and explain how to repair them. (Bugs were reported by referring to the line number and writing the necessary revision.)

For this task the NFC subjects were given 40 minutes and the FC subjects were allowed 50 minutes after which the answer sheets were collected. (The subjects retained the listings and flowcharts.) After a short break the second part of the test was administered.

In the second part the subjects were first informed of the bugs and told to make the corrections in their listings. Next they were instructed to answer 11 multiple-choice questions for which the NFC and FC group subjects were allotted 20 and 30 minutes, respectively. The questions which were designed to measure program comprehension tested basic programming knowledge, hand simulation of the (corrected) program, and other problems necessitating knowledge of the program. (The subjects still had their listings and flowcharts for references while answering the questions.)

Finally, the subjects were asked to respond to questions concerning their feelings about how well they had done in a questionnaire. In each of the questionnaire questions the subjects responded on a scale of 0 to 9. All subjects were asked how well they thought they understood the program. Any subject who had received a flowchart was further asked to respond to questions concerning the usefulness of the flowchart in their understanding of the program and in finding the bugs.

Results

The data gathered from the two separate subject groups is shown in Table II. The two groups were from entirely separate subject pools tested under different conditions and therefore the magnitude of the scores cannot be compared between groups. However, the trends within each group can be compared.

The NFC data (those who do not normally use flowcharts) reflects the subjects' background: the subjects not given flowcharts had the highest mean scores for both the debugging and comprehension tests, micro next, and macro last. The FC data (those who do use flowcharts) reflects a quite different orientation. The subjects given micro flowcharts had the highest average

scores for both tests in the FC group, macro subjects had the next best scores, and the nonflowchart subjects came in last.

This would seem to indicate a correlation between a person's background with flowcharts and their usefulness in programming. However, an analysis of variance showed *none* of the three combinations of experimental conditions (none, micro, or macro) for the three test results (debug score, comprehension score, and total) for both NFC and FC to be statistically significant even at the 10 percent level.

As stated before, one bug caused an obvious error in the output and this bug was the most easily found of the three. Nearly 70 percent of the subjects found and corrected it. In contrast, 12 percent of the subjects found two bugs and only three subjects out of 70 found all three bugs. Of these three, two also answered all the comprehension questions correctly. This suggests that if one successfully debugs a program one must have a thorough comprehension of it. This relationship holds for those finding two bugs but breaks down for those finding only one or no bugs.

The comprehension test results parallel the debugging test for both NFC and FC. For both groups the flowchart subgroup which had the highest average debug score also had the highest mean for comprehension. An item analysis of the individual comprehension questions was inconclusive and no important results could be drawn from it. The subjects perceived very accurately how well they understood the program. To show this, the subjects were grouped by the number of questions they answered correctly. The average questionnaire response to how well they thought they understood the program was obtained for each level of "correctness." At each increment in the number of correctly answered questions the average response would generally increase suggesting that the subjects were in tune with their performances.

The questionnaire responses yielded an unexpected result as to how useful subjects found the flowchart in debugging. For both the macro and micro flowchart subjects in both subject groups those who found only one bug (of three) felt the flowchart helped most in debugging. Those who found zero, two, or all three bugs rated the flowchart's usefulness at a lower level. Perhaps this was because the one bug which manifested itself in the output was traceable through the flowchart, while the less obvious ones had to be found by hand simulation using the listing itself.

Discussion

These results are strong evidence in favor of the contention that flowcharts do not aid a programmer in comprehending or debugging a program. Depending on the programmer's background, flowcharts may help or hinder performance, but the results show any difference is not at a significant level, that is, could have been produced by chance.

The trend of the scores in favor of flowcharts for those who use them (FC group) and away for those who do not (NFC group), was expected. The result that the differences were not significant warrants further experimentation (see conclusion).

The questionnaire results give insights into the subjects' thoughts about the flowcharts. Somewhat surprisingly, the answers given by both NFC and FC subjects are nearly identical in all respects. For both there was a pronounced positive correlation between the number of questions answered correctly and how well the subjects thought they understood the program in every experimental condition.

The detailed micro flowchart listed nearly all the statements from the program (as they should have been), so anyone who had directly compared the listing and flowchart would have found the flowchart very useful in debugging. But even the FC group rated the micro flowchart unprofitable if they found two or more bugs.

Experiment IV (Modification)

The focus of this experiment was the use of flowcharts as a program documentation tool. Since it is claimed flowcharts describe the logic of a program, the task of discovering the placement of a modification should be easier when the programmer has a flowchart of the program. This experiment compared the performance of intermediate programmers in a modification task in which some subjects received flowcharts and others did not. Two levels of flowcharts were used: a detailed, statement by statement micro flowchart and a higher level macro flowchart.

Method

Subject. The subjects for this experiment were students of a second semester programming course at Indiana University and Purdue University. The experiment was conducted during a regularly scheduled class meeting. Subjects from Indiana University were 33 students who were not required to design flowcharts as part of their assignments. These subjects were exposed to flowcharts by the textbook used for the course. The 37 subjects from Purdue University who participated in the experiment were required to turn in flowcharts with their programming assignments.

Materials. The booklet given to each subject contained a set of instructions, a Fortran program, a loader map, sample output, a set of three modification descriptions, and a biographical questionnaire. The 75-line Fortran program produced semester grade reports for a fictitious college. The program had 48 lines of Fortran code and 27 lines of comments of which 23 appeared as a comment block at the beginning of the program. The output from the program was a set of

semester grade reports for all student records input. There were three separate program listings for the subjects to indicate their modifications.

In addition to the booklet, one-third of the subjects received a one-page macro flowchart, another third were supplied with a three-page micro flowchart, and a remainder were given no flowchart.

Procedures. The subjects were instructed to peruse the instructions, program, and flowchart for approximately five minutes. Each subject was asked to make the modifications to the existing program according to the modification description provided. A total of three modifications were to be attempted in the 45-minute period allowed for the experiment. Subjects were told their modifications would be graded on correctness and runability. At Indiana University, each subject was timed individually by the experimenters for each modification. At Purdue, each subject was responsible for recording the amount of time spent on each modification.

Results

To simplify the analysis, the data obtained from 12 subjects was not included in the following analysis. Of these subjects, five did not finish and seven had GPAs of 2.5 or below on a 4.0 scale, where 4.0 is an "A." The results reflect the data from 60 subjects, 30 from each university, and ten in each of the six groups. Also, most subjects did not finish modification III and it was not included in these results. At Indiana University, no one finished modification III; six subjects from the micro flowchart group, eight subjects from the macro group, and seven subjects from the nonflowchart group started modification III. In the Purdue groups, one subject from each of the groups finished modification III. Of those who did not finish, seven from each of the groups had started the problem.

Modifications I and II were graded by an experienced Associate Instructor who assigned a correctness percentage score and kept track of error types such as:

—Incorrect formatting of output
—Incorrect placement of modification, within the existing program
—Violation of the modification description
—Violation of Fortran syntax
—Errors of omission.

The mean percent correct for each group is displayed in Table III. An analysis of variance for the scores showed that the University factor (Indiana vs. Purdue), the modification factor (modification I vs modification II), and the interaction of University and modification factors were all significant. These results indicate that the Purdue groups made fewer errors, modification II was more difficult than modification I, and the Purdue groups performed better than the Indiana groups on both modifications.

An analysis of the types of errors made showed that the second type of error listed, "incorrect placement of modification within the existing program," was as frequent in each of the six groups. The type of flowchart aid—none, macro, or micro—was not a significant factor in these results.

An analysis of variance for the time measure indicated that neither the type of flowchart nor any of the possible interactions was significant. The mean times are displayed in Table IV. While the means reflect that some groups did perform better than others, the variance in these groups was extremely large.

Flowchart Utility Data

Table I. Mean Percent Correct for Experiment II.

	Problem	
	1	2
Flowchart on 1	94.4	89.6
Flowchart on 2	97.0	94.4

Table II. Results for Experiment III.

Mean Percent Correct (Comprehension)

	Flowchart used		
	none	macro	micro
group NFC	52	34	46
FC	53	55	76

Mean Percent Correct (Debugging)

	Flowchart used		
	none	macro	micro
group NFC	12	11	4
FC	29	26	45

Mean Percent Correct (Total Score)

	Flowchart used		
	none	macro	micro
group NFC	34	23	27
FC	42	42	62

Discussion

The results from this experiment showed no advantage for groups receiving flowcharts in a program modification task for either time or percent correct measures. An analysis of the types of errors made indicated

Flowchart Utility Data (continued)

Table III. Mean Percent Correct for Experiment IV.

Univ.	Flowchart aid	Mod I	Mod II
Purdue	none	73	85
	macro	88	77
	micro	87	81
Indiana	none	77	64
	macro	77	71
	micro	77	59

Table IV. Mean Time per Modification in Minutes.

Univ.	Flowchart aid	Mod I	Mod II
Purdue	none	15.5	15.3
	macro	16.1	14.2
	micro	14.0	14.1
Indiana	none	15.6	16.7
	macro	15.4	13.9
	micro	16.0	17.9

Table V. Mean Percent Correct for Experiment V.

	Type of question	
Group	execution	interpretation
flowchart	48.5	51.2
program plus flowchart	56.9	50.0
program	57.8	62.4

that most were errors which flowcharts may not decrease, such as coding errors and errors of omission. After discarding these types of errors, no significant difference can be reported between the flowchart and nonflowchart groups for incorrect positioning of the modification within the existing program. These results contrast with the human factors research in prose and "algorithmic" representations of rules and regulations in which flowcharts have been shown to aid comprehension. As stated previously, there were several task differences between the present experiment and the human factors research.

Part of this experiment was designed to test the utility of detailed micro flowcharts in a program modification task. The intermediate programmers who were given both the program and the micro flowchart of the program did not perform better in the modification task than those who received only the program. This indicated that a micro flowchart and a program do provide equivalent information for a modification task and that the micro flowchart does not provide any additional information.

Another part of this experiment dealt with higher level or macro flowcharts. Again, the subjects who were given both a macro flowchart and a program did not perform differently from those who were supplied with just the program.

The critical factor in program modification might be understanding the existing program at a macro level, which allows identification of where to incorporate the modification and what effects the proposed modification will have on the remainder of the program. A similar statement could be made for program debugging. The macro flowchart or at least some general presentation of the program seems intuitively useful. Since this experiment did not show any significant results for the macro flowchart group, the question of the utility of macro flowcharts as a vehicle of expression remains confused. Perhaps even the macro flowchart is equivalent to the program it describes in the same manner as the micro flowchart and thus provides no additional information.

For both macro and micro flowchart groups, their failure to outperform the other group could be attributed to several experimental variables. It is possible that a replication of the current experiment with a significantly longer and more complex program may show different results, especially for the macro flowchart since it is viewed as an organizer which facilitates the "chunking" of the program into logical modules. Results for the micro flowchart group in such an experiment would not be expected to differ from the nonflowchart group, except for a task which required hand simulation and little overall understanding. Second, the topic of the program might have been familiar to both groups of college students; however, the Purdue groups were possibly not as familiar with the grading system since it was modeled after the Indiana University system. Finally, the program had a comment block at the beginning which explained the general workings of the program and this may have been enough information on which to base a modification.

Experiment V (Comprehension)

The previous experiments reported here have not shown what information about an algorithm can be obtained from a flowchart representation. Experiment V investigated the information content of detailed flowcharts in a comprehension task. Two types of comprehension items were presented: hand simulation (low level) problems and interpretation (higher level) problems.

Method

Subjects. The 58 subjects for this experiment were students of an eight-week summer session introductory course in computer science at Indiana University. The experiment was conducted as the third quiz of the term at the beginning of the sixth week of the course. Subjects were familiar with Fortran and flowcharts.

Materials. Each subject received a quiz booklet. Three different booklets were used. Each contained an algorithm and a set of questions about the algorithm. The algorithm used was a two-way array merging algorithm. The subjects first exposure to the algorithm was the quiz. One group of subjects received a 23-line Fortran program of the algorithm, another group was given a one-page detailed flowchart of the algorithm, and the last group was provided with both the detailed flowchart and the Fortran program. The quiz questions were of two types: hand simulation problems and interpretation items. The two hand simulation items asked for the algorithm's output given a set of inputs values. The three interpretation questions were concerned with the operation and properties of the algorithm.

Procedures. Each subject received a quiz booklet and was given general instructions for the quiz. The subjects were given 25 minutes for the quiz.

Results

The subjects' quizzes were graded on a 100-point scale with each problem worth 20 points. Seven subjects were not included in the results; three dropped the course and four had class averages below "D" level. The mean percent correct for each group and the type of question appear in Table V. An analysis of variance indicated that all groups performed equally well. The means for each group suggested that the group which had only the program performed the best. Again, the variance within each group was large and the difference were not statistically different.

Discussion

The results of this experiment indicated, as argued in experiment IV, that the type of information obtained from a detailed flowchart and a program appear to be equivalent. The most interesting result is the comparison of the program group versus the other two groups for the two types of questions. For the execution items the program group scored the highest and the flowchart group scored the lowest. This result, although not statistically significant, suggests that a program may be more understandable than a flowchart for hand simulation type problems. For the interpretation items, again the program group performed the best, suggesting the program representation may be supplying additional information which a flowchart cannot. The performance of the program group versus the program plus flowchart group suggests that the presentation of redundant information may hinder understanding necessary for a general understanding of an algorithm. The results of this experiment support the work of Mayer [11], who demonstrated different learning outcomes for flowchart groups and that of Wright and Reid [20], who showed different results for flowchart and prose descriptions that were memorized by subjects.

Conclusions

Although our original intention was to ascertain under which conditions detailed flowcharts were most helpful, our repeated negative results have led us to a more skeptical opinion of the utility of detailed flowcharts under modern programming conditions. We repeatedly selected problems and tried to create test conditions which would favor the flowchart groups, but found no statistically significant differences between the flowchart and nonflowchart groups. In some cases the mean scores for the nonflowchart groups even surpassed the means for the flowchart groups.

We conjecture that detailed flowcharts are merely a redundant presentation of the information contained in the programming language statements. The flowcharts may even be at a disadvantage because they are not as complete (omitting declarations, statement labels, and input/output formats) and require many more pages than do the concise programming language statements.

The advent of top-down design techniques and structured control structures may reduce the utility of detailed flowcharts, since programmers are now learning to use higher level concepts than those represented by standard detailed flowcharts. This suggests that further experiments might be done with macro flowcharts and structured flowcharts [12].

Our results should be replicated with a wide variety of programmers and problems. Especially valuable are experiments with professional programmers which put flowcharting and programming in competition. One such experiment would be to require one group of programmers to flowchart a program and another group to write the program without a flowchart. Next, those who had written flowcharts would be given time to code up their programs. Both groups would be timed and compared on how quickly they could get their programs running. This experiment would reveal whether flowcharts aid or delay program development. Also important are studies on large complex programs.

Another important research direction would be to study professional programmers who feel that flowcharts are essential in their work. Is their dedication to this technique well-founded or would their time and energies be better spent in more careful program design or documentation? It has also been suggested that detailed flowcharts are more meaningful than programming language statements for managers and nonprogrammers. This possibility should be investigated.

Experiments focusing on the use of macro flow-

charts with substantially larger programs would be important to the commercial programming industry. Comprehension, debugging, or modification questions could be given to two groups of subjects, only one of which was given macro flowcharts. We conjecture that macro flowcharts may alleviate anxiety and confusion when subjects are given large programs (at least 1000 lines of code). Subjects with macro flowcharts may waste less time in trying to locate particular sections of code.

Finally, a deeper understanding of the cognitive processes used in programming may be developed as a result of these and other human factors experiments. Such an understanding may lead to an explanation of why flowcharts are or are not helpful in specific situations. The internal semantic structure concept developed by Shneiderman [15] suggests that since the detailed flowchart may be merely an alternative representation of the syntax of a program, it should not be helpful to programmers familiar with a programming language. Having a French recipe in addition to an English version of the same recipe would not be helpful to a cook knowledgeable in both languages. The syntactic/semantic model of programmer behavior offered by Shneiderman and Mayer [16] provides a broader theoretical framework, but more experiments must be done to validate the model and make it viable as a predictive tool.

In summary, our experiments have not demonstrated the utility of detailed flowcharts in program composition, comprehension, debugging, or modification. Further work is necessary to replicate the results, especially with professionals, to explore other areas where flowcharts may be helpful, to study other forms of flowcharting and to contribute to a more thorough understanding of the cognitive skills required in computer programming.

Acknowledgments. We would like to express our appreciation to Carl Landwehr of Purdue University for his assistance in running some of the experiments on his students. We are indebted to the many students at Indiana and Purdue Universities for their participation as subjects in these experiments. The detailed comments of the referees have greatly improved the presentation.

References

1. American Standards Institute. Proposed American standard flowchart symbols for information processing. *Comm. ACM 6,* 10 (1963), 601–604.
2. Aron, J. *The Program Development Process; The Individual Programmer.* Addison-Wesley, Reading, Mass., 1974, pp. 104–106.
3. Bohl, M. *Flowcharting Techniques.* Science Research Associates, Chicago, 1971, p. 53.
4. Brooks, F.P. *The Mythical Man-Month.* Addison-Wesley, Reading, Mass., 1975.
5. Chapin, N. New format for flowcharts. *Software: Practice and Experience 4,* 4 (1974), 341–357.
6. Farina, F. *Flowcharting.* Prentice-Hall, Englewood Cliffs, N.J., 1970, iii.
7. Goldstein, H.H., and von Neumann, J. Planning and coding problems for an electronic computing instrument, part II, vol I. Rep. prepared for the U.S. Army Ordinance Dept., 1947. Reprinted in von Neumann, J. *Collected Works, Vol. V,* A.H. Taub, Ed., McMillan, New York, pp. 80–151.
8. Kammann, R. The comprehensibility of printed instructions and the flowchart alternative. *Human Factors 17,* 2 (1975), 183–191.
9. Ledgard, H., and Chmura, L. *COBOL with Style.* Hayden, Rochelle Park, N.J., 1976.
10. Lewis, B.N., Horabin, I.S., and Gane, C.P. *Flowcharts, Logical Trees and Algorithms for Rules and Regulations.* Her Majesty's Stationary Office, London, 1967.
11. Mayer, R.E. Different problem-solving competencies established in learning computer programming with and without meaningful models. *J. Educ. Psych. 67,* 6 (1975), 725–734.
12. Nassi, I., and Shneiderman, B. Flowcharting techniques for structured programming. SIGPLAN Notices (ACM) *8,* 8 (1973), 12–26.
13. Shiffrin, R.M. Memory search. In *Models of Human Memory,* D.A. Norman, Ed., Academic Press, New York, 1970.
14. Shneiderman, B. Experimental testing in programming languages, stylistic considerations and design techniques. Proc. AFIPS NCC, 1975 AFIPS Press, Montvale, N.J., 1975, pp. 653–656.
15. Shneiderman, B. Exploratory experiments in programmer behavior. *Int. J. Comptr. and Inform. Sci. 5,* 2 (June 1976), 123–143.
16. Shneiderman, B., and Mayer, R. Syntactic/semantic interactions in programmer behavior: a model and experimental results (unpublished).
17. Sturgul, J.R., and Merchant, M.J. *Applied Fortran IV Programming.* Wadsworth, Belmont, Calif., 1973.
18. Tulving, E. Subjective organization in free recall of "unrelated" words. *Psych. Rev. 69* (1962), 344–354.
19. Weinberg, G.M. *The Psychology of Computer Programming.* Van Nostrand, Princeton, N.J., 1971.
20. Wright, P., and Reid, F. Written information: some alternatives to prose for expressing the outcomes of complex contingencies. *J. Appl. Psych. 57,* 2 (1973), 160–166.