Management of Computing

Gordon B. Davis Editor

Structured Tools and Conditional Logic: An Empirical Investigation

IRIS VESSEY and RON WEBER

ABSTRACT: Prior research has identified two psychological processes that appear to be used by programmers when they perform design and coding tasks: (a) taxonomizing—identifying the conditions that evoke particular actions; and (b) sequencing—converting the taxa into a linear sequence of program code. Three structured tools—structured English, decision tables, and decision trees—were investigated in a laboratory experiment to determine how they facilitated these two processes. When taxonomizing had to be undertaken, structured English outperformed decision tables, and decision trees outperformed structured English. When sequencing had to be undertaken, decision trees and structured English outperformed decision tables, but decision trees and structured English evoked the same level of performance.

1. INTRODUCTION

The proponents of structured analysis and structured design advocate three tools for representing conditional logic: structured English, decision tables, and decision trees. These tools are used in the final step of structured analysis to describe policy for a transform (bubble) in a data-flow diagram—the major means of partitioning, analyzing, and documenting the problem domain. In addition, they may be converted into pseudocode for modules outlined on the structure chart during structured design (see, e.g., [11]).

This research was funded by the Australian Research Grants Scheme.

Knowledge about the relative strengths and limitations of these three tools is limited, however. De Marco [4, p. 16] claims they are "something better than narrative text"—the existing tool—but he does not advocate one in particular. Gane and Sarson [8] argue that decision tables are more useful than decision trees when the number of actions is large, many combinations of conditions exist, and there is a risk of ambiguities and omissions; but at least for simple problems the pictorial vividness of the decision tree makes it more understandable. Page-Jones [11] advocates using structured English if complex branching processes are not involved.

This article describes an experiment that seeks to provide insight into the relative strengths and limitations of these three tools. Our motivation for the research is the belief that there are significant differences among the tools that facilitate or inhibit the expression of conditional logic, depending on the analysis, design, or programming task to be undertaken. Practically, these differences are important as conditional statements are frequently used in program code (e.g., [5]), even when fourth-generation languages are employed [14]. Theoretically, these differences are also important as they provide insights into several psychological constructs that seem to impact the performance of analysts, designers, and programmers.

The article proceeds as follows. Section 2 reviews some prior research that provides the background and motivation for the current research. Section 3 presents

the theory underlying the current research and the propositions investigated. Section 4 describes the empirical research methodology used to test the propositions. Sections 5 and 6 present the data analysis and discuss the results obtained. Section 7 presents our conclusions.

2. BACKGROUND

An important outcome of recent work on the psychology of programming has been the recognition that we have a poor understanding of how various programming practices—indentation, commenting, naming, etc.—facilitate or inhibit the programming process. After a fairly extensive series of studies, many results obtained are contradictory and counterintuitive (see, e.g., [12]). Part of the problem may be the research methodologies used (see, e.g., [1]). However, the more important problem seems to be the poor theoretical bases that have driven the research (see, e.g., [18]).

The theory seems deficient in two ways. First, we have little knowledge of the psychological constructs that programmers bring to bear when they undertake a programming task. Pennington [12] illustrates this problem in the following way. She reflects upon the equivocal results obtained for commenting in programs; several empirical studies show comments to be of little use. Clearly, she argues, commenting *must* be useful under some conditions, namely, those times when the semantics of the program are unclear. The problem is that we do not know *when* the semantics of the program are not obvious to the programmer from an examination of the executable source code. If the semantics can be deduced easily from the code, then comments may simply confuse the programmer. But if the semantics are not obvious, commenting should assist the programmer.

Second, as a related issue, we have little knowledge of what is "natural" for programmers. Indeed, there is some evidence to suggest that our current notions are misguided. For example, a nonprocedural language is supposed to be more natural than a procedural language. However, Welty and Stemple [20] found that programmers using a procedural language outperformed programmers using a nonprocedural language. As task complexity increases, programmers may need to think in terms of a concrete, procedural model if they are to solve the problem (see, also, [6]). Similarly, Miller [10] studied procedural instructions written by nonprogrammers. He found that people generally avoid conditional statements and prefer qualificational statements. Thus, programming constructs like IF-THEN-ELSE may not be natural (see, also, [17]).

As a result of these problematical findings, several recent studies support the notion that programming language constructs that provide a close cognitive fit with a person's preferred problem solving strategy are used more effectively (see, e.g., [16]). If this notion is to be investigated further, however, the cognitive processes used in programming must be better identified, and propositions about the ways different programming practices facilitate or inhibit these processes must be developed.

3. THEORY

In an important paper on the psychology of programming, Sime, Green, and Guest [15] hypothesized that two tasks must be performed to produce a program: (a) taxonomizing—identifying what conditions lead to a particular action; and (b) sequencing—converting taxa into the linear sequence of program code. These two tasks evoke different psychological processes. The first involves classifying and sorting elements according to their attributes; the second involves specifying the taxonomic criteria and associated actions within the syntactic and semantic constraints of the programming language used. Figure 1b (p. 49) shows (in decision-table form) the result of a taxonomizing operation for the conditional logic described in the narrative in Figure 1a. Figure 1c shows the result of a sequencing operation where the language used prohibits an unconditional transfer of control (GOTO).

In light of our arguments in Section 2, how well do structured English, decision tables, and decision trees support the taxonomizing and sequencing tasks? Consider a situation in which an analyst or designer is interviewing a user about some policy where conditional logic applies. The primary objective is to establish completely and unambiguously the conditions that lead to particular actions; in other words, the initial task is a taxonomizing task. Assume the analyst attempts to translate the user's communications straight into structured English—specifically, structured English where, in line with structured programming precepts, the GOTO is prohibited. The task is difficult for three reasons. First, the taxonomizing task is two-dimensional; the analyst needs to be able to "see" the relevant conditions as one dimension and their applicability or nonapplicability to an action as another dimension. Structured English is unidimensional; it shows a linear sequence of conditions and actions that perceptually are not easy to differentiate. Second, if the analyst wishes to write structured English as a series of nested tests, there is only *one* sequence in which the conditions can be tested when the GOTO is prohibited. Given the taxa in Figure 1b, Figure 1c shows the *only* way in which the series of nested tests can be constructed. Alternatively, a series of repetitive conjunctions (e.g., if A and B and not C) must be written, but Green, Sime, and Fitter [9] report that difficulties arise with this solution. Third, structured English provides no formal way to test for completeness, consistency, and nonredundancy in the code. This may be a serious failing as the number of conditional tests to be undertaken increases.

To illustrate these difficulties, we urge the reader to attempt to write structured English to represent the conditional logic expressed in the narrative shown in Figure 2a (p. 50). The narrative might represent the text of a discussion between an analyst and a user. If nested

Vegetables that are both leafy and crispy should be fried, while those that are crispy but not leafy should be boiled. Prior to cooking, all vegetables that are not crispy should be chopped. Then, those that are green and hard should be boiled, while those that are hard but not green are steamed; those that are not hard are grilled.

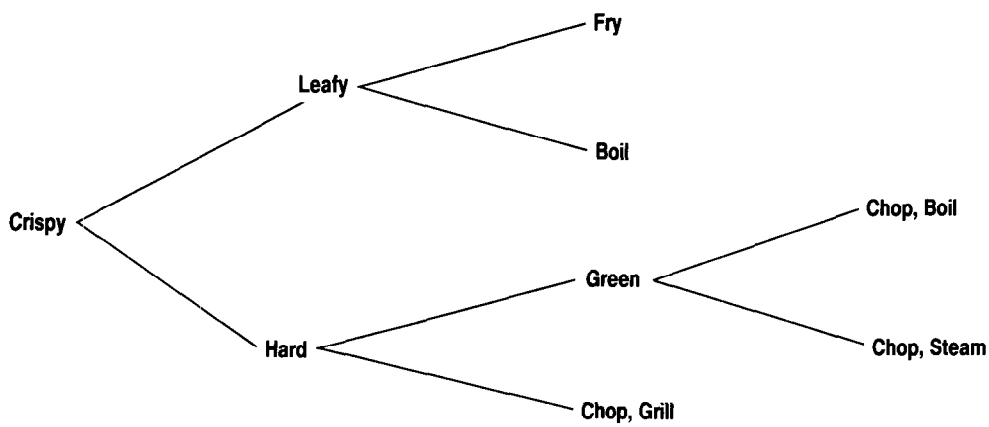
(a) Narrative description of conditional logic

Crispy	Y	Y	N	N	N
Hard	—	—	Y	Y	N
Leafy	Y	N	—	—	—
Green	—	—	Y	N	—
Fry	X				
Chop			X	X	X
Boil		X	X		
Steam				X	
Grill					X

(b) Decision-table solution

```
If crispy
  If leafy
    fry
  otherwise
    boil
otherwise
  chop
  If hard
    If green
      boil
    otherwise
      steam
  otherwise
      grill
```

(c) Structured-English solution



(d) Decision-tree solution

FIGURE 1. Simple Problem Used in Experiments

conditionals are to be written, Figure 2c shows the only solution. We believe the reader will find, as we did, that the solution is difficult to write directly from the narrative text.

Both decision tables and decision trees overcome the problems posed by the unidimensional nature of structured English when the taxonomizing task is to be undertaken. Indeed, a primary strength of these tools is that they separate the conditions from the actions and show via rules or branches the particular combination of truth-values that leads to a particular action. In essence, we are arguing that decision tables and decision trees provide a better cognitive fit than structured English when the taxonomizing task must be undertaken. Which of these two tools is better is more difficult to determine. On the one hand, we suspect that the pictorial vividness of decision trees makes them superior to decision tables. On the other hand, decision tables provide formal procedures for ordering the sequence of conditions to be tested and for checking completeness, consistency, and redundancy. This is not the case with decision trees. Again, attempting to determine the sequence of tests that avoids redundant branches in a decision tree is a difficult task. To appreciate these

Crispy, leafy vegetables that are juicy but not tall, are fried if they are red; otherwise they are steamed. Crispy vegetables that are juicy but neither tall nor leafy are grilled. Noncrispy vegetables that are not tall but are juicy are prepared in two steps: they are first peeled, and then if they are hard they are boiled, otherwise they are chopped. The recommended method of cooking all vegetables that are not juicy is roasting. Juicy vetegables that are tall are chopped.

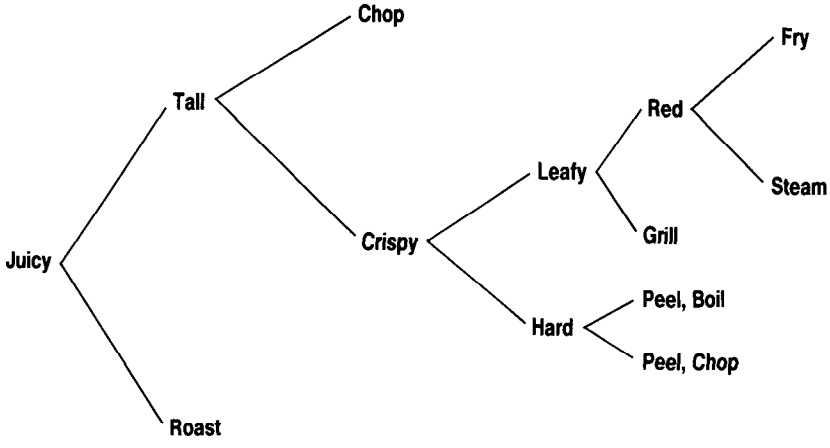
(a) Narrative description of conditional logic

Juicy	Y	Y	Y	Y	Y	Y	N
Tall	Y	N	N	N	N	N	—
Crispy	—	Y	Y	Y	N	N	—
Leafy	—	Y	Y	N	—	—	—
Red	—	Y	N	—	—	—	—
Hard	—	—	—	—	Y	N	—
Fry		X					
Steam			X				
Grill				X			
Peel					X	X	
Boil					X		
Chop	X					X	
Roast							X

(b) Decision-table solution

```
If juicy
  If tall
    chop
  otherwise
    If crispy
      If leafy
        If red
          fry
        otherwise
          steam
      otherwise
        grill
      otherwise
        peel
        If hard
          boil
        otherwise
          chop
otherwise
  roast
```

(c) Structured-English solution



(d) Decision-tree solution

FIGURE 2. Complex Problem Used In Experiments

difficulties, we urge the reader to first attempt the decision-tree solution for the narrative in Figure 2a and then to attempt the decision-table solution (see Figures 2d and 2b).

The sequencing task is another story. When the programmer converts the conditional logic into program code, we argue that the structured English representation facilitates this task most. The mapping from struc-

tured English to program code is almost one to one. The mapping from decision tables or decision trees, however, involves a transformation from a two-dimensional representation of the logic to a unidimensional representation. Notwithstanding that there are some straightforward algorithms for performing this transformation (see, e.g., [19]), we argue that structured English provides a closer cognitive fit with the psychological constructs invoked to write program code.

To recap, then, the transformation of conditional logic into program source code involves two psychological processes. First, the analyst, designer, or programmer must determine the set of conditional truth-values that leads to a particular set of actions—the taxonomizing process. Second, the taxa must be converted into the linear representation of code—the sequencing process. We argue that the relative strengths and limitations of structured English, decision tables, and decision trees are a function of how well they facilitate or inhibit the cognitive processes invoked in each of these two transformations.

Accordingly, we advance the following two propositions:

Proposition 1: Decision tables and decision trees facilitate the taxonomizing process better than structured English.

Proposition 2: Structured English facilitates the sequencing process better than decision tables and decision trees.

4. METHOD

To test these propositions, a laboratory experiment was conducted in which the performance of participants using the three types of structured tools was measured across different programming tasks.

4.1 Participants

One hundred twenty-four volunteer information systems and computer science students in three tertiary institutions who had been trained in COBOL and structured analysis, design, and programming participated in the experiment. Each was paid $30 for participation, providing they completed all parts of the experiment.

4.2 Design

Each participant undertook three experiments. First, the participants were given a narrative description of some conditional logic and asked to represent the narrative using one of the three types of structured tools. Second, they were given conditional logic already described via one of the tools and asked to convert it into COBOL code. Third, they were given a narrative description of some conditional logic and asked to convert it into COBOL code after representing the logic using one of the three types of structured tools. Thus, the first experiment tested Proposition 1, and the second experiment tested Proposition 2. The third experiment allowed the effects of each tool to be investigated across the "full" programming task—that is, design *and* coding—and it also took into account the possibility that participants may switch back and forth between the taxonomizing and sequencing processes. All experiments used adaptations of cooking problems developed by Sime et al. [15] in an attempt to minimize the effects of application domain knowledge.

Within each experiment, a mixed design was used with two between-subjects factors and one within-subjects factor. The two between-subjects factors were tool and problem complexity. Tool was measured at three levels: structured English, decision tables, and decision trees. Problem complexity was measured at two levels: simple and complex. A simple problem used conditional logic that had four conditions, five actions, and four levels of nesting when it was converted into COBOL code. A complex problem used conditional logic that had six conditions, seven actions, and five levels of nesting when it was converted into COBOL code. Two levels of problem complexity were used because, as discussed earlier, some writers argue that the relative strengths and limitations of a tool depend on the level of complexity of the logic to be described. Figures 1a and 2a show, respectively, the narrative used for a simple problem and a complex problem.

The within-subjects factor was problem (trial). Each subject performed the same experimental task twice. For example, a participant assigned to the simple problem-structured English treatment converted narrative to structured English during two consecutive trials. Thus, two different problems with the same level of complexity had to be devised for each task. Problems were judged to have the same level of complexity if they had the same number of conditions, actions, and levels of nesting. The order of problem presentation was randomized across participants in the experiment.

4.3 Measures of Performance

Three measures of performance were used. The primary measure was time taken to perform the experimental task. The secondary measure was the number of syntactic errors made. Participants had to conform exactly with the syntactic requirements of the tool they used and the COBOL language. For example, if a participant misspelled a word, used a wrong level of indentation, or abbreviated a word, this was counted as a syntactic error. The third measure used was number of semantic errors made. Participants had to associate the correct actions with the correct conditions in the tool they used to express the logic or in the COBOL code they wrote.

4.4 Procedure

Prior to the experiment proper, a pilot test was conducted to identify any deficiencies in the experimental materials and to obtain practice at administering the experiment. Ten participants undertook the pilot test; each treatment was administered at least once, and ad-

ditional administrations occurred when the experimental materials had to be modified or when more practice was needed at running a treatment.

The experiment proper comprised a training session and administration of two sets of the three experimental tasks. Participants were first allocated randomly to treatments. Next, a training session was organized at a convenient time for the subjects. Where possible, subjects who had been allocated to the same treatment were trained together. Training sessions normally involved 3–5 participants.

A training session comprised four phases. First, the general characteristics of the tool were reviewed. Second, participants undertook four practice tasks of increasing difficulty for the narrative-to-tool experiment. Third, participants undertook three practice tasks of increasing difficulty for the tool-to-code experiment. Finally, participants undertook three practice tasks of increasing difficulty for the narrative-to-code experiment. After each administration of a task, participants could ask questions of the experimenter. They were also given sufficient time to assimilate the nature of the task and the experiment. The number of practice tasks used was intended to be sufficient for learning to have ceased and was determined on the basis of results obtained during pilot testing. On the average, training sessions took 3–5 hours. Interestingly, training sessions for structured English and decision trees took about the same time while training sessions for decision tables took longer.

For the narrative-to-tool task, participants were first provided with a sheet containing a narrative description of the conditional logic. Decision-table participants were shown the usual procedures for converting the narrative into a limited-entry decision table and for checking the completeness, consistency, and redundancy attributes of the decision table. Structured-English participants were provided with the vocabulary and syntax they were to use. Decision-tree participants were shown the specific type of decision tree they were to construct to represent the conditional logic. In addition, both the structured-English and decision-tree participants were shown an heuristic for determining the sequence of tests to be performed. Recall, one of the difficult aspects of the coding task for conditional logic when the GOTO is prohibited is determining the single, correct sequence of tests. With decision tables, this sequence is determined automatically by virtue of the way the table is constructed. With structured English and decision trees, however, this is not the case. Nevertheless, an heuristic that can be used to determine the order of the tests is to sort the tests in descending frequency and follow the positive branch of a test to its conclusion before following the negative branch.

For the tool-to-code task, participants were first provided with a sheet containing conditional logic expressed in the tool to which they had been assigned and a sheet containing the data definition for the COBOL program they were to write. Next they were shown how to convert the conditional logic into COBOL code. In the case of structured English, this conversion was straightforward. In the case of decision trees, they were shown how to follow a positive branch and then a negative branch and how to determine the appropriate level of indenting. In the case of decision tables, they were shown the algorithm described by Vessey and Weber [19] for determining the path through the table and determining the appropriate level of indenting. For the narrative-to-code task, participants were provided with a narrative description of the conditional logic they were to use and the data division of the COBOL program they were to write. They were then shown how to progress from the narrative to the code after first expressing the conditional logic in the tool to which they had been assigned.

In all cases, participants provided their answers on preprinted sheets. For example, decision-table participants received a sheet with the lines for the condition stub, action stub, and rules already drawn, and structured-English participants received a sheet with several vertical lines drawn on the left-hand side to act as the margins for the various levels of indenting they chose.

When completing the answer sheets, participants were told they were to undertake the task as quickly as possible. Speed was their goal; however, in spite of speed being the primary objective, they had to provide answers that were completely accurate. In other words, no syntactic or semantic errors were to be present in their answers. In addition, they were told not to recheck their answers—they should strive to be accurate on the first iteration of their answer—and they could write or print their answers, whichever they preferred. These latter instructions attempted to force participants to undertake the experiment in a consistent way. Without an admonishment to be completely accurate, different participants might have traded off different levels of accuracy and speed. Similarly, task times might have varied considerably if participants employed varying strategies for checking their answers, and accordingly the effects of the tool on task performance might be difficult to determine. In the training session, participants were given some practice at undertaking the experimental tasks before they were required to comply with the speed instruction and have their performance timed.

Some 1–2 weeks after the training session, participants returned to undertake the experimental tasks. Each participant was run *singly* through the experiments. Prior to commencing the experiments, they received an instruction sheet to remind them of the nature of the tasks they were to undertake and the protocols they were to follow. The experiments were then conducted in a quiet room, free from noise and distractions with only the participant and the experimenter present. The experimenter simply issued and collected the experimental materials and unobtrusively recorded time taken for each task on a stopwatch. On the aver-

TABLE I. Means[a] and Standard Deviations[b] For Number of Syntactic Errors, Number of Semantic Errors, and Time Taken For Each Experiment

	N	Narrative-to-Tool (N-T) Experiment						Tool-to-Code (T-C) Experiment						Narrative-to-Code (N-C) Experiment					
		#Syn1	#Syn2	#Sem1	#Sem2	Time1	Time2	#Syn1	#Syn2	#Sem1	#Sem2	Time1	Time2	#Syn1	#Syn2	#Sem1	#Sem2	Time1	Time2
Simple English	22	.55[a]	.59	.23	.68	227	230	.45	.23	.55	.23	220	172	.32	.27	.95	1.81	391	436
		(1.22)[b]	(1.22)	(.75)	(1.81)	(129)	(161)	(1.06)	(.75)	(1.74)	(.69)	(67)	(45)	(.78)	(.70)	(1.81)	(1.84)	(146)	(170)
Table	22	.14	.18	.23	.00	314	275	.64	.64	.41	.36	275	206	.50	.73	.82	.50	593	530
		(.35)	(.50)	(.53)	(.00)	(86)	(67)	(.95)	(1.92)	(.67)	(.79)	(80)	(49)	(1.14)	(2.14)	(1.33)	(1.06)	(164)	(121)
Tree	20	.05	.05	.20	.05	153	139	.50	.30	.05	.10	202	147	.35	.50	.32	.35	336	336
		(.22)	(.22)	(.41)	(.22)	(47)	(48)	(1.05)	(.92)	(.22)	(.31)	(56)	(26)	(1.14)	(1.15)	(.99)	(.81)	(97)	(64)
Complex English	21	.14	.52	.24	.76	391	427	1.00	.62	.43	.52	285	320	.43	.48	1.43	1.52	806	889
		(.36)	(1.57)	(.54)	(2.19)	(164)	(172)	(1.84)	(1.47)	(.75)	(1.12)	(68)	(119)	(.75)	(1.36)	(1.03)	(2.91)	(305)	(295)
Table	20	1.05	1.25	.35	.40	642	655	1.50	1.80	.20	.25	402	456	1.60	1.90	1.05	1.70	1190	1423
		(3.79)	(4.33)	(1.18)	(.75)	(334)	(201)	(3.10)	(3.35)	(.52)	(.64)	(173)	(214)	(3.08)	(2.99)	(2.09)	(2.74)	(319)	(416)
Tree	19	.00	.00	.05	.00	302	278	.63	.53	.21	.11	282	293	.95	.74	.84	.37	663	784
		(.00)	(.00)	(.23)	(.00)	(127)	(84)	(1.86)	(1.87)	(.42)	(.32)	(75)	(74)	(2.39)	(2.00)	(1.34)	(.96)	(185)	(221)

Note: 1. #Syn1 = Number of syntactic errors, first problem
2. #Syn2 = Number of syntactic errors, second problem
3. #Sem1 = Number of semantic errors, first problem
4. #Sem2 = Number of semantic errors, second problem
5. Time1 = Time taken, first problem
6. Time2 = Time taken, second problem

age, the experimental session was completed in one hour.

Since the experiments assessed the performance of participants, confoundings could have arisen if experimenter expectancies had been conveyed to the participants [3]. Consequently, all training sessions and experiments were conducted by research assistants who were not informed of the propositions that were being tested. To ensure consistency of training, all training sessions were conducted by a single research assistant, but the administrations of the experiments were conducted by four different research assistants.

5. RESULTS

The data analysis proceeded in three steps. The first step involved fitting a repeated measures multivariate analysis of variance (MANOVA) model to the data since Pearson product moment correlation coefficients indicated that the dependent variables were moderately correlated. In all three experiments, the two between-subjects factors (complexity and tool) and their interaction (complexity × tool) were significant at the .01 level. In addition, for the tool-to-code and narrative-to-code experiments, the interaction between the within-subjects factor, problem, and the between-subjects factor, complexity, was significant at the .01 level. No other main or interaction effects were significant at the .01 level.

The second step in the analysis involved fitting a repeated measures analysis of variance model (ANOVA) with time as the dependent variable to those cases where the participant had made neither syntactic nor semantic errors across the two problems in an experiment. This step was undertaken for four reasons. First, a cursory examination of Table I suggests that in a practical sense time is the most important variable in the MANOVA results. The mean number of syntactic and semantic errors was low (see, also, Table II). Second, because of the high proportion of participants who made neither syntactic errors nor semantic errors when undertaking an experimental problem, the distribution of these dependent variables was acutely right skewed. Consequently, an important assumption of MANOVA—homogeneity of the variance-covariance matrices—was violated. Box's *M* statistic remained significant at the .01 level across various transformations of the data. Thus, statistically testing contrasts under MANOVA was a problematical procedure. Third, using repeated measures ANOVAs with the number of syntax errors and number of semantic errors as the dependent variables, neither main effects nor interaction effects were significant at the .05 family level of significance. Fourth, recall that participants were told they were to strive for maximum speed but to ensure they first achieved complete accuracy. The analysis of the *reduced* data set—the participants who made no errors across both problems in an experiment—provided the results for participants who had complied with this instruction.

TABLE II. Syntactic and Semantic Error Rate Statistics

Proportion of Participants Undertaking an Experimental Problem Who Made Neither Syntactic Nor Semantic Errors

	Experiment		
	N – T	T – C	N – C
Problem 1			
Zero syntactic errors made	.87	.69	.74
Zero semantic errors made	.86	.79	.57
Problem 2			
Zero syntactic errors made	.86	.81	.74
Zero semantic errors made	.86	.84	.69

Number of Participants in Each Experiment Who Made Neither Syntactic Nor Semantic Errors Across Both Problems in the Experiment

	Experiment		
	N – T	T – C	N – C
Simple			
English	12	16	10
Table	14	11	11
Tree	14	14	12
Complex			
English	10	8	1
Table	13	10	5
Tree	8	15	8

With only one exception, noted below, the results of the second step in the analysis are the same as those obtained for the third step in the analysis in which a repeated measures ANOVA was fitted to time for the *full* data set obtained from the 124 participants. In other words, the number of syntactic errors and the number of semantic errors seem to have little effect on the results for the time taken to complete an experiment. Given that larger data sets are more desirable for hypothesis testing and statistical estimation purposes, the results of the third analysis are presented below. In all analyses, the dependent variable, time, has been converted to minutes and a square root transformation applied in an attempt to correct for right skewness. This transformation was also used in the analyses undertaken in the second step discussed above. For the transformed dependent variable, Bartlett's test of sphericity was insignificant at the .01 level, indicating the transformation was at least somewhat successful.

For the narrative-to-tool experiment, only the two between-subjects main effects were significant at the .01 level (complexity, $F(1, 118) = 157.3$; tool, $F(2, 118) = 63.6$). Neither the within-subjects factor nor any of the interactions were significant. The results show that the complex problems took more time than the simple problems. For the tool factor, at the .01 family significance level, pairwise contrasts show that decision trees took less time than either structured English or decision tables, and structured English took less time than decision tables (see Table I).

For the tool-to-code experiment, the two between-subjects main effects were significant at the .01 level (complexity, $F(1, 118) = 112.3$; tool, $F(2, 118) = 21.3$), and a within-subjects factor interaction effect was significant (problem × complexity, $F(1, 118) = 30.2$). No other effects were significant. Again, the results show that the complex problems took more time than the simple problems. For the tool factor, at the .01 family significance level, pairwise contrasts show that decision trees and structured English took less time than decision tables, but there was no statistically significant difference between the time taken for decision trees and structured English. The significant interaction arose because the second complex problem on the average took more time than the first complex problem, and the first simple problem on the average took more time than the second simple problem. Recall that the two problems were randomized across trials, so the difference must reflect different levels of complexity between the problems.

For the narrative-to-code experiment, the two between-subjects main effects were significant at the .01 level (complexity, $F(1, 118) = 255.2$; tool, $F(2, 118) = 49.0$). The within-subjects factor and an interaction effect were also significant at the .01 level (problem, $F(1, 18) = 12.53$; problem × complexity, $F(1, 118) = 12.97$). Note, this is the one area of difference between the reduced data set and the full data set; in the reduced data set only the within-subjects main effect was significant. Once again, the complex problems took more time than the simple problems. For the tool factor, at the .01 family significance level, pairwise contrasts show that decision trees and structured English took less time than decision tables. The difference between decision trees and structured English was almost significant ($p = .011$), the former taking less time than the latter. The significant interaction arose because the second complex problem on average took more time than the first complex problem.

6. DISCUSSION

The narrative-to-tool results provide partial support for Proposition 1: as expected, decision trees outperformed structured English for the taxonomizing task; but, contrary to expectations, structured English outperformed decision tables. Table I also shows that even on the basis of the number of syntactic errors made and the number of semantic errors made, there is little support for decision tables as a superior tool to structured English for the taxonomizing task.

The tool-to-code results provide partial support for Proposition 2: as expected, structured English outperformed decision tables for the sequencing task; but, contrary to expectations, structured English and decision trees evoked the same level of performance. Again, there is little support for structured English as a superior tool to decision trees on the basis of the number of syntactic errors made and the number of semantic errors made.

The narrative-to-code results simply confirm the findings of the previous two experiments. Structured English and decision trees outperformed decision tables, and decision trees outperformed structured English, presumably on the basis of the former's superiority in the taxonomizing task. On the average, the narrative-to-tool experiment also took longer to complete than the tool-to-code experiment, so any superiority of a tool in the taxonomizing task presumably would be manifested in the narrative-to-code results.

At first glance the decision-tree results are surprising. Upon reflection, however, they make sense. Decision trees seem to combine the best features of both decision tables and structured English. On the one hand, the graphical tree structure enables taxon information to be represented poignantly. On the other hand participants found it easy to trace a branch to its leaf node to perform the sequencing task. Thus, unlike structured English and decision tables that facilitate only one task, decision trees facilitate both tasks. The superiority of decision trees also may simply confirm the arguments made by Fitter and Green [7]; namely, the desirability of graphically (perceptually) revealing the structure inherent in data or processes rather than using linear symbolic languages.

The results also provide support for the existence of the two important psychological processes used in programming—taxonomizing and sequencing—identified by Sime et al. [15]. As these researchers have pointed out, programming languages do not seem to have been designed with an understanding of the psychological processes that programmers must bring to bear on a task, nor have they been designed with an understanding of the representation that best facilitates the task to be performed. In this respect, current programming languages provide structured-English-like and decision-table-like representations of conditional logic, but few include decision trees as part of their syntax [2]. This can be understood in the context of older technology where graphical representations were difficult, but the current technology surely allows the development of compilers, interpreters, or preprocessors that include decision trees as part of their syntax. Perhaps our perceptions of programming languages as unidimensional, linear sequences of code are too limited and, in this respect, our results support some of the claims made by the proponents of visual programming [13].

Finally, the decision-table results are disappointing and somewhat unexpected. While running the experiments, it became clear that the procedures for constructing decision tables and checking their completeness, consistency, and nonredundancy are time consuming, and for many participants they remained awkward and unwieldy in spite of repeated practice in using them. Furthermore, one of the claimed benefits for decision tables—the formal rules for checking completeness, consistency, and redundancy—is not supported by the results obtained for the number of syntactic errors made and the number of semantic errors made. It might be argued that the tasks were not sufficiently complex for the benefits of decision tables to be demonstrated, but in designing the tasks we examined a large number of programs to gauge the average complexity of conditional logic. The tasks reflect this assessment. Perhaps the results reveal why, in our experience, computer professionals rarely employ decision tables, even when they are skilled in their design and use.

7. CONCLUSIONS

The primary purpose of our research was to examine the relative strengths and limitations of three structured tools—structured English, decision tables, and decision trees—for representing conditional logic. Our results support the notion that the usefulness of a tool must be considered in the context of the psychological task that programmers must perform. While all tools might represent the structure and processes inherent in a task to some extent, different tools manifest different aspects to a varying degree. Designers of information systems tools must understand what aspects of the task they wish to manifest *before* they design their tools.

Finally, the overall superiority of decision trees across the three experiments suggests that perhaps we should attempt to break a fixation with programming languages that use restricted syntactics—specifically, languages that require a linear sequency of text to be written. Programming languages that also provide graphical syntax might improve programmer performance, particularly where conditional logic must be written.

Acknowledgments. We are indebted to Barbara Elliot, Ming Kee Lui, Peter Tait, Byrne Haig, and Adrian Coulston for their research assistance. We are also indebted to colleagues in workshops at the University of Queensland and the New South Wales Institute of Technology for helpful comments on an earlier version of this article. Finally, we thank Jon Turner, Errol Iselin, and Craig McDonald for their detailed comments on the article.

REFERENCES

1. Brooks, R. Studying programmer behavior experimentally: The problems of proper methodology. *Commun. ACM 23*, 4 (1980), 207–213.
2. Brown, G.P., Carling, R.T., Herot, C.F., Kramlich, D.A., and Souza, P. Program visualization: Graphical support for software development. *Comput. 18*, 8 (1985), 27–35.
3. Christensen, L.B. *Experimental Methodology.* 2d ed., Allyn and Bacon, Boston, Mass., 1980.
4. De Marco, T. *Structured Analysis and System Specification.* Prentice-Hall, Englewood Cliffs, N.J., 1979.
5. Elshoff, J.L. A numerical profile of commercial PL/I programs. *Software-Pract. Exper. 6*, (1976), 505–525.
6. Fitter, M.J. Towards more "natural" interactive systems. *Int. J. Man-Mach. Stud. 11*, (1979), 339–350.
7. Fitter, M., and Green, T.R.G. When do diagrams make good computer languages? *Int. J. Man-Mach. Stud. 11*, (1979), 235–261.
8. Gane, C., and Sarson, T. *Structured Systems Analysis: Tools and Techniques.* Prentice-Hall, Englewood Cliffs, N.J., 1979.
9. Green, T.R.G., Sime, M.E., and Fitter, M.J. The problems the programmer faces. *Ergonomics 23*, 9 (1980), 893–907.
10. Miller, L.A. Natural language programming: Styles, strategies, and contrasts. *IBM Syst. J. 20*, 2 (1981), 184–215.
11. Page-Jones, M. *The Practical Guide to Structured Systems Design.* Yourdon Press, New York, 1980.
12. Pennington, N. Cognitive components of expertise in computer programming: A review of the literature. Cognitive Science Tech. Rep. Series #46, Univ. of Michigan, Ann Arbor, Mich., 1982.
13. Raeder, G. A survey of current graphical programming techniques. *Comput. 18*, 8 (1985), 11–25.
14. Read, N.S., and Harmon, D.L. Assuring MIS success. *Datamation 27*, 2 (1981), 109–120.
15. Sime, M.E., Green, T.R.G., and Guest, D.J. Psychological evaluation of two conditional structures used in computer languages. *Int. J. Man-Mach. Stud. 5*, (1973), 123–143.
16. Soloway, E., Bonar, J., and Ehrlich, K. Cognitive strategies and looping constructs: An empirical study. *Commun. ACM 26*, 11 (1983), 853–860.
17. Thomas, J.C., and Carroll, J.M. Human factors in communication. *IBM Syst. J. 20*, 2 (1981), 237–263.
18. Vessey, I., and Weber, R. Research on structured programming: An empiricist's evaluation. *IEEE Trans. Softw. Eng. SE-10*, 4 (1984), 397–407.
19. Vessey, I., and Weber, R. Conditional statements and program coding: An experimental evaluation. *Int. J. Man-Mach. Stud. 21*, (1984), 161–190.
20. Welty, C., and Stemple, D.W. Human factors comparison of a procedural and a nonprocedural query language. *ACM Trans. Database Syst. 6*, (1981), 626–649.

CR Categories and Subject Descriptors: D.2.1 [**Software Engineering**]: Requirements/Specifications—*tools*; D.2.2 [**Software Engineering**]: Tools and Techniques—*decision tables*

General Terms: Experimentation

Additional Key Words and Phrases: structured analysis tools, minispecs, structured English, decision trees, decision tables, task complexity, empirical evidence.

Received 8/85; revised 10/85; accepted 10/85

Authors' Present Address: Iris Vessey and Ron Weber, Department of Commerce, University of Queensland, St. Lucia, Queensland, Australia 4067.