A Comparison of the Organizational Strategies
of Multilingual Computer Programmers

by

Lynn T. Cunningham

Thesis submitted to the Faculty of the Virginia Polytechnic
Institute and State University in partial fulfillment of the
requirements for the degree of

Master of Arts

in

Curriculum and Instruction

APPROVED:

[Signatures]

May, 1988
Blacksburg, Virginia
LD
5655
V855
1988
C867
<.2
A Comparison of the Organizational Strategies of Multilingual Computer Programmers

by

Lynn T. Cunningham

John K. Burton, Chairperson

Curriculum and Instruction

(ABSTRACT)

The objective of this study was to determine whether computer programmers would organize reserved words by programming language or by conceptual category, when given an opportunity to use either strategy. Twenty-seven participants, stratified by programming experience level (novice, intermediate, and expert), were given sixteen reserved words on index cards. The words were taken from four programming languages, as well as six conceptual categories. Participants were given both a recognition and a recall task.

Organizing the words by conceptual category enabled the expert programmers to perform significantly better on the recall task than experts who organized by language. In addition, they made fewer recognition errors, and had more structured recall, in terms of recalling the words by the categories in which they were studied.

Expert computer programmers, similar to natural language multilinguals, can recall more (reserved) words when they are organized by conceptual categories rather than by (programming) language. It is hypothesized that this is because human memory is
organized in a fundamentally interdependent (across languages) manner in many domains other than natural language, such as computer programming.
# Table of Contents

Title Page ....................................................... i
Abstract ......................................................... ii
Table of Contents .............................................. iv
List of Tables .................................................... v
List of Figures .................................................... vi

Introduction ....................................................... 1
Multilingualism ................................................... 4
Research Question and Hypothesis ............................... 9
Method ............................................................ 9
  Subjects ....................................................... 10
  Procedures ................................................. 11
  Analysis ..................................................... 15
Results .......................................................... 16
Conclusion ....................................................... 22
Implications ..................................................... 24
List of Tables

Table 1  Reserved Words Selected for Recognition and Recall Tasks ................. .12

Table 2  Reserved Words Remaining, and used as Distractors .................. 13

Table 3  Means (and Standard Deviations) for Recall Task by Experience Level and Organizational Strategy Chosen ............. .16

Table 4  Mean Concept-Ordered Pairs on Recall Task by Experience Level and Organizational Strategy Chosen ................. .18

Table 5  Mean Language-Ordered Pairs on Recall Task by Experience Level and Organizational Strategy Chosen ................. .18

Table 6  Correlation between Total Items Recalled and Programming Experience Level, Organizational Strategy Chosen, Number of Programming Languages Known, and Number of Computer Science-Related Courses Taken .................. .20

Table 7  Mean Reaction Times in seconds for Recognition Task Language and Conceptual Pairs by Experience Level and Organizational Strategy Chosen .................. .21

Table 8  Mean Conceptual and Language Errors on Recognition Task by Experience Level and Organizational Strategy Chosen ................. .22
List of Figures

Figure 1  Types of Programmer Multilingualism . . . . . . . . . . . . . . 6
Figure 2  Analysis Procedure for Ordered Pairs on Recall Task . .19
A Comparison of the Organizational Strategies of Multilingual Computer Programmers

Introduction

How should future computer professionals be taught programming? This question has been the subject of recent research, including DuBoulay, O'Shea, and Monk (1981), Mayer (1979), Shneiderman (1977), and Weinberg (1982). Glass (1981) (in the context of accounting for programmer efficiency) states, "we need to know why some programmers are an order of magnitude better than others" and "we need to know what those exceptional programmers do differently from others" (pg. 10). These statements were made in light of the findings that capability to debug differs by factors of up to 28-1, and capability to code differs by up to 25-1.

One aspect of this question, which lies at the heart of one's philosophy of computer science education, is what programming languages should be taught (Masterson, 1984), and in what sequence? If a vocational orientation prevails, the aim is to expose the learner to the actual languages he or she will be using. Since the particular languages needed cannot always be known, the strategy then becomes one of exposing the student to as many languages as possible. If, on the other hand, one subscribes to a liberal arts philosophy, the purpose of programming education is to teach the programmer how to learn new programming languages.
In this case, it is not as important to assess the current and future use of popular and widespread programming languages, such as COBOL and FORTRAN. It becomes more important to choose a language which is "cognitively rich" and facilitates thinking about computer problem-solving (Masterson, 1984) over one which is simple or powerful or in widespread use. In fact, it is unlikely that one language will meet the needs of both the novice and the expert. Green (1983) identifies a trade-off concerning the number of features of a language. Fewer features are simpler for novices, because greater discriminability is possible, while more features are more useful to the professional programmer.

Based on his long-term teaching experience and experimentation, Weinberg (1982) recommends the teaching of two languages simultaneously: an assembly language and a high-level language. Supportive of this recommendation, Weyer and Cannara (1975, as cited in Du Boulay, O'Shea, & Monk, 1981) studied the progress of high school students in learning computer programming while learning two programming languages in one of several ways. One group learned LOGO then SIMPER (a simulated decimal assembly language), another group learned SIMPER then LOGO, and the third group learned LOGO together with SIMPER. The researchers found that pupils who learned both languages simultaneously performed best on a test of programming concepts.
The concept of more and less cognitively rich languages raises a related question. How should individual programming languages, especially the first programming language, be taught? Brewer (1977) maintains, "a current shibboleth in computer science is that one must teach 'pure computer science', a set of concepts untainted by any particular language" (p. 193). Shneiderman (1977) uses the distinction between semantics and syntax to advocate a spiral approach to the teaching of both. He describes semantic knowledge as "general programming concepts that are independent of specific programming languages" and "syntactic knowledge (as) a second kind of information stored in long-term memory". Syntactic knowledge "is more precise, detailed, and arbitrary (hence more easily forgotten) than semantic knowledge, which is generalizable over many different syntactic representations" (p. 222). Shneiderman advocates teaching the underlying concepts of programming, and incorporating the syntax of a particular language only for illustrative purposes. For this study, consistent with Shneiderman (1977), a semantic category was defined as a programming concept which is generalizable over many different syntactic representations. For example, looping, conditionals, and subroutines were all considered to be semantic categories. Thus individual reserved words were considered to be instances of syntax, while a group of them from several programming languages with a similar function was considered to be an instance of a semantic concept or category.
Multilingualism

One aspect of the notion of programmer efficiency is how programmers organize their knowledge of multiple programming languages in such a way that they can use it in both an integrated and non-interfering manner. An important area of natural language bilingualism concerns the organization of several languages in the brain, and two theories have been proposed to explain the research findings (McCormack, 1977). The first is that there are "two functionally independent storage and retrieval systems, one for each language, that interact only through translation processes" (p. 57). This is the two-store, or independence, model. Alternate, others advocate interdependence, with all information existing in a single memory store.

Both independence and interdependence are of value in certain contexts of programming, because, for example, independence controls interference from other programming languages. Interference is the phenomenon of, for example, a programmer who is most familiar with Fortran forgetting the colon when coding an assignment statement in Pascal. On the other hand, interdependence is of value in abstracting a structure or schema for programming languages to aid in learning new languages. This is beneficial to the programmer when learning languages that are of the same family, e.g., algorithmic languages. However, an algorithmic programmer who attempts to bring this schema to the task of learning a recursive language such as Lisp is in for
trouble. In this case, negative transfer may occur when more
languages are known.

Recent natural language theory hypothesizes that there are
three types of multilingualism: coordinate, compound, and
subordinate (Paradis, 1977). These types will be defined and
explained in terms of the role they might play in programming
language knowledge.

Coordinate (independent) multilingualism assumes that "the
signs of each language, ... a unit of expression and one of
content, are kept separate" (Paradis, 1977, p. 95), that is,
different conceptual systems are maintained for the two languages
(Hakuta, 1986). This implies that the programmer holds his or her
knowledge of e.g., Basic almost totally separate from his or her
knowledge of e.g., Pascal. Compound (interdependent)
multilingualism, on the other hand, implies that "the signs combine
one single unit of content with one unit of expression in each
language" (Paradis, 1977, p. 95). In this case, the programmer
stores programming concepts, connected to specific instances of
these concepts in terms of the syntax of known programming
languages. A third type, subordinate multilingualism presumes that
"the meaning unit is that of the mother tongue, with its
corresponding unit of expression that, in turn, has an equivalent
unit of expression in the second language"(p. 95). In this case,
the programmer has a single strong programming language, such as
the first language learned, the one best known, or the most
cognitively rich language, and the syntax for other languages is derived in terms of this one. (See Figure 1.) These types are not totally distinct, because the language systems "may stand in a variety of relationships with respect to each other, ...with real individuals situated somewhere on a continuum, sharing in various degrees parts of their linguistic system with all three types" (Paradis, 1977, p. 96).

Figure 1 Types of programmer multilingualism

<table>
<thead>
<tr>
<th>Coordinate</th>
<th>Compound</th>
<th>Subordinate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>Pascal</td>
<td>Fortran</td>
</tr>
<tr>
<td>for-next</td>
<td>for-do</td>
<td>do</td>
</tr>
<tr>
<td>Pascal</td>
<td>Basic</td>
<td>Fortran</td>
</tr>
<tr>
<td>for-do</td>
<td>for-next</td>
<td>do</td>
</tr>
<tr>
<td>Fortran</td>
<td>Pascal</td>
<td></td>
</tr>
<tr>
<td>do</td>
<td>for-do</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>db</td>
</tr>
</tbody>
</table>

These types of multilingualism have been studied extensively (Paradis, 1977) in patients with aphasia (a speech disorder due to brain damage which impairs a person's ability to use language). Selective impairment of a multilingual's languages affords researchers an opportunity to explore the question of how information concerning several languages is stored. Interestingly, researchers have found that sometimes the language most familiar, sometimes the language first learned, and sometimes the language with the strongest emotional ties was the one least likely to be impaired, however, no consistent rule could be found. (Paradis, 1977).
Lambert, Havelka, and Crosby (1958, as cited in Lambert, 1972) concluded, on the independence-interdependence issue, that "the learning contexts in which the bilingual's two languages are acquired may well determine the functional independence of the languages" (p. 73). The more the learning contexts are 'separated', either in time, cultural distinctiveness, or in distinctiveness of the setting of habitual usage, the more likely it is that bilingual "co-ordinateness" will develop. Hakuta (1986) summarizes this matter with, "the real question is the identification of the conditions under which the two languages maintain separation and those under which they are apparently merged" (p. 94). This, indeed, is the most important issue to computer science education, especially for educators teaching a programming language to some students for whom it is their first language, as well as to students who already know several.

A protocol for this study can thus be found in the natural language bilingualism research, grounded in terms of a linguistic model. Kolers and Gonzalez (1980, as cited in Hakuta, 1986) found that when bilinguals, in preparation for a recall test, had the opportunity to cluster words by conceptual (semantic) category or by language (syntax), they tended to choose conceptual category. Kolers and Gonzalez concluded that category clustering, at least in the given situation, is probably more efficacious for recall than language clustering. Does this hold in the domain of programming language knowledge? Recent studies suggest that it may.
Adelson (1981), for example, found that when lines of program code were presented randomly to novices and experts in a free-recall task, novices tended to impose a syntactic organization on the code, while experts shifted to a semantic pattern of organization. Adelson also found that there was a greater uniformity of recall order among experts, presumably because of the common knowledge base upon which they draw. In a similar study, McKeithen, Reitman, Rueter, and Hirtle (1981) found that when subjects were presented with 21 reserved words from the Algol language on index cards, and were asked to recall them in a multtrial, free-recall task, novices tended to chunk the words using common language strategies, such as alphabetic order or by making up a story as a mnemonic device. The experts, however, tended to chunk the reserved words according to their function in the Algol language.

This study was adapted from the McKeithen et al. (1981) study. Reserved words from several programming languages were presented on index cards. Similar to the possible strategies of the Kolers and Gonzalez (1980, as cited in Hakuta, 1986) study, participants had an opportunity to organize and study the words by one of several strategies. The choices were: by the programming language from which the word was taken, by the conceptual category the reserved word represented, or by some other strategy. Chunking patterns were assessed through free recall and by observing the physical placement of the cards in in groups.
Research Question and Hypothesis

The question asked by this study was whether computer programmers of varying expertise organize programming language constructs predominantly by concept or primarily by language, as evidenced in both a pair recognition and a free-recall task, and how the strategy chosen relates to performance on these tasks.

It was hypothesized that a conceptual category organization would provide the more efficacious organization in terms of both words recalled and words recognized. While an organization by language would ostensibly be the more obvious, and thus potentially the more surface-level organization, it is presumed that a conceptual organization will represent greater cognitive economy. This is because, as participants recall conceptual categories one by one as cues to recall, they then activate the small, finite number (four) of languages that have been represented as candidates for recall. In contrast, a participant who chooses the more obvious language organization may then recall languages as category cues, and activate a much larger set of reserved words when choosing potential candidates for recall.

Method

The basic approach of the study was both a recognition and a recall task after a period of study of a set of reserved words from four programming languages. In the recognition task, participants were presented with pairs of reserved words and were asked to judge
whether both words were from the set of cards they studied. The recall task required that participants write down each of the sixteen words from the cards they had studied.

Subjects. Twenty-seven unpaid adult volunteers familiar with at least one programming language participated in a learning task in which organization or study strategy was learner-controlled. Random sampling was not used, partly because a high proportion of expert computer programmers was desired. Novices were sought more for comparison purposes than to be used in statistical analysis.

Prior to the study, information was collected from each participant about their programming background through a questionnaire which asked them to rank programming languages in order of familiarity (familiarity measured by whether they have written a non-trivial program in the language) from a group of commonly-known languages. Participants were asked the number of computer science-related courses they have taken, both undergraduate and graduate, as well as the number of years of full-time years of computer-related work experience they have had. Part-time experience was converted.

Based on the questionnaire information, participants were classified by programming experience level as either novice, intermediate, or expert computer programmers. These classifications were based on formal computer science-related coursework, partly because all participants were students. A classification based on the number of programming languages known
was considered and rejected because it was felt that the number of coursework hours taken was a more valid representation of number of hours of programming experience than number of languages known. Familiarity was defined as having written at least one non-trivial program in a particular language. Novices were defined as persons who had completed three or less programming-related or computer-science related courses. All novices knew three or less programming languages. Intermediate computer programmers had completed 4-11 programming-related courses. Experts had completed either a Bachelor’s or Master’s degree in computer science or are nearing completion of a Master’s degree, but did not have extensive computer-related professional experience. All experts knew six or more programming languages.

(It is proposed that there are two other categories of computer programming expertise that have not been represented in this study: pre-novice and master. A pre-novice would have no exposure to a programming language or programming concepts. A master would have both formal computer-related training and extensive computer-related professional experience.)

**Procedures.** Participants were given a deck of sixteen 3 x 5 in., randomly-ordered index cards on which were written reserved words from four languages: Pascal, BASIC, COBOL, and ForTran, along with the name of the language each is taken from in parentheses below the reserved word.
The words on the cards were chosen from six conceptual categories from four languages. Two words were thrown out from each language, for a total of sixteen words. (See Table 1). Strong parallels between reserved word meanings in the corresponding languages guided the choice of conceptual categories to be used, as well as the individual reserved words represented from each. No redundant reserved words were used. The words that were omitted, as well as reserved words from several other conceptual categories, then became candidates to be used in the recognition task as distractors, items which the participants should have identified as non-list items. (See Table 2).

### TABLE 1 Reserved Words Selected for Recall and Recognition Tasks

<table>
<thead>
<tr>
<th>Function</th>
<th>BASIC</th>
<th>Pascal</th>
<th>COBOL</th>
<th>ForTran</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSIGNMENT</td>
<td>let</td>
<td>:=</td>
<td>move</td>
<td>assign</td>
</tr>
<tr>
<td>DATA STRUCTURE</td>
<td></td>
<td>array</td>
<td>move</td>
<td>dimension</td>
</tr>
<tr>
<td>LOOPING</td>
<td></td>
<td>for-do</td>
<td>move</td>
<td>do</td>
</tr>
<tr>
<td>CONDITIONAL</td>
<td></td>
<td></td>
<td>if-else</td>
<td>if-go to</td>
</tr>
<tr>
<td>SUBROUTINE</td>
<td>gosub</td>
<td>procedure</td>
<td>perform</td>
<td></td>
</tr>
<tr>
<td>COMMENT</td>
<td>rem</td>
<td></td>
<td></td>
<td>*</td>
</tr>
</tbody>
</table>
TABLE 2 Reserved Words Remaining, and used as Distractors

<table>
<thead>
<tr>
<th>Function</th>
<th>BASIC</th>
<th>Language</th>
<th>FORTRAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSIGNMENT</td>
<td>dim</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA STRUCTURE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOOPS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONDITIONAL</td>
<td>if-then</td>
<td>if-then-else</td>
<td>go to</td>
</tr>
<tr>
<td>SUBROUTINE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMMENT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INPUT</td>
<td>read</td>
<td>read</td>
<td>read</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>print</td>
<td>writeln</td>
<td>write</td>
</tr>
<tr>
<td>END</td>
<td>end</td>
<td>end</td>
<td>stop</td>
</tr>
<tr>
<td>DATA TYPE</td>
<td>data</td>
<td>integer</td>
<td>numeric</td>
</tr>
<tr>
<td>ARITHMETIC</td>
<td>/</td>
<td>div</td>
<td>divide</td>
</tr>
</tbody>
</table>

The participants were asked to learn the words so that they could recall them without the aid of the cards. They were given three minutes in which to learn the words. It was suggested that shuffling the cards and/or spreading them out on the table before them might help them learn the words more easily. To categorize the participants' strategies, the experimenter observed and noted what each participant did with the cards in order to learn the words.

In task A, the participants were given five minutes to recall all sixteen words on paper. While this proceeded, the experimenter noted the order in which the cards had been arranged. In order to control for the effects of completing one task on the outcome of the other, half the participants were given task A first, and the other half were given task B first.
Task B was a measured-response, paired-recognition task, in which the participant was presented with a pair of reserved words. Each of the pair may or may not have been on the participant's list. There were six possible cases:

1. both were on a card in the same programming language, e.g., let, rem
2. both were on a card in the same conceptual category, e.g., for-next, do
3. one was on a card, the other was in the same language, but was not on a card, e.g., array, readln
4. one was on a card, the other was in the same conceptual category, but was not on a card, e.g., if-go to, if-then-else
5. neither was on a card, both were in the same programming language, e.g., go to, c
6. neither was on a card, both were in the same conceptual category, e.g., print, write

Four pairs were presented from each of the six cases above, for a total of twenty-four randomly presented pairs. The pairs were presented on a computer screen, one pair at a time. For each pair, the participant was asked to respond Yes or No to whether both words were in the list they studied by pressing one of two keys. Reaction times were measured by the computer, accurate to six significant digits, and data was stored to files, one file for each participant. Ten practice trials were conducted, in which only the word 'Yes' or 'No' appeared on the screen, to which the participant responded 'Yes' or 'No'. The purpose of the practice trials was to familiarize the participants with the motor pattern of the task and to obtain a baseline reaction time.
Reaction time is the time that elapses between the onset of the test stimulus and the subject's response (Klatsky, pg. 101), and gives an indication of how long it takes for internal events to occur. Collins and Quillian (1969, as cited in Anderson, 1985), in their work with mental propositional networks, proposed networks of concepts based on reaction times needed to make true-or-false judgments about statements. The recognition task included in this study was motivated by this work. For the purposes of this study, the underlying assumption of the recognition task is that a longer recognition response time indicates a larger search space, which suggests that information is less tightly organized, with fewer connections.

Following both tasks, a brief verbal questioning session was conducted in order for the participant to describe the organization strategy he or she employed.

Analysis. To analyze task A, a comparison was made between within-language recognition and within-conceptual-category recognition, as evidenced by response times for correct responses. Incorrect responses were omitted from means. Means of incorrect responses for each category were also computed.

To analyze task B, participants were categorized by whether a language or a conceptual organization pattern predominated, based on the order or the layout of the deck of cards. In addition, participants were categorized according to programming experience level and grouped into categories of novice, intermediate, and
expert, based on the questionnaire information. Means and standard deviations of the two-factored groups, characterized by organization strategy and experience level, were computed.

Results

Table 3 shows the recall task means and standard deviations for novice, intermediate, and expert participants who chose a conceptual, language, or other strategy. The total number of participants who chose each type of strategy was eight for Conceptual, thirteen for Language and six for Other (N = 27).

<table>
<thead>
<tr>
<th></th>
<th>Novice (n=6)</th>
<th>Intermediate(n=8)</th>
<th>Expert(n=13)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conceptual</td>
<td>14.0 (.82)</td>
<td>15.6 (.49)</td>
<td>15.00</td>
</tr>
<tr>
<td>Language</td>
<td>7.0 (2.00)</td>
<td>11.67 (1.70)</td>
<td>13.0 (1.58)</td>
</tr>
<tr>
<td>Other</td>
<td>9.0 (1.00)</td>
<td>11.5 (1.50)</td>
<td>9.83</td>
</tr>
<tr>
<td></td>
<td>8.33</td>
<td>12.50</td>
<td>14.00</td>
</tr>
</tbody>
</table>

Grand mean: 12.30

The most important finding illustrated by Table 3 is that experts who chose a conceptual organization performed significantly better on the recall task than experts who chose a language.
Among intermediate participants, those who used a conceptual organization performed significantly better on the recall task than those who used a language organization, $t(4) = 8.0000, p < .001$. Also, intermediate participants who chose a conceptual strategy were able to do better on the recall task than expert participants who chose a language strategy, although this difference was not significant, $t(9) = .944755$.

Tables 4 and 5 summarize the data concerning the order in which participants recalled the reserved words. If two words from the same language were together (sequential) on recall, this was considered a language-ordered pair. If two words from the same conceptual category were sequential on recall, this was considered a concept-ordered pair. (See Figure 2 for an illustration of how this analysis was performed.) The tables show that experts who chose a conceptual pattern of organization had a mean of 7.4 out of ten possible concept-ordered pairs on recall (74%), while experts who chose a language pattern of organization had a mean of 5.75 out of twelve possible language-ordered pairs on recall (47.9%). A chi-square test between these two means did not, however, yield any statistical significance.
### TABLE 4  Mean Semantic-Ordered Pairs on Recall Task by Experience Level and Organizational Strategy Chosen

<table>
<thead>
<tr>
<th></th>
<th>Novice (n=6)</th>
<th>Intermediate (n=8)</th>
<th>Expert (n=13)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conceptual</td>
<td>3.67</td>
<td>7.4</td>
<td></td>
</tr>
<tr>
<td>(36.7%)</td>
<td>(74%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Language</td>
<td>0</td>
<td>2.0</td>
<td>1.0</td>
</tr>
<tr>
<td>(0%)</td>
<td>(20%)</td>
<td>(10%)</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td>0.75</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>(7.5%)</td>
<td>(0%)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(percent of ten possible ordered pairs)

### TABLE 5  Mean Language-Ordered Pairs on Recall task by Experience Level and Organizational Strategy Chosen

<table>
<thead>
<tr>
<th></th>
<th>Novice (n=6)</th>
<th>Intermediate(n=8)</th>
<th>Expert(n=13)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conceptual</td>
<td>1.67</td>
<td>1.8</td>
<td></td>
</tr>
<tr>
<td>(13.9%)</td>
<td>(15%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Language</td>
<td>1.5</td>
<td>2.0</td>
<td>5.75</td>
</tr>
<tr>
<td>(12.5%)</td>
<td>(16.67%)</td>
<td>(47.9%)</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td>2.25</td>
<td>3.5</td>
<td></td>
</tr>
<tr>
<td>(18.75%)</td>
<td>(29.17%)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(percent of twelve possible ordered pairs)
Figure 2: Analysis procedure for ordered pairs on recall task

Example from a participant with a predominantly conceptual strategy:

```
for-next
for-do
if-else
if-goto
gosub
rem
```

Example from a participant with a predominantly language strategy:

```
for-do
array
procedure
gosub
let
do
assign
```

A qualitative analysis of the errors made by participants on recall found that by far the most common recall error were variations of an if construct (e.g., if then, if then else, if-do), appearing ten times in a total of 29 recall errors (37%). Errors that involved using part of an actual list item (e.g., go to, =:, letx) characterized all but five of the remaining errors. The remaining five were: de bug, program, deminlsh, quit, and press.

Table 6 shows the correlation between total items recalled and several other factors.
TABLE 6 Correlation between Total Items Recalled and Programming Experience Level, Organizational Strategy Chosen, Number of Programming Languages Known, and Number of Computer Science-Related Courses Taken (N = 27)

<table>
<thead>
<tr>
<th></th>
<th>Total recall and programming experience (1=Novice, 2=Intermediate, 3=Expert)</th>
<th>Total recall and organizational strategy chosen (1=Other, 2=Language, 3=Conceptual)</th>
<th>Total recall and number of programming languages known</th>
<th>Total recall and number of computer science-related courses taken</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>.7510 *</td>
<td>.6631 *</td>
<td>.8100 *</td>
<td>.5121 *</td>
</tr>
</tbody>
</table>

* p < .01

Table 7 shows the mean reaction times for the recognition task by experience level and organizational strategy chosen. ANOVA tests were used to compare reaction times of participants, categorized by experience level and organizational strategy chosen. This was done for each of the six cases of types of pairs given, as well as for combined cases of all pairs related by language (cases 1, 3, and 5), and all pairs related by concept (cases 2, 4, and 6). Only one statistically significant difference was found. Expertise was a factor in deciding on pairs in which one was on a card, and the other was in the same language but not on a card (distractor), (F(2,25) = 4.18, p < .05). This means that experts were significantly slower in deciding whether both reserved words from this type of pair were list items. This is presumably because they have previous familiarity with these non-list items.
from prior programming experience, and take longer to remember whether they remember the word from the list or from their previous experience. Overall, the research hypothesis concerning the recognition task must be rejected.

**TABLE 7** Mean Reaction Times in seconds for Recognition Task Language and Conceptual Pairs by Experience Level and Organizational Strategy Chosen

<table>
<thead>
<tr>
<th></th>
<th>Novice (n=6)</th>
<th>Intermediate(n=8)</th>
<th>Expert (n=13)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conceptual</td>
<td>S 1.79 L 1.57</td>
<td>S 1.80 L 1.65</td>
<td></td>
</tr>
<tr>
<td>Language</td>
<td>S 2.37 L 1.33</td>
<td>S 2.15 L 1.65</td>
<td>S 1.85 L 1.48</td>
</tr>
<tr>
<td>Other</td>
<td>S 2.35 L 1.99</td>
<td>S 2.02 L 1.44</td>
<td></td>
</tr>
</tbody>
</table>

Table 8 summarizes the data concerning the number of errors made on the recognition task. The difference in mean errors between the expert language group and the expert conceptual group is statistically significant, \( t(11) = 2.415, p < .05 \).
TABLE 8 Mean Semantic and Language Errors on Recognition Task by Experience Level and Organizational Strategy Chosen

<table>
<thead>
<tr>
<th></th>
<th>Novice(n=6)</th>
<th>Intermediate(n=8)</th>
<th>Expert(n=13)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conceptual</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S 1.66 L 1.0</td>
<td>S L 0.6 0.6</td>
<td></td>
</tr>
<tr>
<td>Language</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S 4.0 L 2.5</td>
<td>S L 2.0 1.0</td>
<td>S L 2.125 0.875</td>
</tr>
<tr>
<td>Other</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S 3.0 L 1.75</td>
<td>S L 3.0 0</td>
<td></td>
</tr>
</tbody>
</table>

Conclusion

It has been shown that expert participants who chose an organizational strategy by language could not recall all of the words and thus had lower recall percentages. The amount of study time allowed, three minutes, lends some measure of internal validity to the findings, since three minutes is conceivably enough time to form any of a variety of encoding or mnemonic strategies.

It is not clear why differences observed in the recall task were not substantiated in the recognition task. A recall task is generally acknowledged as being more difficult. The dual-process hypothesis of memory performance suggests that this is because recall involves two processes: generating potential candidates for recall and choosing among them (Klatsky, 1980). In contrast, a recognition task requires only the second process, the choice. Anderson (1985) suggests that a recognition task is easier than a
recall task because it offers the subject more ways to search memory. Apparently, differences in expertise and organization did not prove to be more beneficial until the more difficult recall task was performed.

Nevertheless, expert participants who chose a conceptual organization had better recall, fewer recognition errors, and more structured recall, in terms of recalling reserved words in the categories in which they were studied, than their language-organizing peers. This suggests that programming language information, like natural language information, is more profitably and efficiently organized conceptually in the human brain (or at least for the purpose of this task).

Novice participants who used neither a language nor a conceptual organization were able to improve their recall slightly over participants who organized by language. It is presumed that they relied upon standard memorization and mnemonic techniques. The novices who organized by language, however, performed consistently with short-term memory performance predicted by Miller (1956, as cited in Anderson, 1985), with an average of seven. This suggests that attention to language details may actually impair memory.

Among the interesting findings of this study was an intermediate participant who organized by conceptual categories. However, unlike most of the other intermediate and expert participants who placed the cards into "neat" categories by
overlapping them as they were spread out on the table, this participant organized them much more loosely, not overlapping any of the cards. This suggest that he may not have a tight organizational strategy in his mind for this domain of knowledge.

One expert, who organized the cards by language, reported that he placed them in an order which corresponded to an order in which they might appear in a program. This suggests that programmers may possess scripts (Mandler, 1984) which contain knowledge concerning valid programs, and are activated when coding in a particular programming language.

Interference seemed to occur for many participants in the recognition task when similar constructs were included from several languages, e.g., dimension, from Fortran, which was in the list, and dim, from Basic, which was used as a foil. One expert reported that he recognized several incorrectly because he reported "Yes" to several in which a word from the same conceptual category was included, but from a language that was not part of the list. The topic of interference as it related to knowledge of several programming languages will require further examination in future studies.

Implications

The results of this study should be considered exploratory in nature, both because of the small number of participants and because of the artificial nature of the task employed. However,
several possible implications might be applied to programming language learning.

The linguistic model of various types of bilingualism, adapted in order to describe the knowledge that computer programmers have about programming languages, predicts that individual participants have a dominant strategy, either a language strategy (coordinate) or a conceptual category strategy (compound) which dominates one's thinking about different programming languages. This would explain why participants chose a strategy for organizing the cards and did not shift their strategy. That is, once the cards had been organized by language, even though they might have seen the conceptual categories, they did not reorganize the cards.

If recall percentage can be considered an indicator of programming knowledge and experience (based on the high correlation between recall and experience level), then this study lends strength to the observations of many researchers (Shneiderman, 1977; Mayer, 1979; Shneiderman & Mayer, 1979; Weinberg, 1982) that it is more important to teach programming concepts and generalizable skills than the syntax of specific languages. It is possible that over-attention to syntax may actually obscure, rather than facilitate, recall. This may suggest that an over-attention to syntax prevents the development of an abstracted model of computer programming.
It would seem reasonable to propose that a course in the Structure of Programming Languages, which compares the structure of a variety of computer programming languages, and which is offered in most larger computer science programs, be required of all computer science majors. This course is usually offered as a junior or senior-level elective, presumably for those going to graduate school or in preparation for a career as a systems analyst. However, this course is downplayed, and could be of great value to those preparing to be programmers.

The usefulness of a recall test, similar to this one, by employers as a measure of generalizable programming ability cannot be concluded, for three reasons. First, the artificialness of the recall and recognition tasks cannot realistically assess programming ability. They may have no bearing on more realistic programming situations where programming manuals are usually readily available.

Second, the design of the study, due to limitations on the selection of reserved words, was not powerful enough to make this type of recommendation. A set of sixteen words was not large enough to spread out participants. However, the marvelous adaptation of the human mind to learning tasks might prevent a larger spread even if this study was replicated with a larger set of reserved words to be learned.
Third, it is quite likely that the organizational pattern chosen by participants, and their subsequent recall, are both related to another causal factor, possibly the method by which they learned computer programming. It is possible that the organization chosen may indicate a dominant pattern of organization for individual programmers. This information may be of value to, e.g., employers who seek programmers who will be programming in a variety of languages, and who must be flexible among them. On the other hand, if a programmer is sought who will program in only one language and not mix languages, a co-ordinate type of programmer might be distinguished from this task.

Further research must be conducted to make a possible recommendation for the optimal order of languages to be learned by the future computer scientist to maximize positive transfer and minimize negative transfer. Future research might also ask programmers to write a short bit of code in each of several languages and examine interference patterns. This type of emphasis on actual coding tasks in a variety of languages might provide a more realistic assessment of differential programming ability in several programming languages.
References


Vita

Lynn Teresa Cunningham was born in Naperville, Illinois on September 4, 1964. She completed her bachelor’s degree in computer science at Millersville University of Pennsylvania in December 1985. Following, she entered graduate study at Virginia Tech and has served as graduate assistant/programmer at the Educational Technologies Division of the Learning Resources Center at Virginia Tech.