Factors affecting computer program comprehension

John Peter Boysen
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd
Part of the Computer Sciences Commons

Recommended Citation
Boysen, John Peter, "Factors affecting computer program comprehension " (1979). Retrospective Theses and Dissertations. 7194. https://lib.dr.iastate.edu/rtd/7194
INFORMATION TO USERS

This was produced from a copy of a document sent to us for microfilming. While the most advanced technological means to photograph and reproduce this document have been used, the quality is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help you understand markings or notations which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure you of complete continuity.

2. When an image on the film is obliterated with a round black mark it is an indication that the film inspector noticed either blurred copy because of movement during exposure, or duplicate copy. Unless we meant to delete copyrighted materials that should not have been filmed, you will find a good image of the page in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed the photographer has followed a definite method in "sectioning" the material. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For any illustrations that cannot be reproduced satisfactorily by xerography, photographic prints can be purchased at additional cost and tipped into your xerographic copy. Requests can be made to our Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases we have filmed the best available copy.

University Microfilms International
300 N. ZEEB ROAD, ANN ARBOR, M. 48106
18 BEDFORD ROW, LONDON WC1R 4EJ, ENGLAND
BOYSEN, JOHN PETER
FACTORS AFFECTING COMPUTER PROGRAM COMPREHENSION.

IOWA STATE UNIVERSITY, PH.D., 1979
PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark.

1. Glossy photographs
2. Colored illustrations
3. Photographs with dark background
4. Illustrations are poor copy
5. Print shows through as there is text on both sides of page
6. Indistinct, broken or small print on several pages throughout
7. Tightly bound copy with print lost in spine
8. Computer printout pages with indistinct print
9. Page(s) lacking when material received, and not available from school or author
10. Page(s) seem to be missing in numbering only as text follows
11. Poor carbon copy
12. Not original copy, several pages with blurred type
13. Appendix pages are poor copy
14. Original copy with light type
15. Curling and wrinkled pages
16. Other

University Microfilms International
300 N. Zeeb Rd., Ann Arbor, MI 48106 (313) 761-4700
Factors affecting computer program comprehension

by

John Peter Boysen

A Dissertation Submitted to the
Graduate Faculty in Partial Fulfillment of
The Requirements for the Degree of
DOCTOR OF PHILOSOPHY

Major: Computer Science

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa
1979
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>Program Comprehension: A Programming Activity</td>
<td>2</td>
</tr>
<tr>
<td>Program Notation: A Factor Affecting Program Comprehension</td>
<td>3</td>
</tr>
<tr>
<td>Measures of Program Comprehension</td>
<td>4</td>
</tr>
<tr>
<td>The Use of Reaction Time in Comprehension Measurement</td>
<td>7</td>
</tr>
<tr>
<td>Statement of Problem</td>
<td>9</td>
</tr>
<tr>
<td>Dissertation Organization</td>
<td>10</td>
</tr>
<tr>
<td>REVIEW OF LITERATURE</td>
<td>11</td>
</tr>
<tr>
<td>Measures of Program Comprehension</td>
<td>11</td>
</tr>
<tr>
<td>Factors Affecting Program Comprehension</td>
<td>25</td>
</tr>
<tr>
<td>Psycholinguistic Results</td>
<td>28</td>
</tr>
<tr>
<td>Summary</td>
<td>34</td>
</tr>
<tr>
<td>THE PILOT STUDY</td>
<td>36</td>
</tr>
<tr>
<td>A Proposed Measure of Comprehension</td>
<td>36</td>
</tr>
<tr>
<td>Procedure</td>
<td>41</td>
</tr>
<tr>
<td>Subjects</td>
<td>46</td>
</tr>
<tr>
<td>Experimental Design and Results</td>
<td>47</td>
</tr>
<tr>
<td>Motor Reaction Time Analysis</td>
<td>48</td>
</tr>
<tr>
<td>Simple Expression Analysis</td>
<td>50</td>
</tr>
<tr>
<td>Program-Version Analysis</td>
<td>58</td>
</tr>
<tr>
<td>Summary</td>
<td>67</td>
</tr>
<tr>
<td>THE FINAL STUDY</td>
<td>69</td>
</tr>
<tr>
<td>A Revised Measure of Comprehension</td>
<td>69</td>
</tr>
<tr>
<td>Procedure</td>
<td>72</td>
</tr>
<tr>
<td>Subjects</td>
<td>75</td>
</tr>
<tr>
<td>Experimental Design and Results</td>
<td>75</td>
</tr>
<tr>
<td>Simple Expression Analysis</td>
<td>75</td>
</tr>
<tr>
<td>Program-Version Analysis</td>
<td>84</td>
</tr>
<tr>
<td>Summary</td>
<td>97</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>SUMMARY AND CONCLUSIONS</td>
<td>101</td>
</tr>
<tr>
<td>Recommendations for Future Work</td>
<td>103</td>
</tr>
<tr>
<td>Conclusion</td>
<td>105</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>106</td>
</tr>
<tr>
<td>APPENDIX A. STATISTICAL ANALYSIS OF TRUE AND FALSE DATA FOR EXPRESSIONS</td>
<td>107</td>
</tr>
<tr>
<td>APPENDIX B. HUMAN SUBJECTS FORM</td>
<td>114</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>116</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Operators and Operands in Versions of Minimum Program</td>
<td>19</td>
</tr>
<tr>
<td>2.</td>
<td>Software Science Parameters for Versions of Minimum Program</td>
<td>20</td>
</tr>
<tr>
<td>3.</td>
<td>The Operators and Expressions in the Pilot Study</td>
<td>45</td>
</tr>
<tr>
<td>4.</td>
<td>Background of Students in Pilot Study</td>
<td>47</td>
</tr>
<tr>
<td>5.</td>
<td>Statistical Analysis for Keys</td>
<td>49</td>
</tr>
<tr>
<td>6.</td>
<td>Statistical Analysis for Expressions in Pilot Study</td>
<td>52</td>
</tr>
<tr>
<td>7.</td>
<td>Modified Statistical Analysis for Expressions in Pilot Study</td>
<td>55</td>
</tr>
<tr>
<td>8.</td>
<td>Program-Version Assignments to Blocks</td>
<td>61</td>
</tr>
<tr>
<td>9.</td>
<td>Analysis of Variance Table for Programs in Pilot Study</td>
<td>62</td>
</tr>
<tr>
<td>10.</td>
<td>Subject Background Data for Final Study</td>
<td>76</td>
</tr>
<tr>
<td>11.</td>
<td>Expressions Used in Final Study</td>
<td>77</td>
</tr>
<tr>
<td>12.</td>
<td>Statistical Analysis for LT, GT, AND, OR and NOT</td>
<td>78</td>
</tr>
<tr>
<td>13.</td>
<td>Statistical Analysis for EQ and NE</td>
<td>79</td>
</tr>
<tr>
<td>14.</td>
<td>Statistical Analysis for IF and CASE</td>
<td>80</td>
</tr>
<tr>
<td>15.</td>
<td>Statistical Analysis of Undergraduate Program Data in Final Study</td>
<td>88</td>
</tr>
<tr>
<td>16.</td>
<td>Statistical Analysis of Graduate Program Data in Final Study</td>
<td>89</td>
</tr>
<tr>
<td>17.</td>
<td>Means of Program Versions for Undergraduates</td>
<td>90</td>
</tr>
</tbody>
</table>
18. Summary of Function Misstatements .................................. 90
19. Statistical Analysis of Undergraduate Program Data in Final Study Using only Correct Response Data . 93
20. Statistical Analysis of Graduate Program Data in Final Study Using only Correct Response Data . . 94
21. Means of Program Versions for Undergraduates and Graduates Using only Correct Response Data . . . 95
22. Summary of Cyclomatic and Ec Values for the Program Versions in the Final Study ................. 96
23. Statistical Analysis of True and False Data for EQ 107
24. Statistical Analysis of True and False Data for NE 106
25. Statistical Analysis of True and False Data for AND 109
26. Statistical Analysis of True and False Data for GT 110
27. Statistical Analysis of True and False Data for LT 111
28. Statistical Analysis of True and False Data for NOT 112
29. Statistical Analysis of True and False Data for OR 113
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Three Versions of a program to Compute a Minimum</td>
<td>13</td>
</tr>
<tr>
<td>2.</td>
<td>Vector Representation of Independent Paths</td>
<td>13</td>
</tr>
<tr>
<td>3.</td>
<td>Three Versions of a Maximum Program</td>
<td>37</td>
</tr>
<tr>
<td>4.</td>
<td>An Example Reaction Time Curve</td>
<td>39</td>
</tr>
<tr>
<td>5.</td>
<td>Keyboard Layouts for Subject Input</td>
<td>43</td>
</tr>
<tr>
<td>6.</td>
<td>Trial Means for Expressions in Pilot Study</td>
<td>54</td>
</tr>
<tr>
<td>7.</td>
<td>Versions for Program 0 in the Pilot Study</td>
<td>58</td>
</tr>
<tr>
<td>8.</td>
<td>Versions for Program 1 in Pilot Study</td>
<td>59</td>
</tr>
<tr>
<td>9.</td>
<td>Versions for Program 2 in the Pilot Study</td>
<td>60</td>
</tr>
<tr>
<td>10.</td>
<td>Trial Data for Program-Version Experiment</td>
<td>64</td>
</tr>
<tr>
<td>11.</td>
<td>Reaction Times by Digit for Expressions in Final Study</td>
<td>81</td>
</tr>
<tr>
<td>12.</td>
<td>Program 0: Three Versions to Compute the Minimum</td>
<td>85</td>
</tr>
<tr>
<td>13.</td>
<td>Program 1: Three Versions to Compute the Maximum</td>
<td>86</td>
</tr>
<tr>
<td>14.</td>
<td>Program 2: Three Versions to Merge Two Sorted Arrays</td>
<td>87</td>
</tr>
</tbody>
</table>
INTRODUCTION

If current trends in the development of hardware and software continue, a significant portion of future computing costs will be due to the development and maintenance of software. The cost involved in the development of major system projects provides a good example. Fifteen years ago, the cost of software in major system projects accounted for only ten per cent of the total system cost [2]. However, in a recent survey by Delaney [2], 32 chief programmers estimated that software currently accounts for sixty per cent of the total system cost and this figure is expected to rise to eighty per cent in five to ten years. Therefore, if the costs of computing services are to be controlled, the most likely area of improvement is the software development process.

The software development process also needs to be improved to avoid the serious difficulties caused by program errors. In the past, program errors have resulted in equipment damage or human inconvenience in banking, communications and missile operations [34]. But as our society depends more heavily on computer technology, the risks to
individuals and society become more serious. Air traffic control, nuclear power plant operation and weapon system operations are just a few applications where program errors could have disastrous effects. Therefore, it seems clear that research in programming must be conducted to improve the processes involved in software development. The goal of such research should be the creation of methods and tools which will both improve software quality and reduce software costs.

**Program Comprehension: A Programming Activity**

To conduct research in computer programming, it is first necessary to identify the activities involved. Shneiderman [27] defines four: composition, comprehension, debugging and modification. While composition, debugging and modification are relatively distinct activities, comprehension is essential to all programming activities. The programmer must understand the developing program during the composition process if the program is to ever solve the problem. Likewise, the program which is to be debugged or modified must be understood if the necessary changes are to be correct. Thus, improvements in the comprehension process could have a significant impact on all phases of computer programming.
Improving program comprehension will result in several positive effects. First, the likelihood of making initial errors in the program will be reduced because the developing program will be better understood. Second, better program understanding will mean that the few difficulties and errors that do occur during programming will be identified and corrected more quickly. Thus, the time to compose, debug and modify programs will be shortened, resulting in reduced software costs. Through program comprehension research, the goal of improving the processes involved in programming may be achieved.

Program Notation: A Factor Affecting Program Comprehension

One of the major factors affecting program comprehension is the notation used to describe the program. According to Gries [9], research in program notation is involved in "the development of program representations which foster simpler programming practices and make the resulting programs easier to understand."

Several notational devices which are believed to improve program comprehension have been proposed in the computer science literature and subsequently tested empirically. For example, Love [19] presented subjects with versions of programs which differed in paragraphing style and
complexity of control flow. He found that paragraphing had no significant effect on comprehension but that control flow did. Shneiderman [28] studied the effect of using functionally descriptive comments at the beginning of a program versus one line comments interspersed in the program. He found that the descriptive comments significantly improved comprehension. Weissman [36] has also conducted experiments on the effects of comments, paragraphing, control flow and mnemonic variable names. Unfortunately, his results were inconclusive because of his experimental procedures.

It is interesting to note that the studies of paragraphing, commenting and control flow have all dealt with the global effects of these factors on program comprehension. Yet if we are to understand how humans process programs we should begin by looking at how they process expressions and simple statements. In the next section, a comprehension measure will be suggested which may make the evaluation of simple statements and expressions possible.

Measures of Program Comprehension

Although the literature is replete with articles proclaiming the merits or demerits of notational devices such as the goto [3], flowcharting [29] and commenting [16], the arguments used are mostly based on personal opinion. In an
area in which we are trying to identify which notations are better for humans to use, few objective measures of comprehension have been presented to demonstrate the superiority of one notation over another. Even the design decisions used in the development of the major programming languages in use today were not developed using objective measures of comprehension. As Sammett states [25]:

> Up until now, virtually all languages have been designed on a completely ad hoc basis, even though the designers may have in fact had (or at least thought they had) rational reasons for their decisions. Thus, if a specific objective is defined, there is no algorithm for designing a programming language to meet that objective, there is not even any clear delineation of various tradeoffs involved, let alone any quantitative measures of them.

Given the infancy of the computing industry at the time these languages were developed, it is understandable that no empirical evidence was used. But the computing industry has matured to the point that there is little use for notations which cannot be demonstrated to be "better" by some objective criteria.

Fortunately, computer scientists have begun to develop such measures. For example, McCabe [20] has proposed that program complexity can be measured by the maximum number of linearly independent paths in a program. According to his measure, the more paths a program has, the more complex the
program is; if the number of paths exceeds 10, the program is probably too complex. A second measure was developed by Gordon [6] and Gordon and Halstead [7] using software science relationships developed by Halstead [10,11]. Gordon's measure is based on the number of operators and operands in a program and claims to measure the time it takes to comprehend a program.

While the above measures are based on characteristics in a program, several other measures have been developed using human subjects. For example, Shneiderman [28] has suggested that the better a subject understands a program, the more lines of code he should remember. In his study, subjects were shown a 67 line Cobol program and asked to memorize it. They were also told that they would be given a comprehension test at the completion of the experiment. Shneiderman found a significant correlation (.54) between the per cent functionally correct lines recalled and the number of correct answers to a comprehension test consisting of 15 multiple-choice questions. Several additional measures based on human subject data have been proposed by Weissman [36] including the use of hand-simulation, subjective rankings of program comprehensibility, and multiple-choice tests to measure comprehensibility.
While the measures described above are certainly improvements over previous subjective criteria, difficulties with the new measures still exist. In the case of ranking comprehensibility or counting recalled lines, subjective judgments are required, either by the subject or by the evaluator. Furthermore, Shneiderman's measure is based on a theory that has not been adequately substantiated. Finally, the measures of McCabe and Gordon have only been validated using programs which are supposed to differ in comprehensibility. Yet the difference in comprehensibility between the programs has never been validated using human subjects.

Thus, what is needed is a comprehension measure which is based on established theoretical foundations, can be objectively measured and uses data obtained from human subjects. The use of reaction time may provide the means by which such a measure can be developed.

The Use of Reaction Time in Comprehension Measurement

Reaction time has been used extensively in psychology to measure various human cognitive processes including comprehension. In reaction time studies, the subject is presented a stimulus and is asked to specify a response as quickly as possible. The time to react to the stimulus is a measure of its difficulty. For example, in a study by Slo-
bin \[32\], subjects were asked to compare sentences they heard with pictures they were shown. The time to indicate whether the sentence accurately described the picture was a measure of the comprehensibility of the sentence. In this study, truth, affirmation and voice of the sentences were compared. Results indicated that true statements were easier to process than false statements, active easier than passive and affirmation easier than negation.

A similar approach using reaction time may be applicable to the study of computer program comprehension. For example, functionally equivalent programs which differ only in notation could be presented to subjects. The program which took the least time to understand would be the one with the clearest notation. Another possible paradigm would require subjects to hand-simulate a program. By observing the subject's reaction times over several trials, the point at which understanding occurred could be identified.

The use of reaction time as a measure of comprehension offers several advantages. First, it is objective; that is, it requires no judgements of comprehensibility by the subject or functional correctness by the researcher. Second, reaction time is easy to measure and can be incorporated in a variety of paradigms involving human subjects. Third,
reaction time data are of direct interest. For instance, if a particular notation reduces comprehension time by 30 per cent, the time to compose, debug and modify a program may likely be reduced by a proportional amount, resulting in reduced software costs. Finally, the reaction time data can be used to validate the other measures of comprehension. Using the reaction time measure, the seemingly disparate results of comprehension measure research, comprehension factor research and psycholinguistic research can be reconciled. The present research was conducted to begin that reconciliation.

Statement of Problem

The goals of this research are three-fold:

1. Develop a measure of comprehension using reaction time as the dependent measure

2. Collect and analyze human processing data of simple expressions and statements

3. Compare the results collected in 1) with the measures of comprehension proposed by McCabe and Gordon
Dissertation Organization

In the next chapter, the literature is reviewed in terms of the comprehension measures which have been proposed and the comprehension factors which have been studied. The psycholinguistic data relevant to program comprehension are also reviewed. The third chapter describes the experimental procedures employed in the pilot study. Refinements to the experimental procedures are described and the results of the final study are detailed in the fourth chapter. The fifth chapter contains the summary and conclusions.
REVIEW OF LITERATURE

In the computer science literature, both measures of and factors affecting computer program comprehension have been proposed and studied. The studies relating to both the proposed measures and comprehension factors will be described in the first two sections.

Another source of information concerning program comprehension is the studies reported in the psycholinguistic literature. Psycholinguists have studied both comprehension factors and the measurement of comprehension in human language. Such information is highly relevant to computer program comprehension and will be detailed in the final section of this chapter.

Measures of Program Comprehension

In the computer science literature, control flow has been identified as one of the most important characteristics of a program since it embodies the logic of the program. One would suspect that a program which has complex control flow would be difficult to comprehend. Therefore, by measuring control flow complexity, program comprehensibility could also be determined.
McCabe [20] has proposed the cyclomatic number as a measure of control flow complexity. The cyclomatic measure is based on the maximum number of linearly independent paths in a program. Since all possible paths in a program can be expressed as a linear combination of the independent paths in a program, the number of independent paths serves as a minimum measure of the flow of data through the program. For example, Figure 1 shows three programs which print the minimum of three numbers. Alongside the programs are the corresponding program control graphs. According to McCabe, the maximum number of independent paths in a control graph is given by:

Equation 1. \( V(G) = e - n + 2p \)

where \( G \) is the program control graph
\( e \) is the number of edges
\( n \) is the number of nodes
\( p \) is the number of connected components

For example, a set of independent paths for the second version in Figure 1 might be represented by \( A(127) \), \( B(1256) \) and \( C(1347) \), where the letters \( A, B \) and \( C \) are the independent path names and the numbers refer to the paths between nodes.
IF $X \geq Y$
    THEN IF $Y \geq Z$
        THEN $W = Z$
        ELSE $W = Y$
    ELSE IF $X \geq Z$
        THEN $W = Z$
        ELSE $W = X$
    PUT LIST($W$);

$W = X$;
IF $Y < W$
    THEN $W = Y$
IF $W < Z$
    THEN $W = Z$
PUT LIST($W$);

CASE
    $X < Y$ AND $X < Z$ DO; $W = X$; END;
    $Y < X$ AND $Y < Z$ DO; $W = Y$; END;
    ELSE DO; $W = Z$; END;
ENDCASE;
PUT LIST($W$);

Figure 1: Three Versions of a Program to Compute a Minimum

To describe any other path in terms of the independent paths, it is convenient to use the vector representation shown in Figure 2.

Figure 2: Vector Representation of Independent Paths
Each column represents one of the edges of the graph and an entry in the vector indicates that the edge is included in the path. In the case that Z > Y > X, the path can be expressed as C + B - A (13456).

Equation 1 requires the construction of a program control graph. Since the construction is difficult for large programs, McCabe simplified equation 1 to:

Equation 2. \( V(G) = \text{Number of predicates} + 1 \)

In practice, he found that compound predicates such as "IF C1 AND C2 THEN" should have a complexity of 3 instead of 2 since the compound predicate could be written as "IF C1 THEN IF C2 THEN". Thus, the practical measure he used was actually the number of conditions + 1. He advised that a \( V(G) \) of 10 appeared to be a practical limit for a program module.

Using the number of conditions as a measure of complexity has some interesting ramifications for a CASE statement like the one shown in Figure 1. The operation of the case is such that only one of the three assignments is to be executed. Based on the number of predicates, the cyclomatic measure is 3. But using the number of
conditions, the value of the measure is 5. Yet it does not seem reasonable that the CASE program should be nearly twice as difficult to understand as the other programs. In fact, McCabe found that "the only situation in which this limit has seemed unreasonable is when a large number of independent cases followed a selection function (a large case statement), which was allowed."

Myers [22] has suggested that compound predicates such as "IF C1 AND C2 THEN" are easier to understand than nested IF statements. Therefore, he suggests that the interval (decision statements + 1, conditions + 1) would be a more accurate measure of complexity than just the count of conditions. The CASE program would then have a measure of complexity of (3,5), compared to (4,4) for the nested IF program and (3,3) for the sequential program. Hansen [13] has suggested using the 2-tuple (CYC-MIN, operator count), where CYC-MIN is defined as one plus the number of branches resulting from alternative or repetitive constructs such as IF or DO-WHILE statements. He felt that his measure better reflected the entire complexity of the program. For the programs in Figure 1, Hansen's measure would be (4,8), (2,6) and (2,10), respectively.
All the measures mentioned thus far will partially measure the complexity of a program. However, since the measures are based on only a few characteristics of a program, they may not completely reflect the complexity of the program. As Elshoff and Marcotty [4] stated in their discussion of the cyclomatic measures,

Since this measure only reflects one aspect of a program's complexity, it is hardly reasonable to use it alone as a base for more general results such as the desirability of an occasional GOTO.

A second measure has been proposed by Gordon [6] and Gordon and Halstead [7] using the software science measures developed by Halstead [10,11]. In Halstead's work, the volume (in bits) of a program is postulated to be:

Equation 3. \( V = N \log_2(n) \)

where \( N \) is the total number of occurrences of operators and operands in the program

and \( n \) is the total number of unique operators and operands in the program.

The volume equation can be explained by the following intuitive argument: \( \log_2(n) \) bits are required to specify \( n \) unique operators and operands. Since there are \( N \) such items in the program, the volume is given by \( N \log_2(n) \). The level of abstraction of a program is given by:
Equation 4. \[ L = \frac{V^*}{V} \]

where \( V^* \) is the most compact representation of a program.

For example, \( Y = \sin(x) \) has a \( V^* \) of \( 4 \cdot \log_2(4) = 8 \), since this is the most compact representation possible. If the language allows this notation, then the level of the program would be 1. However, if the \( \sin(x) \) must be implemented by computing the Taylor series, the actual volume of the program would be much larger than 8. Subsequently, the level of the program which actually computed \( \sin(x) \) would be less than 1.

Halstead originally proposed that the effort to create a program was \( E = \frac{V}{L} \) since larger programs would require more effort to create but the effort would be reduced if it could be written using more abstract notation. Halstead claimed that effort is measured in the units of mental discriminations or decisions a programmer has to make as the program is being created. Stroud [33] has found that the human mind is capable of between 5 and 20 discriminations per second.

Gordon proposed that the measure of effort was actually the effort to understand rather than to create a program.
(Tables 1 and 2 show the calculations of Gordon's measure, \( Ec=V^2/V^* \), for the programs in Figure 1). To test this hypothesis, he selected 46 programs from various publications. He chose cases where authors compared both well-written and poorly-written programs which implemented the same task. He then compared both his measure \( Ec=V^2/V^* \) and the number of statements \( Ns \) to see if \( Ec \) and \( Ns \) ranked the programs in the same order. To "agree" with the author, the values had to differ by 10%. Gordon found that \( Ec \) ranked the programs in the same order in 40 of 46 cases, while \( Ns \) ranked the program properly in only 31 of the 46 cases. He also found mitigating reasons why the \( Ec \) measure did not rank the 6 remaining programs properly.

Gordon's results are not too surprising. In general, the volume of a program should decrease if the program is simplified, resulting in a lower value of \( Ec \). On the other hand, merely reducing the number of statements would not necessarily make the program simpler since statement complexity is variable. Thus \( Ns \) would not be as valid a measure of comprehensibility as \( Ec \). While Gordon's experiment demonstrates that \( Ec \) corresponds to the authors' subjective rankings of program comprehensibility, the experiment does not demonstrate that comprehensibility, as measured by \( Ec \), is proportional to \( V^2 \). A better test of \( Ec \) would be to com-
**TABLE 1**

Operators and Operands in Versions of Minimum Program

<table>
<thead>
<tr>
<th>Operators or Operands</th>
<th>Nested IF Version</th>
<th>Sequential IF Version</th>
<th>CASE(^1) Version</th>
<th>MIN(^2) Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>ELSE</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ASGN</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>&gt;=</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>&lt;</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>PUT LIST</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>;</td>
<td>5</td>
<td>4</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>AND</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>DO-END</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>END OF PROGRAM</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MIN</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>W</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Y</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Z</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

\(^1\)The CASE statement is assumed to act like a sequence of IF statements.

\(^2\)The MIN program would be: PUT LIST(MIN(X,Y,Z));.

pare Ec to the actual time it takes an individual to understand a program. Such a test is described in Chapter 4.

Weissman [36] used a number of comprehension measures including:
TABLE 2

Software Science Parameters for Versions of Minimum Program

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Nested IF Version</th>
<th>Sequential IF Version</th>
<th>CASE Version</th>
<th>MIN Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>n&lt;sup&gt;1&lt;/sup&gt;</td>
<td>11</td>
<td>10</td>
<td>13</td>
<td>7</td>
</tr>
<tr>
<td>N&lt;sup&gt;2&lt;/sup&gt;</td>
<td>35</td>
<td>24</td>
<td>40</td>
<td>7</td>
</tr>
<tr>
<td>V&lt;sup&gt;3&lt;/sup&gt;</td>
<td>121</td>
<td>80</td>
<td>148</td>
<td>20</td>
</tr>
<tr>
<td>E&lt;sup&gt;c&lt;/sup&gt;</td>
<td>732</td>
<td>320</td>
<td>1095</td>
<td>20</td>
</tr>
</tbody>
</table>

<sup>1</sup>Count of unique operators and operands.
<sup>2</sup>Count of all occurrences of operators and operands.
<sup>3</sup>Volume of version in bits.
<sup>4</sup>Gordon's measure of effort.

1. the number of data values hand-simulated within a given time
2. a self-evaluation of program understanding by the subjects
3. a multiple-choice quiz
4. a fill-in-the-blank description of the program

Unfortunately, the results he obtained were inconclusive. In many cases, the sample sizes were too small to perform an adequate statistical analysis. Furthermore, each of the
comprehension measures he used were too subjective or may not have been valid measures of comprehension. For example, self-evaluation tests are not reliable since subjects may indicate that they understand a program even though they may not. Using quizzes or fill-in-the-blank tests are more likely to measure actual comprehension but program comprehension tests are difficult to construct. Also, the subjects may be guided to the answer by the choices on the test or the wording in a fill-in-the-blank question. Finally, recording data values can be accomplished without really understanding the program. As Weissman has stated: "Although hand-simulation as such is not a valid measure of understanding, it is an important factor which can contribute to one's understanding of a program."

A different measure of comprehension has been proposed by Shneiderman [28]. According to Shneiderman's syntactic/semantic model of programming behavior, a programmer builds two types of knowledge structures: syntactic knowledge -- language dependent, acquired through rote memorization -- and semantic knowledge -- language independent, acquired through meaningful learning and understanding. Evidence that the syntactic/semantic knowledge structures are separate is given in a study by Sachs [24]. In his experiment, subjects were read passages followed by a test
item sequence. They were asked if the test sentence was identical to, a syntactic variation of, or a semantic variation of the sentence in the text passage. The results indicated that once the sentence was fully processed, the syntactic information of the sentence was forgotten while the semantic information was retained.

Shneiderman proposed that if a programmer were asked to memorize a program, the number of lines recalled would be a good measure of comprehension. Since the program could not be learned by rote memorization, the number of lines recalled would have to indicate the degree to which a program had been understood or semantically encoded. To investigate this theory, Shneiderman conducted two experiments. In the first experiment, 62 students were given a 26-line Fortran program and were asked to memorize and make three modifications to it. The students were divided into two groups. One group (HI) saw a program with a six line high level comment block at the beginning of the program, and the other group (LO) saw a program with single line comments interspersed in the program. The memorization task was measured by the number of lines perfectly recalled, while each of the three modification tasks was graded by a teaching assistant using a ten point score. The results indicated that the main effects of comments and modification
difficulty were significant in favor of the HI group but the interaction was not. The correlation between the total modification score and lines perfectly recalled was .56 (p<.01).

In a second experiment, Shneiderman had 48 students memorize a 67-line COBOL program and complete a 15 question comprehension test. The order of the recall task and quiz was counterbalanced. The lines perfectly recalled were recorded by a grader unfamiliar with COBOL while the lines functionally correct were recorded by an experienced grader. A correlation between quiz score and lines recalled perfectly was found to be .3, while the correlation between quiz score and per cent of lines functionally correct was .54.

Although the studies of Shneiderman indicate that the per cent of lines recalled might partially measure comprehension, several results cast doubt on the validity of this measure. First, the previously cited work of Gordon seems to indicate that the number of lines is not as good a predictor of comprehensibility as is the function of operators and operands in a program, possibly indicating that comprehensibility is at least partly a function of statement complexity. If this is the case, the differential effects of
Sentence complexity should be reflected in the recall measure; however they are not. More importantly, using recall as a measure of comprehension seems to be refuted by the previously cited study of Sachs. If the detailed syntax of a sentence is retained for only a short time after a sentence is semantically encoded, then there is no reason to believe that a programmer who completely understands a program will be able to reproduce the program syntax exactly as it was presented. Finally, the goals expressed in any experiment can have a profound effect on the outcome of an experiment as was demonstrated by Weinberg [35]. By asking subjects to memorize a program, the subjects may be accomplishing a different cognitive function from comprehension. They would certainly make a point of trying to retain the exact syntax even though such information may not be necessary for comprehension. Thus, the number of lines recalled would likely be much lower if subjects were only instructed to understand a program but later asked to recall it. On balance, the evidence for using the number of lines recalled as a measure of comprehension is intriguing but not yet compelling.
Factors Affecting Program Comprehension

A number of factors affecting program comprehension and composition have been identified and studied. One of the first studies was reported by Sime, Green and Guest [30]. The researchers were trying to determine if a language with nested IF-THEN statements was easier to use for naive programmers than a similar language with IF-JUMP statements. In the study, two groups of 9 students were each taught one of the languages. Each group was given five sorting-task problems and asked to translate the problem into a program. For example, if a card had the words leafy, green and juicy on it, the program action should be to boil. In other cases, the action might be to grill, fry, roast or reject based on the attributes on the card. The subjects entered the program on a terminal which checked for both syntactic and semantic errors. The results indicated that the nested group performed significantly better based on the number of completed problems, number of semantic errors and time to complete the problems.

In a more recent study by the same authors [31], scope marking in computer conditionals was studied using the same methodology. Subjects were taught one of three languages: the branch-to-label language used in the previous experiment, a nested language which used begin-end pairings to
indicate the scope of conditionals as in ALGOL 60, and a
nested language which used a redundant marker scheme such as:

IF hard peel
   NOT hard chop
END hard

The results indicated that the branch-to-label language
resulted in more semantic errors per problem than the nested
languages by an order of magnitude. However, more syntactic
errors were caused by the ALGOL-60-like language. Further­
more, the language using the redundant marker scheme
resulted in a much smaller error lifetime, i.e. the number
of additional attempts required to solve the problem. The
data seem to indicate that the nested languages made it eas­
ier to construct a semantically correct program but often
resulted in syntactic errors, especially forgetting END
statements.

To explain the results, the authors proposed that the
conditional programs consisted of two types of information:
sequence information which indicated the order of execution,
and taxon information which indicated under what conditions
a certain action will be taken (e.g., discovering whether
hard juicy objects are ever roasted). For the tasks in the experiment, the subjects had to translate from taxon information into sequence information to write the program. To debug the program, the process had to be reversed. The sequence information was more clearly displayed using the nested languages because they were automatically indented, indicating the scope of the conditionals. Since the nested languages had fewer semantic errors, this fact is well-supported. Taxon information is best displayed using the redundant marker language because the taxon which controls a given action can be determined without using any sequence information. Because the act of debugging requires both sequence and taxon information, the redundant marker language should, and does, result in the smallest error lifetime.

Using a similar methodology, Miller [21] investigated the effects of conjunction, disjunction and negation in the program composition process. Miller taught naive programmers a laboratory programming language and then asked them to write a program from statements like:

Put a card in box 3 if either the name's second letter is not L or if its last letter is N or both -- only one condition must be satisfied.

The statements varied in the use of OR, AND and NOT. The results indicated that it was easier to develop a program
for a function statement which contained conjunctions rather than disjunctions. Also, the use of negatives in the statement resulted in more incorrect programs.

Love [19] has used the number of lines recalled to study the effects of paragraphing and control flow. In his experiment, Love constructed four versions of four programs: paragraphed, unparagraphed, simple control flow and complex control flow. He then randomly assigned 31 subjects to one of four sequences. The sequences were designed so that each subject saw each program and version but not all program-version combinations. The dependent measure was the percentage of lines of source code recalled correctly in the proper order. He found that the effects of paragraphing were not significant but that the effects of control flow were. The interaction between control flow and class (graduate vs undergraduate) was also significant, indicating that graduate students were more adversely affected by complex control flow than the undergraduates.

Psycholinguistic Results

The processes of program comprehension and sentence comprehension are likely to be very similar. Therefore, the results reported in the psycholinguistic literature may, and in fact do, provide information pertinent to the study of program comprehension.
A study by Gough [8] is a good example. In his study, Gough was trying to determine how people understand complex sentences. His hypothesis was that to understand a complex sentence a person must transform it to its kernel sentence. For example, the passive sentence "the girl was hit by the boy" would be transformed to the kernel sentence "the boy hit the girl". He then theorized that if the transformation takes place, the latency of understanding of a complex sentence would be a function of the number and nature of the transformations separating the complex sentence from its kernel.

To test his hypothesis, subjects were shown a sentence and a drawing and asked to indicate as quickly and accurately as possible whether the sentence correctly described the drawing. The sentences varied in voice (active-passive), truth (true-false) and affirmation (affirmative-negative). Response latency was used as the dependent measure in the statistical analysis. His results confirmed his hypothesis: true sentences were processed significantly faster than false, active faster than passive and affirmative faster than negative. Later studies by Slobin [32] and Haslett [14] using a similar methodology have confirmed these results.
The results of Gough's study are important to program comprehension for two reasons. First, the study indicates the utility of response latency as a measure of comprehension. Second, if affirmative sentences are easier to understand than negative sentences, then expressions which are stated affirmatively should be easier to understand than those which are stated negatively. Therefore, computer languages which contain complementary statements which allow programmers to always state expressions affirmatively may make programs easier to understand. For example, rather than saying "DO WHILE(x≠5 AND x≠8)" a programmer should be able to say "REPEAT UNTIL(x=5 OR x=8)". As in natural language, statement diversity may promote understanding.

A second related area of research in psycholinguistics has been the study of embedded clauses. Below are examples of three types of embedding:

Center-embedding: The angels that the theologians that the later cynics that modern science favors ridiculed counted stood on the head of a pin.

Right-embedding: The umpire called a balk that the southpaw pitcher hit that the coach replaced.

Left-embedding: the electricity-powered, toe-chomping, rock-throwing lawn mower ran over its own cord.
In a study by Schwartz [26], subjects were asked to indicate if they thought an embedded sentence was comprehensible and to mark on a scale from 1 to 7 the confidence of their choice. Results indicated that subjects felt that the center-embedded sentence was most difficult followed by the right-embedded and finally the left-embedded sentence. Adding clauses to the center and right-embedded sentences made the sentences less comprehensible but did not affect the comprehensibility of the left-embedded sentence. Hamilton and Deese [12] obtained similar results, leading them to believe that comprehension depends heavily on the contiguity of the main constituents (subjects and verbs) in a sentence.

A further confirmation of the embedding results was found by Larkin and Burns [18] using a different experimental technique. Larkin proposed that to understand an embedded sentence, subjects must pair the nouns and verbs properly. To observe the effects of embedding, he presented four conditions or types of stimuli to the subjects: a series of digits, letters and digits, nouns and verbs, and sentences. The length of the stimuli varied from 4 to 10 items. For each stimulus the subjects were asked to pair the first and last items, the second and next to last item, etc. For example, if the stimulus was FJL352, the pairs would be F2, J5 and L3. The subjects could recall the pairs
in any order. The sentences in the fourth condition (sentences) were constructed from the nouns and verbs from the third condition by placing "The" at the beginning of the sentence and inserting "that the" between adjacent nouns. Thus, the noun-verb stimulus "army defenders withdrew stopped" would become the sentence "The army that the defenders withdrew stopped". Based on the per cent of correct pairs reported, the results showed that digit pairs were easiest to recall, followed by letters and digits, nouns and verbs, and sentences. Furthermore, as the length of the stimulus increased, the per cent of correct pairs decreased, indicating the primary limitation of immediate memory on pairing performance.

Again, the psycholinguistic results reported above have possible applications to programming languages. For example, the center-embedding nature of languages such as LISP or APL may be partly to blame for the difficulty in understanding programs written in these languages. Possibly, some expressions written in these languages might be written in left-embedded form (e.g., red flag rather than flag(red)). Also, if the main constituents of programs can be considered to be the conditions and actions in a program, nested if statements should be avoided because they result in a center-embedded structure which is difficult to under-
stand. For cases where nested if statements are used, new control structures should be developed to accommodate the logic of the program.

A final study in psycholinguistics which has relevance to program comprehension was completed by Paris [23] on the comprehension of logical connectives such as conjunction (and, but, both-and), conjunctive absence (neither-nor), disjunction (or, either-or), conditionality (if-then) and biconditionality (if-and-only-if-then). The paradigm used by Paris was to present a proposition like "if the candle is burning, then the bird is in the nest", as well as a pair of pictures. One picture might have the candle burning with the bird in the nest, while the other picture might not have the candle burning but show the bird in the nest. Based on the picture pair, the subjects were asked to indicate if the proposition was true or false. Using both the number of errors and reaction time as dependent measures, the ranking of connectives from hardest to easiest was found to be: (1) conditionality, (2) disjunction, (3) biconditionality and (4) conjunction. The conditional proposition was most difficult for the cases when the condition of the IF was false. In this case, the proposition is true according to formal logic, but most subjects indicated that it was false. Paris also found that either-or was superior to or, and mixed
truth values for the conditions of the OR caused the greatest difficulty. While the conditionality results don't appear to have direct application to programming, the conjunction and disjunction results do. If the AND operator is easier to comprehend than the OR, then converting OR-expressions to AND-expressions may often make the program easier to understand.

Summary

In this chapter, several measures of comprehension have been described including McCabe's cyclomatic measure, Gordon's Ec and Shneiderman's count of correctly recalled lines in a program. While there is evidence that these measures can assess some aspects of program comprehensibility, further experiments are needed to validate these measures.

Factors affecting program comprehension were also described. These included control flow, commenting, paragraphing and mnemonic variable names. It was also shown that redundant marking of IF statements could improve the program development process for naive programmers.

Finally, the psycholinguistic results indicate that the embedded structure of languages like LISP and APL are likely to be difficult to understand because similar constructions
in English have been found difficult to understand. Also, logical connectives like AND and OR differ in comprehensibility. Finally, the use of reaction time has been demonstrated to be a viable means of measuring comprehension.
THE PILOT STUDY

In this chapter, reaction time is proposed as a new measure of computer program comprehension. A pilot study is described which used the new comprehension measure to evaluate human processing of simple expressions (less than, equal to etc.) and selection statements (IF-THEN-ELSE and CASE). Results of the pilot study described in this chapter lead to refinements in the comprehension measure which are described in the fourth chapter.

A Proposed Measure of Comprehension

Consider three program versions shown in Figure 3. Each version stores the maximum of A, B and C in L. But which version is the most comprehensible and how can the comprehensibility of each be measured? Shneiderman has defined comprehension as

the recognition of the overall function of the program, an understanding of intermediate level processes including program organization and comprehension of the function of each statement in a program.

In the context of Figure 3, a subject would understand one of the versions if he could state that it finds the maximum
of three numbers. Of course, it might be possible for a subject to state the program function without fully understanding the intermediate level processes. The most obvious example would be a program which included a comment stating the function of the program. But barring obvious hints in the program, a version which is claimed to be more comprehensible should be able to convey its function more clearly and quickly to the programmer than the other versions.

```
IF A>=B
    THEN IF A>=C
        THEN L=A;
        ELSE L=C;
    ELSE IF B>=C
        THEN L=B;
        ELSE L=C;
```

```
L=A;
IF B>L
    THEN L=B;
IF C>L
    THEN L=C;
```

```
CASE
    A>=B AND A>=C DO L=A END;
    B>=A AND B>=C DO L=B END;
    C>=A AND C>=B DO L=C END;
ENDCASE;
```

Figure 3: Three Versions of a Maximum Program
To measure how well a program version conveys its function, consider what would happen if a subject were given values for $A$, $B$ and $C$ and asked to hand-simulate one of the versions in Figure 3 and indicate the resulting values for $L$. If the subject were aware of the version's function, the time to respond with the answer would be considerably less than if the subject had no idea what the function was. Thus, if a subject could be timed in a hand-simulation task both before and after he had discovered the program function, the time he took to determine the function could be established by noting when a large decrease in reaction time occurred. A version could then be judged more comprehensible if it took less time to understand.

The actual paradigm might be conducted as follows. Initially, the subject would be shown the program and input values and asked to specify the results of the program. When the answer was provided, and the time to answer recorded, the subject would be given a brief period of time to determine the function of the program. The next set of input values would then be provided and the process repeated. Under these conditions, a reaction time curve similar to Figure 4 might occur. During the first several trials, the subject would not know the function of the program and the reaction times would be relatively large.
Reaction times would decrease with practice but no dramatic drop would be expected. However, if the subject determined the function after the third trial, a large drop in reaction time would occur on the fourth trial, indicating that the subject had internalized and understood the function of the program. Subsequent trials would then remain at this near optimal level.

Figure 4: An Example Reaction Time Curve

Ideally, the time the subject took to comprehend the program could be measured directly by calculating the time from the start of the trial series until comprehension occurred. Unfortunately, it may be difficult to distinguish a drop in reaction time which was due to comprehension from a drop which was due to practice or individual variation.
Furthermore, the difference in processing time between understanding and not understanding is likely to vary widely between individuals, making the judgement of comprehension a subjective process. A more objective measure would be the mean of the trial reaction times. A version which resulted in earlier understanding than another version would have a smaller mean trial time because more trials with small reaction times would occur after understanding. Thus the version with the smallest mean trial time would be judged the most comprehensible. The only difficulty with using the mean trial time is that it does not reflect if a subject ever actually understood the program. However, the subjects could be asked to specify the function of the program at the end of the trial series. Individuals who did not understand the program could then be identified.

Using mean trial time as a measure of comprehension offers several advantages. First, it is easily measured especially if a terminal is used to display the programs and record the reaction times. Second, it is almost entirely objective. The only aspect which is subjective is the judgement of the correctness of the function statement provided by the subject. But judging the function should not be difficult, particularly if the program function can be stated easily such as "find the minimum" or "find the
maximum\textsuperscript{*}. Also, by tabulating the correctly stated functions, another measure of comprehensibility would be available. For example, if version 1 resulted in more correctly stated functions than version 2 (assuming that an equal number of subjects viewed both), then version 1 would likely be more comprehensible.

**Procedure**

The pilot study was conducted using a PLATO terminal. PLATO is a large time-sharing computer system centered at the University of Illinois and was designed primarily for instruction. The PLATO IV terminal used in this study consists of an 8-inch square plasma screen, touch panel and keyset. Output text and graphics can be displayed on the plasma panel which, while using different technology, resembles the more familiar CRT display. Input to the terminal can be provided by either typing on the keyset or touching the touch panel. The touch panel consists of 16 infrared transmitters and receivers located in the borders of the panel and lies flush with the plasma screen. When a subject touches the screen at the intersection of a vertical and horizontal infrared beam, the panel can sense the location where the subject touched. All reaction times were recorded using the touch panel as the input device.
The terminal session began when the subject arrived at the terminal room at his scheduled time. The terminal room was a small 8-by-11-foot room in which two PLATO terminals were located. Access to the room was controlled by the researcher to minimize noise and reduce distractions. Each subject was shown how to operate the terminal and was given a sign-on which he could use to observe instructional computer science lessons after the experiment was over.

The remainder of the session was controlled by the experimental program. The researcher was available in the next room if the subject had difficulty operating the terminal. The experimental program consisted of four sections. In the first section, the subject was asked to enter background information about his or her programming experience. Then the information necessary to operate the touch panel and terminal was described. Following this, the programming notation used in the experiment was described. Each operator was shown, its meaning defined and some examples given. The subject was also asked if the operator had been used before. For example, the operator '>\text{'} was defined as yielding true if, in the expression A>B, A was greater than B. Otherwise, the expression yielded false. Examples such as '2>1 yields true' and '5>6 yields false' were also presented. The subject was then asked to evaluate three expres-
sions like '5>3' and indicate the answer by touching the screen. If the subject made an error, the answer was requested again. The subject's response time was also displayed to familiarize the subject with the timing procedures used later in the experiment.

The second section consisted of a motor reaction test. The subjects were shown a display similar to Figure 5. A number appeared in the small isolated box and the subject was asked to touch the same number on the calculator-style keyboard on the screen. The digits 1 through 9 were displayed three times, the order being initially randomized and repeated three times, yielding 27 reaction times. In a similar fashion, the true and false keyboard was tested using a random order of 3 T's and 3 F's, yielding 6 reaction time values. These data were later used as a covariate in the statistical analysis.

```
[11231] [IT]
[14561] [FP]
[17891]  
```

Figure 5: Keyboard Layouts for Subject Input
The third section consisted of reaction time tests for the operators shown in Table 3. The test for each operator or statement consisted of six trials. For the first trial, the subject saw an input value along with an expression involving the operator and was asked to evaluate the expression and indicate its result. For instance, the subject might see 'x=5; x>5' and be asked if the second expression were true or false. If the subject correctly answered the question, the program recorded the reaction time, delayed 3 seconds, erased the first statement and replace it with a different assignment statement for which an answer was requested. The later expression was the same for all six trials. If the subject answered incorrectly, the reaction time was recorded but the subject was asked to try again. The six data values were chosen so that most of the logical expressions had an equal number of true and false answers. The order of the input data values was initially randomized, but all subjects saw the same data values in the same order for all the expressions so that comparisons between expressions and trials could be made. The order in which the expressions were presented was also fixed for all subjects.

In the fourth and final section of the program, three programs were presented to subjects. As in the third
### TABLE 3

The Operators and Expressions in the Pilot Study

<table>
<thead>
<tr>
<th>Symbol used in Text</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASGN</td>
<td>(X=5) (^1)</td>
</tr>
<tr>
<td>EQ</td>
<td>(X=2)</td>
</tr>
<tr>
<td>NE</td>
<td>(X \neq 4)</td>
</tr>
<tr>
<td>GT</td>
<td>(X &gt; 4)</td>
</tr>
<tr>
<td>LT</td>
<td>(X &lt; 5)</td>
</tr>
<tr>
<td>AND</td>
<td>(X &gt; 2 \ AND \ X &lt; 6)</td>
</tr>
<tr>
<td>OR</td>
<td>(X &gt; 7 \ OR \ X &lt; 2)</td>
</tr>
<tr>
<td>NOT</td>
<td>(\text{NOT}(X &gt; 4))</td>
</tr>
<tr>
<td>IF</td>
<td>IF (X &gt; 4) THEN (Y = 2) ELSE (Y = 4)</td>
</tr>
<tr>
<td>CASE</td>
<td>CASE (x = 1) DO (Y = 7) END (x = 2) DO (Y = 5) END (x = 4) DO (Y = 2) END (x &gt; 4) DO (Y = 1) END ENDCASE</td>
</tr>
</tbody>
</table>

\(^1\)The values for \(X\) were 2, 5, 9, 4, 8 and 1. The order of the \(X\) values was the same for all expressions.

In this section, six trials were used for each program. Each trial consisted of a different set of input values. For the first trial, the subject was shown the data values and program and asked what the value of a certain variable would be if the program were executed. When the subject gave the correct
answer, he or she was directed to try to determine the function of the program. At the end of five seconds, the old input values were erased, new values printed and the process repeated. As before, the reaction time of the first answer was recorded, irrespective of the correctness of the answer. When all six trials for a program had been given, the screen was erased and the subject was asked to write down the function of the program on a sheet of paper provided by the researcher. The subjects could then rest or proceed to the next program by touching the screen. At the conclusion of the fourth section, the subject was thanked for his or her participation and the session was terminated.

Subjects

The subjects were undergraduate students enrolled in two sections of a sophomore level programming languages course at Iowa State University. The subjects were asked to volunteer for the study but were given two incentives to participate. First, they were told that if they participated in the experiment and received a borderline final grade in the course, they would be given the higher course grade. Also, they were given access to the PLATO system computer science lessons after the experiment was over. Fifty-seven of 75 students agreed to participate.
A summary of the background of the students is shown in Table 4. In general, these statistics indicate that the average subject had been programming for about two years, had produced a program in excess of 1000 lines and spent about 40 minutes performing the experiment.

**TABLE 4**

Background of Students in Pilot Study

<table>
<thead>
<tr>
<th>Measure</th>
<th>N</th>
<th>Mean</th>
<th>S.D.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Months Programming</td>
<td>50</td>
<td>25.92</td>
<td>17.05</td>
</tr>
<tr>
<td>Size of Largest Program</td>
<td>50</td>
<td>3.38</td>
<td>.53</td>
</tr>
<tr>
<td>Time of Session</td>
<td>50</td>
<td>41.96</td>
<td>8.72</td>
</tr>
</tbody>
</table>

Subjects were asked to classify the size of the largest program as less than 10, 100, 1000 or 10,000 lines. Thus, the measure is in terms of \( \log(\text{lines}) \).

in minutes.

**Experimental Design and Results**

The entire experiment actually consisted of three distinct experiments: (1) motor reaction time analysis, (2) simple expression analysis and (3) program-version analysis.
To simplify the description of these experiments for the reader, the appropriate design and results for each experiment will be described in separate sections.

Motor Reaction Time Analysis

For this experiment, each subject responded to the digits and the logical values true and false three times. To determine if differences existed between the digits and letters (hereafter referred to as keys), a randomized block design [17] was used in which the dependent measure was the mean of the three trials for each key, with keys being the independent measure. The logarithmic transformation was used to normalize the data.

Although 57 subjects participated in the experiment, the data for four subjects were lost due to program and system errors unrelated to the experiment. The subsequent analysis included 53 subjects. The analysis of variance table for this experiment is shown in Table 5. The main effects of keys was significant, indicating the differential effect of the keys.

The results indicate that the True and False keys were touched the quickest, probably because the subjects had to choose between only two squares for the true-false test.
TABLE 5
Statistical Analysis for Keys

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>MS</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key¹</td>
<td>10</td>
<td>.104</td>
<td>31.35**</td>
</tr>
<tr>
<td>Subject</td>
<td>52</td>
<td>.020</td>
<td>6.16**</td>
</tr>
<tr>
<td>Residual</td>
<td>520</td>
<td>.003</td>
<td></td>
</tr>
</tbody>
</table>

¹The key means (in seconds) are shown below:

<table>
<thead>
<tr>
<th>T</th>
<th>F</th>
<th>1</th>
<th>2</th>
<th>5</th>
<th>9</th>
<th>8</th>
<th>3</th>
<th>4</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>.782</td>
<td>.805</td>
<td>.865</td>
<td>.912</td>
<td>.942</td>
<td>.946</td>
<td>.974</td>
<td>.988</td>
<td>1.037</td>
<td>1.050</td>
<td>1.053</td>
</tr>
</tbody>
</table>

**p<.01

For the digits, it appears that the keys in the upper left corner were touched the quickest while the keys in the outer perimeter tended to be touched more slowly. The differing key effects imply that subsequent experiments using the digit and true-false displays as answer pads could influence the outcome of the experiments. Since the main effect of subjects was also significant, it was decided to use the data in the motor reaction time experiment to compute appropriate values which could be used as covariates in the subsequent experiments. The computation of the appropriate covariate values will be described in the next section.
**Simple Expression Analysis**

The order of the expressions, the order of the input data and the expressions themselves were the same for each subject. The expressions and input data are shown in Table 3.

For the first expression, the assignment statement with one of the input values was displayed and the subject was asked to indicate the value of $x$ if the statement were executed. For the remaining expressions, the assignment statement was printed above the expression to be evaluated. The same process was used with different input values being substituted in the assignment statement.

In the previous section, it was found that different numbers on the screen resulted in significantly different motor reaction times (MRT). To compensate for the MRT differences in the statistical analysis, a covariate was formed from the motor reaction times. The covariate was a mean of the MRTs for the answers for each expression by subject. For example, each subject had a covariate value for the assignment statement which was the mean of the MRTs to touch $2, 5, 9, 5, 8$ and $1$. The same was true of the other expressions except that the covariate was the mean of the MRTs to touch the appropriate answers for each expression. However, in
the statistical analysis of the expressions using reaction time as a covariate, the effect of the covariate was not significant. Since the conclusions regarding the main effects and interactions were found to be the same whether the covariate was present or absent, it was decided to conduct all subsequent analyses without the covariate present.

The statistical design employed was a randomized block factorial with two repeated measures: (1) expressions (10 levels), and (2) trials (6 levels). The analysis is shown in Table 6. As can be seen, both the effects of expressions and trials were highly significant. Additionally, the interaction between expressions and trials was highly significant, indicating that expressions behaved differently across trials.

The expression means were analyzed using a DUNCAN test and the results are also shown in Table 6. As expected, the assignment statement was the easiest to process while the IF and CASE statements were the most difficult. Additionally, OR was found to be significantly different from AND, which agrees with previous results found in the psycholinguistic literature.

However, the means for the IF and EQ expressions seem unusually large. The reason for this can be explained by
TABLE 6
Statistical Analysis for Expressions in Pilot Study

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>MS</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject</td>
<td>49</td>
<td>.535</td>
<td>15.44**</td>
</tr>
<tr>
<td>Expression¹</td>
<td>9</td>
<td>5.594</td>
<td>161.36**</td>
</tr>
<tr>
<td>Digit</td>
<td>5</td>
<td>5.012</td>
<td>144.58**</td>
</tr>
<tr>
<td>Expression*Digit</td>
<td>45</td>
<td>.208</td>
<td>5.99**</td>
</tr>
<tr>
<td>Residual</td>
<td>2891</td>
<td>.035</td>
<td></td>
</tr>
</tbody>
</table>

¹The Duncan test for the expression means (in seconds) was (means not significantly different are underlined): ASGN NE GT LT EQ AND OR NOT CASE IF .869 .939 1.142 1.145 1.204 1.473 1.721 1.788 2.033 2.079 **p<.01

examining Figure 6 which shows the trial means for the ten expressions. As can be seen, the slope from the first trial to the second for the IF and EQ expressions was considerably different from the other expressions. The reason lies in the order of presentation of the expressions. The assignment statement was the first expression to be presented and required the numerical screen display. When the trials for the assignment statement were complete, the screen display had to be changed to the true-false format to accommodate the answers for the EQ expression. The opposite switch in screen displays occurred between the NOT and IF expressions.
Presumably, subjects required extra time to adapt to the switch in screen displays. Also, the first trial times for all the expressions were higher than any subsequent trials because the subjects had to read and evaluate the expressions on the first trial but only evaluate the expressions on subsequent trials.

To eliminate the above anomalies, an analysis was conducted without trial one. The results are shown in Table 7. Now the mean for IF was less than CASE but not significantly so; the mean for EQ was less than the previous analysis but still significantly greater than NE. The reason for this latter result is probably due to the fact that expressions which evaluate true are typically processed more quickly than those that evaluate false [21]. Since EQ evaluated true for only one trial of six while NE evaluated true for five trials, the differential effects of processing true and false expressions probably caused the difference between NE and EQ.

Since the data values were presented in a fixed order for all subjects, order effects like those mentioned in the previous paragraph make further analysis of the trial data for expressions difficult. However, some intriguing trends in the trial data are worthy of comment. First, note that
Figure 6: Trial Means for Expressions in Pilot Study
55

TABLE 7

Modified Statistical Analysis for Expressions in Pilot Study

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>MS</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject</td>
<td>49</td>
<td>.486</td>
<td>14.16**</td>
</tr>
<tr>
<td>Expression(^1)</td>
<td>9</td>
<td>4.910</td>
<td>142.95**</td>
</tr>
<tr>
<td>Digit</td>
<td>4</td>
<td>1.457</td>
<td>42.41**</td>
</tr>
<tr>
<td>Expression*Digit</td>
<td>36</td>
<td>.115</td>
<td>3.33**</td>
</tr>
<tr>
<td>Residual</td>
<td>2401</td>
<td>.034</td>
<td></td>
</tr>
</tbody>
</table>

The Duncan test for the expression means (in seconds) was (means not significantly different are underlined):

ASGN NE EQ GT LT AND OK NOT IF CASE
.787 .856 1.021 1.066 1.143 1.324 1.609 1.657 1.851 1.972

\(^{**}p<.01\)

the curves for LT and GT tend to be opposite in form. A partial explanation for these trends comes from the work of Banks, Fujii and Kayra-Stuart [1]. They found that if subjects are asked to specify the larger or smaller of two digits, subject reaction times decrease as the difference or split between the two digits increase. Furthermore, as the magnitude of the smallest digit increases, the reaction time also increases. In the case of the GT expression in the present experiment, the smallest reaction times occurred on trials 3 and 6 when the maximum splits occurred. Also, a relatively larger reaction time occurred on trial 4 when the
split was 0. Yet the trend for the LT expression was opposite to the aforementioned behavior. On trial 3 for LT, the reaction time was maximum when the split was largest, while on trial 2 the reaction time was minimum when the split was 0. Further experiments are required to investigate this phenomenon.

The trend for the Or data suggests the way in which the OR expression was processed. If the first sub-expression was true (trials 3 and 5), some of the smallest reaction times were recorded. In the one case when the first sub-expression was false but the second sub-expression was true (trial 6), there was a slight upturn in reaction time, possibly signifying that the whole expression had to be processed before a true result could be deduced. Of course, in cases where the OR expression was false, the whole expression had to be processed. Coupled with the longer reaction time for processing a false expression, the reaction times would be expected to be among the largest of all six trials.

For the AND expression, the first sub-expression was false on the first and last trials. If subjects quit processing at this point, the reaction times should be lower than the trials where both sub-expressions had to be
evaluated. Although it is difficult to make any statement about the first trial because of the confounding with reading time, the last trial had an intriguing downturn in reaction time, possibly suggesting that subjects process an AND expression only as far as needed to evaluate the result. Again, more experiments are needed.

The zig-zag nature of the two selection statements is also worthy of comment. For the IF statement, the THEN portion of the statement appeared to be processed more quickly than the ELSE, if the learning effect is taken into account. For the CASE statement, repeated processing of a particular case resulted in reduced reaction times (trials 2, 3 and 5). The order of the cases appeared to have an effect on the reaction times of a case when it was first processed, with the cases listed first having faster initial reaction times.

Of course, the above comments are only speculative since order and learning effects had pronounced effects on the processing of the expressions. Further experiments are needed in which the order of both expressions and digits is counter-balanced to overcome order and learning effects. Also, more digits need to be presented.
Program-Version Analysis

For the program-version analysis, three programs were constructed to reflect increasing selection complexity: program 0 selected one of two assignments to perform, program 1 selected one of three and program 2 selected one of four. Within each program, three versions were constructed to implement the program: nested IF statements (NESTED), the use of AND expressions to simplify the IF-THEN-IF structure of the NESTED version (AND), and a CASE-type selector as described by Keller [15]. Thus, there were nine program versions which are shown in Figures 7,8 and 9.

```
Version 0
IF A>5
  THEN IF B>5
      THEN L=2
      ELSE L=1
  ELSE L=1

Version 1
IF A>5 AND B>5
  THEN L=2
  ELSE L=1

Version 2
CASE
  A>5 AND B>5 DO L=2 END
  NOT(A>5 AND B>5) DO L=1 END
ENDCASE

Figure 7: Versions for Program 0 in the Pilot Study
```
Figure 8: Versions for Program 1 in Pilot Study

Forty-two subjects participated in the experiment. A randomized block, partially confounded factorial design with repeated measures was used [17]. In this design, each subject was randomly assigned to one of six blocks as shown in Table 8. Each subject saw three program versions, the order of the versions being randomized for each subject. The versions for each block were chosen so that each subject saw all three programs and all three versions but not all nine combinations. In this way, the main effects of program and version could be determined while their interaction was confounded with blocks. However, the interaction is only
Version 0

IF A=B
THEN IF A=C
    THEN L=4
    ELSE L=3
ELSE IF A=C
    THEN L=2
    ELSE L=1

Version 1

IF A=B AND A=C
THEN L=4
ELSE IF A=B AND A≠C
THEN L=3
ELSE IF A≠B AND A≠C
THEN L=2
    ELSE L=1

Version 2

CASE
    A=B AND A=C DO L=4 END
    A=B AND A≠C DO L=3 END
    A≠B AND A=C DO L=2 END
    A≠B AND A≠C DO L=1 END
ENDCASE

Figure 9: Versions for Program 2 in the Pilot Study

partially confounded with blocks because the last three blocks are replications of the first three blocks, giving an estimate of the interaction.

For each version, the subject saw a set of input values for six trials. The order of the input data was constant for all subjects. The data were chosen to insure that all version paths were processed at least once and the order of the data was initially randomized to remove any experimenter bias.
TABLE 8
Program-Version Assignments to Blocks

<table>
<thead>
<tr>
<th>Block</th>
<th>Treatment 1</th>
<th>Treatment 2</th>
<th>Treatment 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>12</td>
<td>21</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>10</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>02</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>00</td>
<td>11</td>
<td>22</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>02</td>
<td>21</td>
</tr>
<tr>
<td>5</td>
<td>01</td>
<td>12</td>
<td>20</td>
</tr>
</tbody>
</table>

Entries are listed by program-version.

After each trial, the subjects were given five seconds to determine the function of the program. Then new input values were printed on the screen. At the completion of the trial series, the screen was erased and the subject was asked to write down the function of the program. The subject then proceeded to the next program.

The analysis of variance table is shown in Table 9. The dependent measure for this analysis was the mean of six trials for each version seen. A logarithmic transformation was performed to normalize the data. Both the program factor and the interaction between program and version were significant, while the main effect of versions was not. In
general, these results mean that the programs differed significantly in difficulty but the difficulty of a particular version depended on which program it was seen in.

TABLE 9
Analysis of Variance Table for Programs in Pilot Study

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>MS</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blocks</td>
<td>5</td>
<td>.031</td>
<td>.73</td>
</tr>
<tr>
<td>Subject (Blocks)</td>
<td>36</td>
<td>.043</td>
<td></td>
</tr>
<tr>
<td>Program</td>
<td>2</td>
<td>1.066</td>
<td>47.22**</td>
</tr>
<tr>
<td>Version</td>
<td>2</td>
<td>.013</td>
<td>.57</td>
</tr>
<tr>
<td>Program*Version</td>
<td>4</td>
<td>.090</td>
<td>3.99**</td>
</tr>
<tr>
<td>Residual</td>
<td>76</td>
<td>.023</td>
<td></td>
</tr>
</tbody>
</table>

**p<.01

The trial data for each program are shown in Figure 10. With a few exceptions, the versions in each program tended to run parallel, indicating that there was a consistent difference between versions in each program. The use of AND appeared to simplify the nested IF statement but was more difficult to process when the nesting increased. On the other hand, the CASE statement was inferior for simple
applications but was superior in more complex selection applications. The difficulty of the NESTED version tended to lie between the two extremes of AND and CASE.

It is interesting to compare these empirical results with McCabe's cyclomatic measure and Gordon's Ec. In both theories, the versions should be increasingly difficult within a program because the number of conditions generally increased across versions as well as the number of operands and AND operators. Yet the AND simplified program 0, which agrees with the comments of Myer. Unfortunately, the simplification the AND affords does not extend to the more complex nesting of program 2. In fact, the number of independent paths for all the versions in program 2 was the same yet there were large differences between the versions. Similarly, versions 1 and 2 in program 0 and program 1 had the same cyclomatic number but differed according to the reaction time data. Thus McCabe's measure does not adequately measure the complexity of the versions.

Criticism of Gordon's measure must be more cautious. First, Gordon has noted that his measure is basically statistical in nature and requires programs which are sufficiently large to obtain a reasonable estimate of the parameters in his measure. Second, his measure is supposed
Figure 10: Trial Data for Program-Version Experiment
to measure the time to understand a program. But the task in this experiment may reflect the time to process a program rather than understand it. Still, the time to process a program may, at least partially, indicate the comparative difficulty in understanding the versions. If this is true, Gordon's measure does not order the versions in the same way as they are ordered in the data for any of the three programs.

It should be noted that version 0 in program 0 and program 1 were incorrect. The statement "ELSE L=1" should have been added to version 0 of program 0 and "ELSE L=1" should have been added to version 0 of program 1. Although these errors were not intentional, some interesting results occurred. First, in trial 2 of program 1, the first THEN was executed but the time for version 0 for this trial did not follow the normal pattern of the other versions. The reasons for this behavior are not definite but two explanations are possible. One explanation might be that the subjects were not quite convinced that both IF conditions were true and the extra time resulted from double-checking. Another possibility is that the program was asymmetrical with respect to the ELSE clauses and this caused confusion on the first trial. Unfortunately, a similar asymmetry occurred for version 0 of program 0 but the same behavior as in program 1 was not observed. Since
the same situation as in program 0 occurred on the first trial of program 0, the effect cannot be assessed because of confounding with reading time. One interesting result occurred in trial 3 when the data values given resulted in an indeterminate value for L, and the reaction time was found to be somewhat higher, possibly indicating confusion by the subjects.

The data were analyzed for any clear-cut drop in reaction time which might indicate that comprehension had occurred. But the reaction times for the versions were so highly data-dependent that the point of comprehension could not be distinguished from input data values which were easy to process. Also, although the subjects were told to ignore the program when they knew the function and process the data according to their own "internal" function, many subjects may have continued to hand-simulate the program just to be sure of their answer. A better method of detecting comprehension needs to be found.

Finally, the function statements were analyzed for correctness. However, the functions for program 0 and program 2 could not be simply stated -- they really only set L based on certain conditions -- and were not easily analyzed. For example, program 1 computed the maximum of
three numbers and 7 subjects were not able to state the function: 3 subjects who saw version 1 and 2 subjects each who saw version 0 and version 2. These results indicate that if further experiments require a function statement, it must be easily stated if it is to be evaluated.

Summary

The results can be summarized as follows:

1. The use of motor reaction time as a covariate is not necessary. This is fortunate since collecting the MRT data was time-consuming and tiring for the subjects.

2. The methodology used to rank the simple expressions appears sound. Recommended improvements include: (1) presenting each expression before the hand-simulation begins to eliminate reading time confounding the first trial, (2) presenting more digits and (3) counter-balancing the order of both expressions and digits.

3. Immediate feedback needs to be given when the subject touches a key on the screen. Because the touch panel will beep whenever the screen is touched, no matter what the location, some sub-
jects assumed that their answer was correct when they heard the beep. When nothing happened for several seconds, the subjects realized their error and reentered the answer, resulting in greater response variances.

4. The suggested methodology does not appear to adequately evaluate the comprehensibility of programs. However, it can distinguish between programs based on their computability. Another measure needs to be devised.

5. The programs must have easily stated functions if function statements are to be used to measure comprehension.
THE FINAL STUDY

The results of the third chapter indicate that a better method of evaluating program comprehension than the method proposed in the third chapter is needed. Therefore, a revised measure of comprehension is proposed below. A study is then described in which the simple expressions are reevaluated using the suggestions of the third chapter and a new set of programs are evaluated using the revised measure of comprehension.

A Revised Measure of Comprehension

There were several reasons why the measure of comprehension proposed in the third chapter was not able to detect the occurrence of comprehension. First, some data values were more easily processed than others making it difficult to distinguish true comprehension from easily-processed data. Second, subjects may have tended to hand-simulate the program after they knew the function just to be sure of their answer or because they had become accustomed to the hand-simulation mode of processing. It was also possible that the hand-simulation task actually interfered with the
understanding process. If so, the proposed measure would not truly assess the comprehensibility of the programs. To overcome the above difficulties, a revised measure of comprehension was needed to avoid the use of hand-simulation.

In reaction time research, it is assumed that if one statement is more comprehensible than another, it will take a subject less time to process it. If this reasoning is extended to the evaluation of two versions of a program, the version which takes less time to process will be more comprehensible.

But what does it mean to "process" a program? Processing a program could mean hand-simulating a program, but as we have seen, a program can be hand-simulated without understanding it. A better method would require a subject to determine the function of the program, measuring the time the subject takes to deduce the program's function. If two versions exhibited the same function, the version which revealed its function more quickly would be judged more comprehensible.

Ideally, of course, all the subjects would be able to state the function of the version they saw. Since this is unlikely to happen under normal circumstances, a modification to the procedure described above is necessary. The
modification chosen is described as follows, with the data analysis being divided into two parts. In the first part, the reaction time data are still analyzed as described above, but the times now represent the time until the subject "thinks" he understands the function. The second part is an analysis of the correctness of the function statements, yielding information regarding the difficulty of the programs and versions.

There are both advantages and disadvantages to the modified approach. First, the subjects can use whatever understanding process they typically use to comprehend the program. Thus the experimental task requires behavior which the subject normally uses to comprehend a program. Second, the experimental procedure is easier to administer than the procedure described in the third chapter and can be administered without a computer. The disadvantage of the modified procedure is that the reaction time data might be biased by subjects who cannot correctly state the function of a program. However, the functional statement analysis can provide the information necessary to cross-validate the reaction time data.
Procedure

The same room, terminals and basic procedures were used in the present study as were used in the pilot study. The experimental program consisted of three sections. In the first section, the operation of the terminal was described, the background information was collected, the notation used in the experiment was detailed and a practice reaction test was given. The description of the notation for the simple expressions was shortened from that given in the pilot study to a table of the symbols used in the experiment. Only for the IF and CASE expressions were examples given and practice statements evaluated. The practice reaction test consisted of presenting the digits 1 through 9 three times in random order. For each digit, the subject was required to touch the appropriate square on the screen in which the digit was displayed. The format of the numerical display was identical to the one used in the Pilot study, but whenever a box containing a digit was touched, the box border flashed to provide immediate feedback on the location of the touched box. The feedback helped to notify the subject when an answer was not accepted by the computer and helped reduce the variance of the responses. A similar procedure was conducted for true and false answers.
The second section of the experiment was the evaluation of the simple expressions. The same basic procedure as described in the third chapter was used, with the following exceptions:

1. The order of the expressions was randomized for each subject. However, all logical expressions were presented first followed by the expressions requiring numerical answers to reduce the problem of changing screen displays.

2. To avoid reading time confounding the first trial reaction time, each expression was displayed before the trial series for that expression began. After the expression had been read, the subject began the trial series by touching the screen.

3. The order of the digits was randomized for each expression and subject. Ten digits were displayed (0-9). Both reaction times and errors were recorded.

4. The logical expressions were chosen so that an equal number of true and false answers were given. In the case of EQ and NE, eight additional fives had to be presented to maintain the same number of
true and false answers. The selection statements were chosen so that all paths were traversed an equal number of times.

The third section consisted of three programs of three versions each. As in the pilot study, each subject saw each version and program but not all program versions. Prior to displaying the first program, the procedure was described to the subject by the experimental program. A practice example of a program was presented for which the subject had to describe the function. The example program computed the mean of three values. After the example, the subject was presented with the first program.

For each program, the appropriate version for the subject was displayed on the screen and the subject was directed to determine the function of the program. When the subject thought he knew the function, he touched the screen and the version was erased. The subject was then directed to write down the function as quickly as possible on paper provided by the experimenter. When the subject touched the screen again, the next program was displayed. When all three programs were presented, the session was terminated.
Subjects

The subjects were both graduate and undergraduate students at Iowa State University. The graduate students were volunteers from an operating systems course and programming language course, while the undergraduate subjects were volunteers from a beginning programming language course. The same incentives described in the third chapter were used in the present experiment, except that the graduate students were not offered the grade incentive. Thirty-four undergraduate and 15 graduate students participated. A summary of the subjects' background data is shown in Table 10.

Experimental Design and Results

The entire experiment actually consisted of two experiments: simple expression analysis and program-version analysis. To simplify the description of the experiments, the appropriate design and results for each experiment will be described in separate sections.

Simple Expression Analysis

The statistical design used for the simple expression analysis was a split-plot factorial with two repeated measures: expressions (9) and digits (10). The expressions used in the experiment are shown in Table 11. The logical
TABLE 10
Subject Background Data for Final Study

<table>
<thead>
<tr>
<th>Measure</th>
<th>N</th>
<th>Mean</th>
<th>S.D.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Months Programming</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Undergraduate</td>
<td>34</td>
<td>26.85</td>
<td>20.37</td>
</tr>
<tr>
<td>Graduate</td>
<td>15</td>
<td>39.26</td>
<td>29.51</td>
</tr>
<tr>
<td>Size of Largest Program¹</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Undergraduate</td>
<td>34</td>
<td>3.41</td>
<td>.66</td>
</tr>
<tr>
<td>Graduate</td>
<td>15</td>
<td>3.47</td>
<td>.64</td>
</tr>
<tr>
<td>Time of Session²</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Undergraduate</td>
<td>34</td>
<td>35.32</td>
<td>18.59</td>
</tr>
<tr>
<td>Graduate</td>
<td>15</td>
<td>27.87</td>
<td>6.53</td>
</tr>
</tbody>
</table>

¹Subjects were asked to classify the size of the largest program as less than 10,100,1000 or 10,000 lines. Thus, the measure is in terms of log(lines).

²in minutes.

expressions were evaluated first, the order being randomized for each subject. The selection statements were then presented in random order for each subject. The order of the digits was randomized for each expression and subject.

The analysis of variance for the logical expressions and selection statements are shown in Tables 12, 13 and 14, respectively. As can be seen, the main effect of experience was not significant nor were the interactions involving experience, although the interaction with the logical expressions LT, GT, AND, OR and NOT approached significance.
### TABLE 11

**Expressions Used in Final Study**

<table>
<thead>
<tr>
<th>Symbol used in Text</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQ</td>
<td>(X=5)</td>
</tr>
<tr>
<td>NE</td>
<td>(X\neq 5)</td>
</tr>
<tr>
<td>GT</td>
<td>(X&gt;4)</td>
</tr>
<tr>
<td>LT</td>
<td>(X&lt;5)</td>
</tr>
<tr>
<td>AND</td>
<td>(X&gt;2) AND (X&lt;8)</td>
</tr>
<tr>
<td>OF</td>
<td>(X&gt;3) OR (X&lt;2)</td>
</tr>
<tr>
<td>NOT</td>
<td>NOT ((X&lt;5))</td>
</tr>
<tr>
<td>IF</td>
<td>IF (X&lt;5) THEN (Y=1); ELSE (Y=2);</td>
</tr>
<tr>
<td>CASE</td>
<td>CASE (X\geq 0) AND (X&lt;2) DO; (Y=4); END; (X\geq 2) AND (X&lt;4) DO; (Y=2); END; (X\geq 4) AND (X&lt;6) DO; (Y=3); END; (X\geq 6) AND (X&lt;8) DO; (Y=1); END; ELSE DO; (Y=5); END; ENDCASE;</td>
</tr>
</tbody>
</table>

1The values for \(X\) were the digits 0-9. The order of the digits was randomized for each expression and subject.

The ordering of the logical expression was similar to the ordering found in the pilot study, except that in the present experiment OR was processed significantly slower than NOT, and GT was processed significantly faster than LT. No differences were found between AND and LT. As in the


### TABLE 12

**Statistical Analysis for LT, GT, AND, OR and NOT**

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>MS</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experience</td>
<td>1</td>
<td>1.060</td>
<td>.76</td>
</tr>
<tr>
<td>Subject (Experience)</td>
<td>47</td>
<td>1.400</td>
<td></td>
</tr>
<tr>
<td>Expression</td>
<td>4</td>
<td>3.325</td>
<td>14.82**</td>
</tr>
<tr>
<td>Experience*Expression</td>
<td>4</td>
<td>.530</td>
<td>2.36</td>
</tr>
<tr>
<td>Subject*Expression (Experience)</td>
<td>188</td>
<td>.224</td>
<td></td>
</tr>
<tr>
<td>Digit</td>
<td>9</td>
<td>.368</td>
<td>4.43**</td>
</tr>
<tr>
<td>Experience*Digit</td>
<td>9</td>
<td>.063</td>
<td>.76</td>
</tr>
<tr>
<td>Subject*Digit (Experience)</td>
<td>423</td>
<td>.083</td>
<td></td>
</tr>
<tr>
<td>Expression*Digit</td>
<td>36</td>
<td>.318</td>
<td>4.15**</td>
</tr>
<tr>
<td>Experience<em>Expression</em>Digit</td>
<td>36</td>
<td>.056</td>
<td>.73</td>
</tr>
<tr>
<td>Subject<em>Expression</em>Digit (Experience)</td>
<td>1692</td>
<td>.077</td>
<td></td>
</tr>
</tbody>
</table>

\(^1\) The Duncan test for the means (in seconds) was (means not significantly different are underlined):

<table>
<thead>
<tr>
<th>GT</th>
<th>LT</th>
<th>AND</th>
<th>NOT</th>
<th>OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>.842</td>
<td>1.021</td>
<td>1.043</td>
<td>1.080</td>
<td>1.429</td>
</tr>
</tbody>
</table>

**p<.01

pilot study, IF was processed significantly faster than CASE. The differences in results between experiments can probably be attributed to differences in experimental procedures.

The reaction times by digit for each expression are shown in Figure 11. (Recall that eight additional fives had to be presented for EQ and NE to ensure an equal number of
true and false responses. The additional fives in the graph are listed in the order they were presented.) The most dramatic feature of these graphs can be described by the following statement: when two digits were compared, the time to carry out the comparison was greater when the digits were the same than when they were different. This phenomenon was present in all the expressions, most notably in the EQ and NE expressions. A second general phenomenon was that the trials in which the expression evaluated true were processed
TABLE 14

Statistical Analysis for IF and CASE

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>MS</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experience</td>
<td>1</td>
<td>.231</td>
<td>.48</td>
</tr>
<tr>
<td>Subject (Experience)</td>
<td>47</td>
<td>.479</td>
<td></td>
</tr>
<tr>
<td>Expression¹</td>
<td>1</td>
<td>42.838</td>
<td>212.11**</td>
</tr>
<tr>
<td>Experience*Expression</td>
<td>1</td>
<td>.131</td>
<td>.65</td>
</tr>
<tr>
<td>Subject*Expression (Experience)</td>
<td>47</td>
<td>.202</td>
<td></td>
</tr>
<tr>
<td>Digit</td>
<td>9</td>
<td>.148</td>
<td>2.35**</td>
</tr>
<tr>
<td>Experience*Digit</td>
<td>9</td>
<td>.019</td>
<td>.33</td>
</tr>
<tr>
<td>Subject*Digit (Experience)</td>
<td>423</td>
<td>.059</td>
<td></td>
</tr>
<tr>
<td>Expression*Digit</td>
<td>9</td>
<td>.146</td>
<td>2.58**</td>
</tr>
<tr>
<td>Experience<em>Expression</em>Digit</td>
<td>9</td>
<td>.038</td>
<td>.68</td>
</tr>
<tr>
<td>Subject<em>Expression</em>Digit (Experience)</td>
<td>423</td>
<td>.056</td>
<td></td>
</tr>
</tbody>
</table>

¹The means (in seconds) for IF and CASE were 1.157 and 3.03, respectively.
**p<.01

significantly faster than when the expression evaluated false. However, no significant differences were found for the operations involving negation (NOT and NE). The statistical analyses are shown in the Appendix.

The split hypothesis discussed in the last chapter was clearly exhibited for the LT and GT trial curves. As predicted by the hypothesis, the reaction times increased with decreased split, resulting in a tent-like shape for the
Figure 11: Reaction Times by Digit for Expressions in Final Study
trial curves. However, the split hypothesis did not extend to the comparison operations of EQ and NE. Reaction times tended to be uniformly low except when the split was zero, possibly indicating a different processing mechanism for NE and EQ comparisons.

The effect of negation was also noteworthy. Because the expression "NOT(X<5)" is functionally equivalent to "X>4", the NOT curve might be expected to closely resemble the GT curve. While the two curves were similar, the largest reaction time occurred for the digit 5. It is likely that subjects processed the expression as "X>=5", which would explain the above results.

Similarly, "X#5" is functionally equivalent to "NOT(X=5)". Thus, if the NE expression was processed as EQ and then negated, the NE and EQ curves should be parallel with NE taking longer than EQ for each digit. While both parallelism and increased processing time for NE were noted, the results were not sufficiently compelling to confirm that NE was processed as EQ and negated.

The experiment also provided evidence of the processing of AND and OR. For the expression "X>2 AND X<8", it was unnecessary to evaluate the second sub-expression when the first sub-expression was false. Therefore, the reaction
times for the digits 0, 1 and 2 might be expected to be less than for 8 and 9 if the above strategy were used. Yet there was no indication for this type of strategy. On the other hand, subjects appeared to terminate processing of the expression "\(x < 3 \text{ OR } x > 7\)" when the first sub-expression evaluated true. Using this strategy, the reaction times for the digits 0, 1 and 2 were less than the reaction times for the digits 8 and 9.

The trial curves for the IF and CASE data indicate the significant effect the evaluation of the simple expressions had on the processing of the selection statements. The trial curve for the IF expression shows an almost identical form to the LT expression. Since the condition of the IF expression was the LT expression, this result probably means that the primary contributor to the processing time of the IF statement is the processing of the condition of the IF. For the CASE statement, the only differences in reaction times for the individual cases occurred because of the zero split factor previously discussed, although not all possible zero splits resulted in larger reaction times. More importantly, it does not appear that the cases were searched linearly until a true condition was found, otherwise the trial curve would have increased as the value of the digits increased. Thus, additional cases may not appreciably affect the processing time of the CASE statement.
When the subject made an error in a trial, the time was recorded, the error noted and the subject was asked to reenter the correct answer. The errors were analyzed for all expressions but no discernible trends could be found. The error rate was 4.5 per cent.

Program-Version Analysis

Three programs were constructed to reflect increasing selection complexity: a program to compute the minimum of three numbers, a program to compute the maximum of a series of numbers and a program which merges two sorted arrays into a third array. For all three programs, version 0 was written using nested IF statements while version 2 was written using a CASE statement. Version 1 was written as a sequence of IF statements for the first two programs and as an IF..ELSE IF structure for the third program. These programs are shown in Figures 12, 13, and 14.

34 undergraduate students and 15 graduate students participated in the experiment. For each group, a randomized block, partially confounded factorial design with repeated measures was used. As in the pilot study, subjects were randomly assigned to one of the six groups, each group seeing three program versions.
Figure 12: Program 0: Three Versions to Compute the Minimum

For each program version, the subject was asked to determine the function of the program. When the function was determined, the subject touched the screen, the time was recorded, the screen was erased, and the subject was directed to write down the function of the program. The next program was displayed when the subject touched the screen again.
Version 0

DCL (X,Y) FIXED DEC;
Y=0;
NEXT: GET LIST(X);
IF X>=0
THEN IF X>Y
    THEN Y=X;
    ELSE GOTO NEXT;
ELSE GOTO FINISH;
GOTO NEXT;
FINISH: PUT LIST(Y);

Version 1

DCL (X,Y) FIXED DEC;
Y=0;
NEXT: GET LIST(X);
IF X>Y
THEN Y=X;
IF X>=0
THEN GOTO NEXT;
PUT LIST(Y);

Version 2

Y=0;
NEXT: GET LIST(X);
CASE
  X>Y DO; Y=X; GOTO NEXT; END;
  X<0 DO; GOTO FINISH; END;
  ELSE DO; GOTO NEXT; END;
ENDCASE;
FINISH: PUT LIST(Y);

Figure 13: Program 1: Three Versions to Compute the Maximum

The analysis of variance tables for the undergraduates and graduates are shown in Tables 15 and 16, respectively. The dependent measure for these analyses was the time to understand the program. The data were normalized using a logarithmic transformation. The tables show that both program and version factors were significant for the
Version 0

```
DCL (A(N), B(M), C(N+M)) FIXED DEC,
    (I, J, K) FIXED DEC;
/* ARRAYS A AND B ARE ASSUMED TO BE SORTED IN
ASCENDING ORDER */
I=1; J=1; K=1;
DO WHILE ( K<= N+M);
    IF I<=N
        THEN IF A(I)<B(J)
                THEN DO; C(K)=A(I); I=I+1; END;
                    ELSE DO; C(K)=B(J); J=J+1; END;
            ELSE DO; C(K)=B(J); J=J+1; END;
        K=K+1;
    END;
```

Version 1

```
I=1; J=1; K=1;
DO WHILE ( K<= N+M);
    IF I>N
        THEN DO; C(K)=B(J); J=J+1; END;
    ELSE IF J>M
        THEN DO; C(K)=A(I); I=I+1; END;
            ELSE IF A(I)<B(J)
                    THEN DO; C(K)=A(I); I=I+1; END;
                        ELSE DO; C(K)=B(J); J=J+1; END;
        K=K+1;
    END;
```

Version 2

```
I=1; J=1; K=1;
DO WHILE ( K<= N+M);
    CASE
        I<=N AND J<=M DO; IF A(I)<B(J)
                THEN DO; C(K)=A(I); I=I+1; END;
                    ELSE DO; C(K)=B(J); J=J+1; END;
            END;
        I>N DO; C(K)=B(J); J=J+1; END;
        J>M DO; C(K)=A(I); I=I+1; END;
    ENDCASE;
K=K+1;
END;
```

The declarations and comments were present for all versions of program 2.

Figure 14: Program 2: Three Versions to Merge Two Sorted Arrays
undergraduates, while no significant factor differences were found for the graduates. No interactions were significant.

**TABLE 15**

Statistical Analysis of Undergraduate Program Data in Final Study

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>MS</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blocks</td>
<td>5</td>
<td>.036</td>
<td>.86</td>
</tr>
<tr>
<td>Subject (Blocks)</td>
<td>28</td>
<td>.0944</td>
<td></td>
</tr>
<tr>
<td>Program</td>
<td>2</td>
<td>1.451</td>
<td>34.95**</td>
</tr>
<tr>
<td>Version</td>
<td>2</td>
<td>.276</td>
<td>6.64**</td>
</tr>
<tr>
<td>Program*Version</td>
<td>4</td>
<td>.009</td>
<td>.23</td>
</tr>
<tr>
<td>Residual</td>
<td>60</td>
<td>.042</td>
<td></td>
</tr>
</tbody>
</table>

**p<.01

The results of the undergraduate experiment were clear-cut. As can be seen in Table 17, the comprehensibility of the programs clearly differed, with program 0 being the most comprehensible and program 2 the least. The ordering of the versions was the same for all three programs, with version 2 being the easiest and version 0 being the hardest to
**TABLE 16**

Statistical Analysis of Graduate Program Data in Final Study

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>MS</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blocks</td>
<td>5</td>
<td>.073</td>
<td>.35</td>
</tr>
<tr>
<td>Subject (Blocks)</td>
<td>9</td>
<td>.206</td>
<td></td>
</tr>
<tr>
<td>Program</td>
<td>2</td>
<td>.124</td>
<td>2.37</td>
</tr>
<tr>
<td>Version</td>
<td>2</td>
<td>.049</td>
<td>.93</td>
</tr>
<tr>
<td>Program*Version</td>
<td>4</td>
<td>.115</td>
<td>2.19</td>
</tr>
<tr>
<td>Residual</td>
<td>22</td>
<td>.052</td>
<td></td>
</tr>
</tbody>
</table>

understand. The use of the CASE statement apparently simplified all three programs for the undergraduate subjects while the nested IF statements made the programs difficult to understand.

However, these results must be interpreted within the context of the number of function statements which were stated incorrectly. In Table 18, the number of incorrectly stated functions are shown by program and version for all subjects. A total of 36 out of 147 function statements were incorrect. Based on the number of incorrectly stated functions, the CASE version was superior to the other versions for programs 0 and 2, but inferior for program 1.
### TABLE 17
Means of Program Versions for Undergraduates

<table>
<thead>
<tr>
<th>Program</th>
<th>Version 0</th>
<th>Version 1</th>
<th>Version 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>57.943</td>
<td>56.624</td>
<td>46.174</td>
</tr>
<tr>
<td>1</td>
<td>78.163</td>
<td>64.938</td>
<td>60.549</td>
</tr>
<tr>
<td>2</td>
<td>199.526</td>
<td>125.170</td>
<td>93.046</td>
</tr>
</tbody>
</table>

1In seconds

### TABLE 18
Summary of Function Misstatements

<table>
<thead>
<tr>
<th>Program</th>
<th>Version 0</th>
<th>Version 1</th>
<th>Version 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Undergraduates: 5, Graduates: 1</td>
<td>Undergraduates: 4, Graduates: 1</td>
<td>Undergraduates: 1, Graduates: 0</td>
</tr>
<tr>
<td>1</td>
<td>Undergraduates: 3, Graduates: 0</td>
<td>Undergraduates: 7, Graduates: 1</td>
<td>Undergraduates: 8, Graduates: 1</td>
</tr>
<tr>
<td>2</td>
<td>Undergraduates: 3, Graduates: 0</td>
<td>Undergraduates: 1, Graduates: 0</td>
<td>Undergraduates: 0, Graduates: 0</td>
</tr>
</tbody>
</table>
The types of errors the subjects made in stating the functions were revealing. For program 0, 7 of 12 incorrect statements described the program as finding the maximum rather than the minimum: 4 in version 0 and 3 in version 1. A possible reason for version 0 being misleading was that the tests were "greater than" rather than "less than". The direction of the tests may have given the impression that a maximum was being calculated. Version 1 relied on the implicit sequential processing of statements to carry out its function. Subjects who failed to recognize the importance of the sequential processing used in this version may have been more likely to misstate the function. For program 1, the most prevalent error occurred in version 1 when 4 subjects thought that the version was performing a sorting operation. The errors for version 2 were not so easy to classify, but 3 subjects thought that it printed the first number greater than zero while 2 subjects thought that it printed the last number less than zero.

A possible explanation for the error and reaction time results comes from the previously cited work of Sme, Green and Guest [31]. According to their theory, the process of comprehending a program involves the conversion of the program's sequence information -- the order in which the actions execute -- and taxon information -- the conditions
under which the actions execute -- into a functional statement which is likely to be taxonomic in nature. For example, program 0 was mostly taxonomic since the value printed was based on relationships in the data and requires little sequencing of actions. The CASE statement was superior for this application because it conveys the taxon information without additional sequence information. Program 2 was similar in nature, with the loop providing the only essential sequential information. Again, the CASE version was superior based on both time to understand and number of functional errors committed. However, neither version 1 or 2 in program 1 explicitly stated all the important taxon information for this program. If the conditions of the CASE statement been written as 'X\geq0 \AND \ X>y\prime', 'X\geq0 \AND \ X\leq Y\prime' and 'X<0\prime', the CASE version would likely have been more comprehensible.

It was possible that the subjects who misstated the functions could have biased the reaction time analysis. Thus, the reaction time data were reanalyzed, using only the data from subjects who correctly stated the program functions. The results are shown in Tables 19, 20 and 21. The results of the undergraduate reanalysis were similar to the previous results, while the results for the graduate students showed a significant interaction between the
programs and versions. The effect of programs for the graduate students also approached significance.

### TABLE 19

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>MS</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blocks</td>
<td>5</td>
<td>0.029</td>
<td>.43</td>
</tr>
<tr>
<td>Subject (Blocks)</td>
<td>28</td>
<td>0.069</td>
<td></td>
</tr>
<tr>
<td>Program</td>
<td>2</td>
<td>0.917</td>
<td>17.20**</td>
</tr>
<tr>
<td>Version</td>
<td>2</td>
<td>0.213</td>
<td>4.00*</td>
</tr>
<tr>
<td>Program*Version</td>
<td>4</td>
<td>0.034</td>
<td>.63</td>
</tr>
<tr>
<td>Residual</td>
<td>28</td>
<td>0.053</td>
<td></td>
</tr>
</tbody>
</table>

*p<.05  
**p<.01
Although the program-version means changed for the undergraduates, the CASE version was still more comprehensible than the other versions for programs 0 and 2. However, the reanalysis of the graduate data provided some new results. The CASE version was superior for the minimum program, the sequential If version for the maximum program, and the nested IF version for the merge program. It is
**TABLE 20**

Statistical Analysis of Graduate Program Data in Final Study Using only Correct Response Data

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>MS</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blocks</td>
<td>5</td>
<td>.117</td>
<td>.58</td>
</tr>
<tr>
<td>Subject (Blocks)</td>
<td>9</td>
<td>.200</td>
<td></td>
</tr>
<tr>
<td>Program</td>
<td>2</td>
<td>.142</td>
<td>3.38</td>
</tr>
<tr>
<td>Version</td>
<td>2</td>
<td>.051</td>
<td>1.21</td>
</tr>
<tr>
<td>Program*Version</td>
<td>4</td>
<td>.152</td>
<td>3.61*</td>
</tr>
<tr>
<td>Residual</td>
<td>18</td>
<td>.042</td>
<td></td>
</tr>
</tbody>
</table>

*p < .05

possible that the experienced subjects were more familiar with IF statements, making the versions with IF statements more comprehensible for the more complex programs. Further testing will be required to determine the effects of experience.

The theoretical measures proposed by McCabe and Gordon were calculated for the programs in this experiment and are shown in Table 22. As can be seen, the cyclomatic measure did not predict the ordering of the versions found in the experiment, while Gordon's measure was only able to predict the ordering in program 1. The reason can again be
### TABLE 21

Means of Program Versions for Undergraduates and Graduates Using only Correct Response Data

<table>
<thead>
<tr>
<th>Program</th>
<th>Version 0</th>
<th>Version 1</th>
<th>Version 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Undergraduate</td>
<td>Graduate</td>
<td>Undergraduate</td>
</tr>
<tr>
<td>0</td>
<td>54.094</td>
<td>72.295</td>
<td>59.850</td>
</tr>
<tr>
<td>1</td>
<td>81.343</td>
<td>44.671</td>
<td>72.063</td>
</tr>
<tr>
<td>2</td>
<td>230.092</td>
<td>122.144</td>
<td>93.043</td>
</tr>
</tbody>
</table>

1In seconds

explained by the sequence/taxon theory. In the previously reported study of Sime et al. [31], it was found that if redundant tests were included as part of the ELSE portion of the IF statements, the program was easier to debug. In the present experiment, the form of the CASE statement required redundant conditions so that only one of many cases was chosen. Subsequently, the CASE version was more comprehensible, at least for programs 0 and 2. But adding redundant conditions in a program can increase the cyclomatic measure and will increase Ec because more operators and operands are present. Therefore, these
<table>
<thead>
<tr>
<th>Program</th>
<th>Version 0</th>
<th>Version 1</th>
<th>Version 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Ec^1 732</td>
<td>320</td>
<td>1095</td>
</tr>
<tr>
<td></td>
<td>C 2 4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>Ec 1644</td>
<td>935</td>
<td>1540</td>
</tr>
<tr>
<td></td>
<td>C 3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Ec 13539</td>
<td>13204</td>
<td>16382</td>
</tr>
<tr>
<td></td>
<td>C 5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

^1Gordon's measure  
^2McCabe's measure

measures will predict that a program with redundant conditions inserted to increase the taxonomic information will be less comprehensible than a program without the redundant conditions. Yet the increased taxonomic information will in fact increase the comprehensibility. Thus the real factor affecting comprehensibility may be how well a program can convey both its sequence and taxonomic information, rather than a simple count of operators, operands or conditions.
Summary

The results of the simple expression analysis can be summarized as follows:

1. For all comparison operations, it was significantly more difficult to compare two numbers when they were equal than when they were not.

2. For all simple expressions which did not involve negation, it was significantly easier to process an expression which evaluated true than one which evaluated false. No differences were found between processing true and false expressions for expressions involving negation.

3. The ordering of increasing difficulty for the operators and selection statements was:
   \[ \text{EQ, NE, GT, LT, AND, NOT, IF, OR, CASE} \]

4. The larger the difference between two numbers, the faster a subject determined if one number was less than or greater than another. This phenomenon did not appear to extend to EQ and NE.

5. No evidence exists that subjects terminated processing when the first sub-expression of an AND was false. However, the data did suggest that sub-
jects terminated processing when the first sub-expression of an OR was true.

6. The evidence suggests that subjects converted the expression involving negation to its functionally equivalent form. The NE expression took longer to process than the EQ expression for most digits but the mean trial times for the two expressions were not significantly different.

7. The IF condition dominated the processing of the IF. For the CASE, the time to process a particular case did not appear to depend on where the case was located in the CASE statement, at least for the statement seen in the experiment.

The simple expressions were analyzed to obtain information about how subjects processed programs and where the processing difficulties lay. The intent was not to provide practical suggestions for programmers. However, some of the results can be practically applied. For example, since the OR expression appears to be difficult to process, programmers should avoid using it when possible by negating the OR expression to produce an AND expression. Unfortunately, few languages have complementary statements which allow either an expression or its complement to be tested. An example of
complementary statements might be 'DO WHILE' and 'REPEAT UNTIL' where both tests are at the top of a loop. Therefore, language designers should consider including complementary statements in languages they create. Another suggestion to programmers is to avoid the use of the NOT operator since it is difficult to process. Finally, the execution of expressions should correspond to the way programmers process expressions to avoid confusion for the programmer. For example, if the array A has N elements, many languages will report an out-of-bounds error if a statement like "IF I>N OR A(I)>5" is used (assuming that I is greater than N). Yet according to the experimental results, subjects may not detect this error because their evaluation of the expression will cease when I is greater than N. Of course, the above comments relate to simplifying hand-simulation tasks which are encountered in debugging and code walkthroughs. The suggestions may or may not apply to the comprehension of a program.

The results from the program-version analysis indicate that the programs and versions used in this study differed significantly for the undergraduate subjects, while the interaction of programs and versions was significant for the graduate students. Of the three versions for the undergraduates, the CASE statement was most quickly understood and
resulted in the fewest function misstatements in two of the three programs. The large number of function misstatements for the CASE and sequential IF versions of program 1 may have occurred because of the additional sequence information which needed to be conveyed to understand the program. For the graduate students, the CASE version was most comprehensible for the minimum program, the sequential version for the maximum program, and the nested IF version for the merge program. Further research needs to be conducted to investigate the effects of experience.

The actual time to understand each of the versions was compared to the cyclomatic measure of complexity and Gordon's effort measure. The cyclomatic measure did not predict the ordering of the versions for any of the programs, while Gordon's measure was only able to predict the ordering for program 1. It was suggested that the sequence/taxon theory proposed by Sime et al. [31] may account for the above results. According to their theory, the act of program comprehension involves the assimilation of the sequence and taxon information in a program into a meaningful mental representation. By clearly displaying either type of information in the program (e.g. using the CASE to display taxon information), the comprehensibility of the program will be improved. Further research needs to be conducted to test the validity of this statement.
SUMMARY AND CONCLUSIONS

As the use of computers pervades more aspects of our society, it becomes essential that computer software improve in quality. Without software improvements, the role of the computer may have a negative, if not dangerous, impact on the individuals in our society. Only through research in the programming process can the needed improvements in software be achieved.

Program comprehension is one of several programming activities which has been studied by researchers. One active area of comprehension research has been the development of comprehension measures. While considerable progress has been made, difficulties with the present measures of comprehension still exist. For example, two measures have never been validated using human subject data and several others are subjective or are based on unproven theories. Either the present measures must be validated or new measures must be developed.

Another active area of research has been the study of factors affecting comprehension, such as control flow, para-
graphing and commenting. However, little research has been conducted at the statement or expression level to determine factors affecting comprehension at this more elementary level. If we are to truly understand how people process programs, we must first look at how they process the expressions and statements of the program.

The present research was conducted to achieve three goals: (1) to provide information about the human processing of simple expressions, (2) to develop a methodology which can assess the comprehensibility of programs, and (3) to compare the results of an analysis using this methodology with the results predicted by the measures reported in the literature.

The following results were obtained:

1. The difficulty level of the simple expressions and selection statements was as follows (from easiest to hardest):
   \[ =Q,NE,GT,LT,AND,NOT,IF,OR,CASE \]

2. Comparing two numbers which are the same is significantly more difficult than if they are different
3. True expressions are processed significantly faster than false expressions.

4. Measuring the time it takes a subject to discover the function of a program provides an adequate measure of comprehension.

5. For the undergraduate subjects, the CASE version used in the final study was the most comprehensible version for two of the three programs. However, more function misstatements occurred for the CASE and sequential IF version of the program which consisted primarily of sequential information. For the graduate students, each of the versions was superior in comprehensibility for one of the three programs.

6. McCabe's cyclomatic measure did not predict the ordering of version comprehensibility in any of the programs, while Gordon's Ec only predicted the ordering of versions in program 1.

**Recommendations for Future Work**

Although the reaction time measure developed in this study adequately assessed comprehensibility, two difficul-
ties arose in the use of the methodology. First, a large number of subjects were unable to determine the function of the program, making the interpretation of the reaction time data difficult. Therefore, changes in the measure are needed to insure that all of the subjects are able to discover the function. Second, finding functions which all subjects can recognize is not an easy task. The methodology needs to be modified to insure that all subjects are at least familiar with the function presented in the program.

This study had demonstrated that the comprehension measures proposed by McCabe and Gordon could not adequately distinguish the comprehensibility of programs. However, this study did provide support for the sequence/taxon theory proposed by Sime, Green and Guest. According to their theory, the drafting of a program requires the conversion of taxon information to sequence information, while the process of comprehension involves the reverse process. Using the CASE statement, it was possible to specify the taxon information directly, so that no conversion was required. This apparently resulted in more comprehensible programs. The task now is to develop specific tests of the taxon/sequence theory, possibly using the revised methodology described above. If the theory is correct, it could have wide implications for the programming process and programming language development.
Conclusion

By studying some of the factors affecting program comprehension, it has been possible to make some practical suggestions to both programmers and language designers for improving software. But more importantly, the study of comprehension has led to a theory about the programming process which is based on human subject data. It is appropriate that the object of research be the human rather than the machine, since the important improvements in software will only occur when software tools are developed to accommodate those who actually use them.
ACKNOWLEDGEMENTS

The author would like to thank the following people for making this dissertation possible. First, Dr. Roy Keller, the author's major professor, for his insights into the programming process and his support and comments during the preparation of this dissertation. Second, the other members of the author's committee: Dr. Rex Thomas, for his critical comments and gentle prodding; Dr. Arthur Oldehoeft and Dr. Roy Zingg, for their encouragement of and interest in this research; and Dr. Clair Maple for his continued financial support. Third, the many students who participated in the experiments as well as Dr. Kafura, Dr. KrishnaSwamy and Mr. Steve Allen, for allowing the use of their students as subjects. Fourth, Dr. Strahan and Dr. Warren for their statistical assistance. Finally, to my wife, Vicki, for her emotional support and her help in making this dissertation comprehensible.
APPENDIX A. STATISTICAL ANALYSIS OF TRUE AND FALSE DATA FOR EXPRESSIONS

TABLE 23
Statistical Analysis of True and False Data for EQ

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>MS</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experience</td>
<td>1</td>
<td>.159</td>
<td>.14</td>
</tr>
<tr>
<td>Subject (Experience)</td>
<td>47</td>
<td>1.156</td>
<td>4.75*</td>
</tr>
<tr>
<td>Truth¹</td>
<td>1</td>
<td>.578</td>
<td>4.75*</td>
</tr>
<tr>
<td>Experience*Truth</td>
<td>1</td>
<td>.000</td>
<td>0.00</td>
</tr>
<tr>
<td>Residual</td>
<td>831</td>
<td>.122</td>
<td></td>
</tr>
</tbody>
</table>

¹The means (in seconds) for True and False were .663 and .745, respectively.
* p<.05
TABLE 24

Statistical Analysis of True and False Data for NE

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>MS</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experience</td>
<td>1</td>
<td>1.909</td>
<td>2.34</td>
</tr>
<tr>
<td>Subject (Experience)</td>
<td>47</td>
<td>.817</td>
<td></td>
</tr>
<tr>
<td>Truth¹</td>
<td>1</td>
<td>.006</td>
<td>.06</td>
</tr>
<tr>
<td>Experience*Truth</td>
<td>1</td>
<td>.053</td>
<td>.54</td>
</tr>
<tr>
<td>Residual</td>
<td>831</td>
<td>.099</td>
<td></td>
</tr>
</tbody>
</table>

¹The means (in seconds) for True and False were .761 and .771, respectively.
<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>MS</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experience</td>
<td>1</td>
<td>.368</td>
<td>.81</td>
</tr>
<tr>
<td>Subject (Experience)</td>
<td>47</td>
<td>.455</td>
<td></td>
</tr>
<tr>
<td>Truth</td>
<td>1</td>
<td>.904</td>
<td>15.23**</td>
</tr>
<tr>
<td>Experience*Truth</td>
<td>1</td>
<td>.002</td>
<td>.03</td>
</tr>
<tr>
<td>Residual</td>
<td>439</td>
<td>.059</td>
<td></td>
</tr>
</tbody>
</table>

The means (in seconds) for True and False were .945 and 1.152, respectively.

**p<.01
TABLE 26  
Statistical Analysis of True and False Data for GT

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>MS</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experience</td>
<td>1</td>
<td>1.135</td>
<td>2.48</td>
</tr>
<tr>
<td>Subject (Experience)</td>
<td>47</td>
<td>.459</td>
<td></td>
</tr>
<tr>
<td>Truth†</td>
<td>1</td>
<td>1.161</td>
<td>20.09**</td>
</tr>
<tr>
<td>Experience*Truth</td>
<td>1</td>
<td>.007</td>
<td>.09</td>
</tr>
<tr>
<td>Residual</td>
<td>439</td>
<td>.078</td>
<td></td>
</tr>
</tbody>
</table>

†The means (in seconds) for True and False were .740 and .959, respectively.
**p<.01
TABLE 27

Statistical Analysis of True and False Data for LT

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>MS</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experience</td>
<td>1</td>
<td>1.089</td>
<td>3.29</td>
</tr>
<tr>
<td>Subject (Experience)</td>
<td>47</td>
<td>.331</td>
<td></td>
</tr>
<tr>
<td>Truth¹</td>
<td>1</td>
<td>.593</td>
<td>7.09**</td>
</tr>
<tr>
<td>Experience*Truth</td>
<td>1</td>
<td>.006</td>
<td>.07</td>
</tr>
<tr>
<td>Residual</td>
<td>439</td>
<td>.084</td>
<td></td>
</tr>
</tbody>
</table>

¹The means (in seconds) for True and False were .943 and 1.106, respectively.

**p<.01
TABLE 28
Statistical Analysis of True and False Data for NOT

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>MS</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experience</td>
<td>1</td>
<td>.086</td>
<td>.15</td>
</tr>
<tr>
<td>Subject (Experience)</td>
<td>47</td>
<td>.558</td>
<td></td>
</tr>
<tr>
<td>Truth(^1)</td>
<td>1</td>
<td>.063</td>
<td>.52</td>
</tr>
<tr>
<td>Experience*Truth</td>
<td>1</td>
<td>.175</td>
<td>1.46</td>
</tr>
<tr>
<td>Residual</td>
<td>439</td>
<td>.120</td>
<td></td>
</tr>
</tbody>
</table>

\(^1\)The means (in seconds) for True and False were 1.053 and 1.109, respectively.
TABLE 29

Statistical Analysis of True and False Data for OR

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>MS</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experience</td>
<td>1</td>
<td>.500</td>
<td>1.01</td>
</tr>
<tr>
<td>Subject (Experience)</td>
<td>47</td>
<td>.494</td>
<td></td>
</tr>
<tr>
<td>Truth^1</td>
<td>1</td>
<td>2.242</td>
<td>38.23**</td>
</tr>
<tr>
<td>Experience * Truth</td>
<td>1</td>
<td>.043</td>
<td>.70</td>
</tr>
<tr>
<td>Residual</td>
<td>439</td>
<td>.061</td>
<td></td>
</tr>
</tbody>
</table>

^1The means (in seconds) for True and False were 1.219 and 1.676, respectively.
**p<.01
APPENDIX B. HUMAN SUBJECTS FORM

The form approved by the human subjects committee is shown on the next page.
1. Title of project (please type): Factors affecting Computer Program Comprehension

2. I agree to provide a proper surveillance of this project to ensure that the rights and welfare of the human subjects are properly protected. Additions to or changes in procedures affecting the subjects after the project has been approved will be submitted to the committee for review.

John P. Boysen
Typed Name of Principal Investigator
Date
Signature of Principal Investigator

233 Computer Science
Campus Address
294-2219
Campus Telephone

3. Signatures of others (if any) Date Relationship to Principal Investigator

4. ATTACH an additional page(s) (A) describing your proposed research and (B) the subjects to be used, (C) indicating any risks or discomforts to the subjects, and (D) covering any topics checked below. CHECK all boxes applicable.

☐ Medical clearance necessary before subjects can participate
☐ Samples (blood, tissue, etc.) from subjects
☐ Administration of substances (foods, drugs, etc.) to subjects
☐ Physical exercise or conditioning for subjects
☐ Deception of subjects
☐ Subjects under 14 years of age and/or Subjects 14-17 years of age
☐ Subjects in Institutions
☐ Research must be approved by another institution or agency

5. ATTACH an example of the material to be used to obtain informed consent and CHECK which type will be used.

☐ Signed Informed consent will be obtained.
☑ Modified Informed consent will be obtained.

6. Anticipated date on which subjects will be first contacted: Month Day Year

7. If Applicable: Anticipated date on which audio or visual tapes will be erased and/or identifiers will be removed from completed survey instruments: Month Day Year

8. Signature of Head or Chairperson Date Department or Administrative Unit

9. Decision of the University Committee on the Use of Human Subjects in Research:

☑ Project Approved
☐ Project not approved
☐ No action required

George G. Karas
Name of Committee Chairperson
Date
Signature of Committee Chairperson


