COGNITIVE MODEL AND PROBLEM-SOLVING PROCESSES OF COMPUTER PROGRAMMERS

BY

(c) D. M. BENOIT 1988

SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR A DEGREE IN
MASTER OF APPLIED SCIENCE (I/O PSYCHOLOGY)

DEPARTMENT OF PSYCHOLOGY
SAINT MARY'S UNIVERSITY
HALIFAX, NOVA SCOTIA
MARCH 1988

ISBN 0-315-45059-2

COGNITIVE MODEL AND PROBLEM-SOLVING PROCESSES OF COMPUTER PROGRAMMERS

D.M. BENOIT

SUBMITTED IN PARTIAL FULMILLMENT OF
THE REQUIREMENTS FOR A DEGREE IN
MASTER OF APPLIED SCIENCES (I/O PSYCHOLOGY)
SAINT MARY'S UNIVERSITY
HALIFAX, NOVA SCOTIA

APPROVED: _____
V.C. CATANO
THESIS ADVISOR

_____
KEN HILL
COMMITTEE MEMBER

_____
D. SHERIDAN
COMMITTEE MEMBER

March 30, 1988
DATE

For Daniel, who believed in me,
For Paulette, who taught me courage,
For Cheryl and J.J., who helped me see this through, and
For Vic Catano, who made it worth reading.

Abstract

COGNITIVE MODEL AND PROBLEM-SOLVING PROCESSES OF COMPUTER PROGRAMMERS

D. M. Benoit

Although researchers have started concentrating on the programming behavior of computer programmers, the material produced so far has not been merged into a testable theory; each study focuses on a particular problem without integrating the results into a workable overall model. In addition, most studies have concentrated on the measure of programming skills and aptitudes, rather than on the underlying cognitive processes differentiating programmers from non-programmers. While many studies focus on the human-system interaction, very little has been attempted to specify which strategies computer programmers use to solve problems, how they process and integrate information; whether these strategies are specific to a certain type of people, or whether these processes can be

taught and improve with experience. Two partial models were examined, the syntactic/semantic model of programmer behavior, and the heuristic/algorithmic problem-solving model, in order to attempt to build a stronger base for the evaluation of programmer aptitudes.

Two major groups were compared: an Experienced Programmers Group (n = 31) and a General Population Group (n = 44), which was further divided into three groups, a Control Group (n = 23), a Novice Programmers group (n = 11), and a Non-programmers group (n = 10). Experienced Programmers and General Population were compared on two tests, the Semantic Ability Test, and the Master Mind Game. The Control Group and the Novice Programmers were retested after treatment, which consisted of a PSYCH course for the Controls and a BASIC programming course for the Novice Programmers.

Analyses indicated that: on Master Mind, Experienced Programmers performed better than the General Population; Novice Programmers performed better than the Control Group; and Novice Programmers performance was related to their grade on the BASIC course. On Semantic Ability Test, Experienced Programmers performed better than General Population, although not significantly so; Novice Programmers performance was related to their grade on the BASIC course, although not significantly so.

Results are discussed within the context of implications for the measure of computer programming aptitudes. The Master Mind game seems to tap into the problem-solving processes, and seem to indicate that the heuristic/algorithmic problem-solving model may be valid. On the other hand, the Semantic Ability Test may have serious procedural limitations, being very sensitive to its manipulation. Due to these difficulties with the Semantic Ability test, results are inconclusive.

Table of Contents

page

Tables and Figures

page

Introduction

In the last few years, research has begun to focus on the human factors of computer programming instead of concentrating on the mechanical aspects of the discipline, and such human-centered issues as programming skills are being isolated from machine-centered issues. This movement has enabled psychologists, psycholinguists and computer scientists to study programmer behavior separately from programming procedures.

Weinberg's book "Psychology of Programming" (1971) has prompted researchers to start concentrating on the cognitive processes of programmers. This type of study has taken the form of examining programming tasks such as composition, comprehension, debugging, and modification, as well as learning of programming skills (Shneiderman and Mayer, 1979). The subjects widely range from naive to highly experienced programmers, and the tasks from simple to complex.

Notwithstanding this shift in interest, the material produced so far has not been merged into a testable theory; each paper focuses on a particular problem without integrating the results into a workable overall model. In addition, most

studies have concentrated on the measure of programming skills and aptitudes, rather than on the underlying cognitive processes differentiating programmers from non-programmers.

There is no comprehensive theory or model, at this point, that specifies what strategies programmers use to solve problems and how they process information, whether these strategies are specific to a certain type of people, or whether these processes can be taught and improve with practice.

When writing a computer program, programmers start from a problem and attempt to find the best solution. To do so, they use a programming language, and apply some form of problem solving strategies to shape the solution. But how can we determine if a particular strategy works best? And how is it possible to find whether the programmer has been successful in this process?

Shneiderman (1977) asserts that measuring the quality and ease of comprehension of a computer program is very difficult; he cites Gilb's description of "up to 40 metrics of program quality such as reliability, maintainability, repairability, accuracy, generality, portability, logical complexity, modularity, efficiency, total system cost, operational cost and stability" (p. 465). It is argued that these measures are faulty through their generality, their lack of validation and their

potential lack of relevance. In addition, some of these measures may be mutually exclusive; for example, a program that is very complex may be difficult to change or maintain due to its nature. Shneiderman himself found that commenting, mnemonic variable names and modular program design had significant impact on comprehension and program quality (i.e. written code), whereas indentation and flowcharting did not.

Factors such as programming techniques, the features of the various programming languages, teaching procedures and difficulty of the problem may influence an individual's ability to be successful at programming, both in terms of program comprehension and program quality. (It is very important that a programmer be able to understand an already written program, in order to make appropriate corrections or modifications).

Even accounting for these differences, the basic principles behind programming are similar in any programming language. Better performance may depend on the comprehension of these principles; certain individuals may have a greater ability understanding them, thus making it easier to write good programs (for example, transfer from one programming language to the other would be less difficult). On the other hand, the information acquired by computer programmers may be more extensive and better organized than the novice's information, thus

making it easer to arrive at an ideal solution (Mayer, 1981).

Taking for granted that everybody has a certain ability for solving problems, the focus of this research has been to investigate whether computer programmers need to use a specific type of problem solving strategy in order to write good programs. Based on what has been discussed previously, two major questions can be posed:

1. Can the processes underlying programming aptitudes be identified?

2. Can the measure of these processes be a good predictor of success in computer programming?

A review of the literature indicates that specific abilities are needed to be successful at computer programming, although there is still contention as to which ones are essential. In addition, cognitive processes appear to be a crucial factor in enhancing these abilities. If these abilities could be measured, it would then become possible to predict who would be successful at programming, thus cutting down training and labor costs. Therefore, the goal of this research is to outline the design of a better predictive test (or test battery), by underlining the necessary cognitive processes before the breakdown of the important elements of computer

programmer ability.

Purpose of the study. The purpose of this study was to verify two partial models: the syntactic/semantic model of programmer behavior, and the heuristic/algorithmic problem-solving model. By examining both processes in isolation, the study attempted to build a stronger base for the evaluation of programmer aptitudes. It must be noted, however, that this can only be part of a long-term project; consequently, the present study is considered exploratory in nature.

Background research for this study indicates that programmers may have better control over semantic elements in language comprehension and over algorithmic processes in problem-solving. It was considered desirable to measure these two processes separately. The present research has attempted to do so, by trying to answer two main questions:

1. Are there separate measures that can differentiate the cognitive processes, such as semantic ability and algorithmic problem-solving, of programmers and non-programmers?

2. Can these semantic and algorithmic measures predict, in a general way,

success in a programming course?

What is needed is an approach starting with an approximation of the way in which programmers process information, rather than one which is looking into special abilities.

This paper has five main sections. In the first, the fundamental notions underlying the models are presented, including the basic assumptions for the information-processing models of cognition. The second section describes the two partial models, and discusses the validated measures of programming aptitudes available as well as the theorems supporting this study. In the third section, the problem is stated and the measures described. The Research Methodology, including the discussion of the Results obtained is included in the fourth section. The fifth section discusses in detail the results obtained.

Background

The field of cognition is dedicated to the study of how individuals gather, store, retrieve, and utilize information (Carroll, 1983). This is done through the examination of cognitive processes such as language, perception, retention, transfer and memory; these processes are interdependent and are often difficult to

separate or discuss in isolation. To illustrate, it would be impossible to explain what is a dog if the concept, the memory of that concept, or the label "dog" were not available to the individual. Similarly, solving a problem without the use of concepts and language is difficult to imagine. This study is concerned with three important processes of cognition: concept formation, problem solving, and language. They will be treated separately as much as possible.

The study of cognitive processes is performed usually through cognitive tasks. A cognitive task is "one that critically requires the processing of information -- information from the outside world that can be perceived by the individual and placed in some kind of memory, and/or information derived from previous experiences and retrieved from memory" (Carroll, 1983, p.3). That information can be processed in many ways: it can be stored, compared with other similar information, retrieved, modified, or "manipulated by complex procedures or algorithms" (Carroll, 1983, p.3). Cognitive tasks are tailor-made problems that examine specific areas of a process. They enable the researchers to break down the larger field of cognition.

Two basic classes of theories of concept learning and problem solving have made use of cognitive tasks to demonstrate their characteristics:

The continuity theory is an associationist model: it attempts to apply classical and instrumental conditioning principles to concept formation. Associative psychologists explain problem solving in terms of response hierarchy: specific responses become associated with specific stimuli because of repeated pairings or reinforcement (high strength response). When the previously learned responses are inadequate to the situation, a problem is identified, and it stays unsolved until a low probability response has been aroused (Lipman, 1979). For example, in a classic experiment where a pair of scissors had to be used as a balancing weight instead of what it is normally used for, it took longer for the subjects to arrive at that solution due to its low association response (Meyer,1979).

The noncontinuity theory derives from the Gestalt approach. It contends that concept learning is a process whereby hypotheses are constructed and tested until the appropriate one is found. The problem solver makes a guess (a hypothesis) as to the solution, in accordance with the information available. As more information is gained, the individual either confirms the hypothesis, or rejects it and selects a new one.

The noncontinuity theory also includes the information processing approach, which defines the existence of a problem when a series of alternatives exist for the same solution (Meyer, 1979). This approach elaborates the type of

strategies or procedures a problem solver chooses while adopting one of the alternatives to the solution. The development of information-processing theories has led researchers to break down problem-solving operations and write computer programs that mimic human problem-solving strategies. One of the most prominent and most quoted attempt to do so is Newell and Simon's (1972) General Problem Solver (GPS) program, which uses heuristic-type methods for solving problems.

In this study, the noncontinuity theory and the information processing approach have been adopted.

Problem-solving Processes

Concept learning. When an individual responds to several stimuli in the same way, that person has made a categorization, and is said to have acquired a "concept". For instance, identifying animals that have a beak, wings and feathers as being birds is to have acquired the concept "bird". This type of categorization reduces strain on memory (since not every instance of every object has to be kept in memory) by developing an abstraction independent of any particular object.

Four basic factors affect concept learning:

a. Number of attributes. The more attributes (or aspects, facets) a concept has, the more difficult it is to learn (Bulgarella and Archer, 1962).

b. Positive and negative instances. Learning is easier if only positive instances are encountered. However, in order to rule out irrelevant attributes, negative instances are essential (Johnson, 1971).

c. Cue salience. The salience of an attribute (or how much it stands out compared to others) also affects concept learning. The more different attributes are, the easier they are to learn (Trabasso, 1963). For instance, it is easier to differentiate between a circle and a square, than between an hexagon and an octagon.

d. Feedback. Learning is easier if it is accompanied by confirmation or infirmation of the relevancy of an attribute (Bourne and Pendleton, 1958); feedback also helps in the formulation of hypotheses. Hypotheses "represent systematic attempts to eliminate or confirm the role of the many varying dimensions in a situation" (Meyer, 1979).

The most often cited research in the area of concept learning and hypothesis testing was done by Bruner, Goodnow and Austin (1956). In their experiment, they used a set of 81 stimuli, consisting of four dimensions with three attributes per dimension (e.g., a card with two borders and three black crosses in the center). They presented one card at a time; their subjects had to decide on which attribute was the correct concept (selected by the experimenters). The experimenters then told the subjects if the card was a "positive instance" (a card representing the concept) or a "negative instance" (a card without the concept).

By observing their subjects, Bruner et al. (1956) arrived at two major types of selection strategies for concept learning: scanning, where every instance is examined and kept in memory, and focusing, where one instance is examined and discarded if found unsuitable. They found that the focusing strategies were more efficient, because scanning relies too heavily on memory. The more attributes there are, the more complex the problem becomes. For example, while a three-attribute problem has seven possible positive concepts, a six-attribute problem has 63 concepts. Consequently, it "is quite evident that the task of keeping track of possible hypotheses increases considerably in difficulty with an increase in the number of attributes in the array" (Bruner et al., 1956). The focusing strategies can be divided in two:

a. Conservative focusing, where an individual picks one concept then changes one attribute of that concept at a time, thus eliminating unusable attributes directly. With this strategy, the subject would always arrive at the right answer;

b. Focus gambling, where an individual picks one good concept, then changes several attributes at a time, thus trying to "guess" the right attributes.

Conservative focusing was the best strategy to use in solving a problem: the information was more easily monitored, and it minimized the amount of risk involved. However, the more attributes there were, the more costly it was in terms of time since only one attribute is changed at a time.

On the other hand, focus gambling "provides a way of attaining the concept in fewer trials" (Bruner et al., 1956). The individual is taking chances by changing several attributes at a time, and may "get lucky". Bruner et al. found that most people used focus gambling as a strategy.

As discussed, concept learning is based on hypothesis testing, which is an essential process in problem solving.

Problem solving. What is a problem? Glass et al. (1979) define a problem

as something which has no immediate solution. They identify three basic components to a problem: availability of specific information, the use of a series of operations by the problem solver to arrive at a solution, and the definition of the solution to the problem, or the "goal". Prior learning, the complexity of the problem, and the embeddedness of clues (Lipman, 1979) -- as opposed to their salience -- can all affect problem solving. In a general way, there are four major steps to the problem-solving process (Polya, 1957): understanding the problem, searching for a solution, implementing the solution, and checking the results.

When an individual is faced with a problem, the first step is to attempt to understand that problem. One way to attempt this is to arrive at an internal representation of the problem. Thus an algebraic expression may take the form of more concrete objects such as apples and oranges (Luria, 1968). Of course, as a problem becomes more complex, it also becomes more difficult to represent it concretely; similarly, Pellegrino (1985) found that "the more elements and the more transformations, the longer it will take to solve the problem" (p.51). In addition, "a problem may have more than one representation...[and] some problems can be solved much more easily with one form of representation than another" (Glass et al., 1979, p.400). Therefore, "understanding the nature of the task and defining what is necessary for a solution is usually a major step toward

finding a representation that can be used effectively to solve the problem" (Glass et al., 1979, p.403).

Planning the solution, implementing it and verifying the results often operate simultaneously. In order to arrive at a solution, the problem solver can use two very different strategies or procedures: Heuristics or Algorithms.

Heuristic reasoning is "reasoning not regarded as final and strict but as provisional and plausible only, whose purpose is to discover the solution of the present problem" (Newell and Simon, 1957), whereas algorithmic thinking is "a search method which, with certainty, will produce the correct response for any stimulus in the set of possible stimuli" (Hunt, 1962). An algorithm is "an infallible, step-by-step recipe for obtaining a prespecified result. 'Infallible' means the procedure is guaranteed to succeed positively in a finite number of steps..." (Haugeland, 1985); thus, an algorithmic strategy requires that every hypotheses be tested until the best solution is found. This strategy can be costly in terms of time, although it was found that "the best reasoners are often slower at encoding than are less skilled reasoners...slower, more accurate encoding of information at the outset speeds up subsequent processes" (Pellegrino, 1985, p.52). However, "solvers worked faster than nonsolvers" (Rowe, 1985, p.335), suggesting that because solvers are more careful at understanding the problem, they arrive at a solution

more quickly. The algorithm which works best at problem solving is the "branched schedule" (or primitive) algorithm, because it works by arriving at the solution based on conditional branches (or answering yes/no questions). This means that:

> an entire primitive algorithm could legitimately serve as a single "primitive" instruction in a more sophisticated algorithm. In other words, if we think of primitive algorithms as ground level, then second-level algorithms can use simple operations and directives that are actually defined by whole ground-level algorithms; and third-level algorithms could use second-level algorithms as their primitives, and so forth...using these bricks, and algorithmic glue, there's no limit to how high we can build (Haugeland, 1985, p.70-71).

This process, albeit slow, enables the problem solver to solve a problem more quickly than if a "straight schedule" algorithm were used, where every instance of a solution, regardless of its usefulness, is considered one after the other.

A heuristic strategy will usually find good, although not necessarily optimum solutions; the individual is willing to accept any nontrivial approximate solution that can be obtained in a reasonable amount of time.

Newell and Simon (1972) give a larger definition to the heuristic strategy through the means-end analysis method of problem-solving. This method requires the problem solver to determine the goal or the solution, then to decide on the means to reach that solution. This is done through establishing subgoals, which

provide the information necessary to reach the final solution. The heuristic component is the fact that the problem solver does not start to generate hypotheses which are tested one after the other; rather, the problem solver starts by applying the "problem-reduction" approach (Nilsson, 1971), eliminating the hypotheses which are obviously wrong or useless. The formation and testing of subgoals, however, has a definite algorithmic component.

This is best demonstrated through Egan and Greeno's (1974) experiment with the three-disk Tower of Hanoi problem, where it was demonstrated that planning and setting up subgoals were extremely important; in addition, the more complex the problem, the more possibility of error. This is because some subgoals often bring the problem solver away from the final goal (Thomas, 1974). Glass et al. (1979, p. 326) confirm this by stating that "...people rely on a limited number of heuristic principles which reduce the complex task of assessing probabilities and predicting values to simpler judgemental operations. In general, these heuristics are quite useful, but sometimes they lead to severe and systematic errors".

Newell and Simon concentrated on the problem-solving search, rather than the actual solving of the problem; that is, they argued that the method for solving a problem was arrived at heuristically -- the actual solution was most often done algorithmically.

Haugeland (1985) argues that GPS was based on two main assumptions which proved to be false. First, means-end analysis turned out to be more effective for a very specialized form of problem-solving (i.e. for very narrow problems), rather than having a single application for a wide range of problem types. Second, the "...second unfulfilled assumption undermines not only GPS, but heuristic search in general [...] It is that "formulating" a problem is the smaller job, compared to solving it once formulated" (Haugeland, p.183).

Therefore, it can be advanced that although people use heuristics to decide how a problem is going to be solved, algorithmic procedures could be used to apply the solution. In a sense, Newell and Simon's argument is more on the definition of an algorithm (an infallible, time-consuming, non-economic procedure). However, when distinctions are made between types of algorithms, such as what Haugeland (1985) has done, it becomes possible to apply a 'conditional' algorithm to GPS procedures. This approach is the understructure in the development of a model of cognitive processes of computer programmers.

Language

One of the very complex and most controversial processes of cognition is language, or more specifically language- comprehension. How do people

comprehend language, and in this case, the English language? Because "speech acts express thoughts...nearly all speech acts express corresponding cognitive states or events..." (Haugeland, 1985, p.89). Nevertheless, speech must not only be uttered, it must also be understood: "Order is the opposite of chaos. An ordered text must have a systematic internal structure that is not accidental" (Haugeland, 1985, p. 94), thus delivering an internal representation of the concept to be expressed.

A major linguistic breakthrough was the development of the theory of transformational grammar by Noam Chomsky in 1957. Although Chomsky's theories no longer dominate linguistics, he revolutionized the field by developing a generative grammar. This grammar contains a set of rules for the formation of any grammatical sentence in a language; it takes the study of language away from a descriptive grammar, that is, the study of how sounds are understood to form words. Chomsky's focus was the total competence, or the pure syntactic formulation of language, without due concern about the meaning of texts (i.e. semantics), or individual differences such as motivational levels, short- term and long-term memory or information retrieval capabilities (Kintsch, 1977).

Today, linguists realize that isolating language competence from performance is unrealistic, and current theories include both syntactic and

semantic elements. Marks and Miller (1964), for example, through scrambled and unscrambled sentences, demonstrated that language processes were dependent on both syntactic and semantic rules.

Syntactic rules. In Chomsky's theory, there are two types of syntactic rules: phrase structure rules and transformational rules (Kintsch, 1977).

The phrase structure rules group (or chunk) sentences into phrases and determine the relationships between these phrases. For example, the sentence "The small boy ate the red apple" consists of a noun phrase (The small boy) and a verb phrase (ate the red apple) which also contains another noun phrase (the red apple). These rules are the basis for perception and memory of the sentence, since they break it down into manageable components.

Transformational rules enable relationships between syntactically related sentences; for example, the active transformation "The boy ate the red apple" can be changed to the passive transformation "The red apple was eaten by the boy".

These rules assume that a sentence has "an abstract underlying phrase structure...the deep structure of the sentence" (Kintsch, 1977, p.311). This deep structure "determines the semantic interpretation of the sentence" (Paivio, 1971,

p.399). Thus, transformational rules are mainly used to "convert the deep structure into the more concrete surface structure that describes the form of the sentence" (Paivio, 1971, p.400). It was therefore argued that syntactic (both structure and transformation) rules were essential to sentence comprehension. In other words, while semantics were a complement to syntax for language competence, syntactic elements formed the basis of expression.

Further study, however, showed the converse: meaning takes precedence to sentence organisation. For example, Aaronson and Scarborough (1976) found that semantics, rather than the syntactic structure, affected reading time when comprehension was required of their subjects. Bever (1970) pointed out that syntactic organisation was less important to people than semantic representation, and that syntax was used merely as a support to help determine the meaning of a sentence.

Semantic rules. Experiments by Gough (1965) and Slobin (1966) demonstrated that semantic elements were the most important for sentence comprehension, sometimes even suppressing syntactic rules. Grammar, or syntax, does not address meaning for lexical units of sentences. Consequently, two syntactically identical sentences, such as "The boy hit the girl" and "The girl hit the boy", have different meanings (Gough, 1965). The words themselves, or the way

they are combined in text, can facilitate or hinder comprehension. Semantic cues (or rules) establish a basis for understanding. For instance, all words are not semantically equal; some are more complex than others, and affect comprehension and memory. For example, the words "short", "bad", "narrow" (as opposed to "long", "good", and "wide") cannot be used to ask a neutral question; in addition, the "marked" word is part of a pair whose other member is always "unmarked" (Kintsch, 1977).

As a result, one can ask "How good is the meal?" without assuming that it is bad or good; the opposite, however, "How bad is the meal?" assumes a degree of badness rather than a neutral quality. Clark (1969) demonstrated that the unmarked words are easier to comprehend and remember, and that retrieval is more rapid if the question is congruent with the statement in word problems (e.g. "If John is better than Pete, then who is best?" as opposed to "If John is better than Pete, then who is worse?")

Most studies on comprehension confirm the importance of grammar in language. More importantly, however, they show "that word meaning plays a crucial role in comprehension...a role that overrides syntactical information unless care is taken to suppress relevant semantic cues" (Paivio, 1971, p.47).

Sentence processing. The main purpose of syntactic and semantic rules is to assist comprehension of language. Researchers have been interested in determining how people process sentences for comprehension, and how long-term and short-term memory play a role in sentence processing. Begg and Wickelgren (1974) established that semantic information was learned and retained better by their subjects than syntactic or lexical information. Sachs (1967) tested syntactic/semantic recognition of sentences in text. Subjects listened to sections of text; immediately after each section, a sentence from the text was repeated. The repeated sentence was identical or changed slightly. The changes were either syntactic or semantic in nature, and the subjects had to decide whether the sentence was "changed" or "identical". If the sentence appeared early in the text, only its semantic aspect (its meaning) could be remembered, and semantic changes were detected better than syntactic changes.

Sachs concluded that, when delayed recall was required, the subjects transformed the sentences into an internalized semantic interpretation, while the syntactic form was forgotten. As recall operated, reconstruction was done through the remembered meaning. As the repeated sentences approached the end of the text, the accuracy of recall increased. One way to determine if language comprehension is an ability which can be developed is to study individual

differences in the utilization of language. Perfetti (1983) notes differences in language ability:

> Some people read well, have large vocabularies, and score high on verbal intelligence tests. Others read with difficulty, have smaller vocabularies, and score lower on verbal intelligence tests. What processes underlie such pervasive differences in verbal ability? (p.65).

Perfetti advances that the differentiation between good and poor verbal ability comes from the ability to retrieve information efficiently. Studies by Jackson and MacLelland (1979) and Jackson (1980) confirm that name retrieval is a major factor in adult verbal ability. Most studies, however, have been done with simple verbal processes, processes which require only one step in the retrieval process. They occur when semantic elements associated to a word are activated in memory.

Complex verbal processes are components of verbal ability and are more difficult to evaluate. A complex verbal process is "one which requires multiple memory access and manipulations of accessed units...decoding may sometimes be complex and comprehension may sometimes be simple" (Perfetti, 1983, p.66).

Most studies, such as Perfetti and Goldman (1976), Lesgold and Perfetti (1978), and Jackson and McLelland (1979) (cited in Perfetti, 1983), have

concentrated on the study of simple verbal processes. However, Perfetti (1983) claims that, since language is not unidimensional, complex processes reflect verbal processing more accurately.

Perfetti's discussion concurs with Paivio's argument that current linguistic theories are incomplete; Paivio has attempted to discuss psycholinguistics "with special emphasis on the mediational role of imagery" (Paivio, 1971, p.394). He developed the dual-coding model of psycholinguistics, in which concrete sentences are stored in memory verbally and in the form of nonverbal imagery, whereas abstract sentences are stored in verbal form only. He criticizes comprehension studies using test sentences by stating that "the sentences, when concrete, may be decoded instead into nonverbal imagery and the information in the image compared with the information in the referent picture or sentence" (p.417), the measure being corrupted by an additional process. He adds that in language, the more abstract the concept, the less nonverbal imagery plays a role:

> ...verbal descriptions of concrete situations and events from memory and verbal expressions of the manipulation of spatial concepts are likely to be mediated efficiently by nonverbal imagery, whereas abstract discourse and verbal expressions of abstract reasoning are more likely to be mediated entirely by the verbal system (p.434).

Paivio's theory is based in part on a study by Begg and Paivio (1969), in which they investigated the relationship between concreteness and imagery, and sentence meaning. Based on Sachs' findings (1967), they arrived at two major hypotheses:

1. Sachs found that the majority of her subjects could recognize semantic changes more easily than syntactic changes. Begg and Paivio thought this was because her texts were concrete, thus creating an image which was disturbed when the semantics were transformed. Therefore, by using concrete sentences, they would be able to replicate Sachs' findings.

2. By testing subjects with abstract sentences, they would find a greater difference among subjects. The more abstract the sentence, the lower the semantic recognition.

Both hypotheses were confirmed, through a modification of Sachs' procedure. Begg and Paivio tested subjects with either concrete or abstract sentences, modifying them semantically or syntactically. They found that subjects were able to recognize semantic changes easily in the concrete sentences (e.g., The sharp arrow pierced a frantic bird), but had more difficulty with the abstract sentences (e.g., The arbitrary regulation provoked a civil complaint). They

concluded that imagery was used to understand meaning in concrete sentences, but that the abstract sentences were mediated mainly by verbal processes.

Paivio's dual-coding model has been supported by research. It makes sense in that it relies "on "knowledge of the world" as a crucial substrate of language performance...imagery plays an important role in the comprehension, retention, and production of concrete (descriptive) language in particular, whereas the processing of abstract language is assumed to be tied more closely to the linguistic representational system alone" (p.476). The model can be used to test language acquisition, as well as language ability, which is of concern in this present study.

Cognitive models of computer programmers

The literature reviewed above forms the basis for the study of the cognitive processes specific to computer programmers. Although a substantial body of research relates to the use of application programs as tools, only a small portion of it has considered the cognitive processes required to be a good programmer. As with the previous section, this review is divided in two parts, based on two partial models: (a) the syntactic/semantic model of programmer behavior (Shneiderman and Mayer, 1977), and (b) the heuristic/algorithmic model of computer programming behavior. The present study is based on these two models.

According to Shneiderman and Mayer (1979), a model of cognitive processes for programmers must address five main programming tasks: composition (how to write a program), comprehension (how to understand a problem to translate it into a program), debugging (how to find errors in an already written program), and learning (how to acquire new programming skills) (p.221). The model must account for the strategies and processes used to acquire, retain, retrieve, and integrate the information necessary to perform programming tasks effectively. The following literature review summarizes the research done to date on the cognitive processes of computer programmers.

Syntactic/semantic model of programmer behavior

Shneiderman and Mayer (1977) proposed the syntactic/semantic model when they studied the recall scores of experienced and novice programmers with two types of programs: one executable program and one shuffled program. They found that, as the experience level of the programmers increased, there was also an increase in the ability to remember the executable program; there was minimal change for the shuffled program. During the recall phase, the experienced subjects would tend to retain the semantics of the program while writing a syntactically different program than the original, whereas novices tried to remember the exact lines of the program. That behavior led to the

syntactic/semantic model of programmer behavior.

Shneiderman and Mayer hypothesized that as programmers become experienced, their capacity to recognize programming structures increases; this ability helps them to recode the syntax of the program into a more abstract level of internal semantic structure.

In the case of programming, syntactic knowledge is language dependent and is acquired by rote memorization, whereas semantic knowledge depends on the understanding of general concepts that are not related to any specific programming language. The knowledge is organized hierarchically from low level constructs to a higher level problem domain. Kahney (1983) has found evidence that " 'average' novices spend considerable time trying to come to grips with concepts like recursion ... the average novice commits a segment of code to memory with the rule that the segment has a particular effect without having a model of the way the effect is achieved" (p.127). So while a more "talented" programmer can manipulate the recursion concept to write a program and determine its output, the average programmer "...writes recursion procedures without a model of the behavior of the program, and therefore needs to use the computer to evaluate code" (p.129). Without understanding the structure, or the semantics, of the program, the code, or syntax, is simply copied and less easily adaptable to more complex problems.

Shneiderman and Mayer's study relies on Feigenbaum's (1970). "Information processing theory of memory". Feigenbaum asserts that information is channelled from short-term and long-term memory into "working memory", where it is integrated before it is used. As an example, an individual would extract (from short-term memory) the information necessary about a problem and from long-term memory the strategies to solve it, transfer the two in working memory and arrive at a solution. This process describes program composition, and the five programming tasks mentioned above can also be described through the working memory process. According to Shneiderman, the best definition of program comprehension is "the recognition of the overall function of the program, an understanding of intermediate level processes including program organisation and comprehension of the function of each statement in a program".

Thus the semantic and syntactic information is stored in long-term memory, albeit differently. Shneiderman and Mayer (1979) describe syntactic knowledge as being "More precise, detailed and arbitrary" (p.222), with some points in common but generally unique to each programming language. Semantic language, on the other hand, is stacked in memory, the low level details being readily accessible, and high level concepts requiring a more abstract understanding of programming principles. Semantic knowledge can cut across several programming languages,

and is independent of syntactic knowledge. The syntactic/semantic model proposes that once semantics have been applied to the problem, the use of syntactic elements to write the program becomes simple, the languages to be used interchangeably.

In program comprehension, syntax and semantics are used in the same way as in the comprehension of English text, the semantic elements being more important for comprehension than syntax (see Gough (1965) in previous section). Shneiderman and Mayer (1979) add that "the programmer, with the aid of his or her syntactic knowledge of the language, constructs a multileveled internal semantic structure to represent the program" (p.226). They parallelled their findings on recall of programs with Sachs' study (1967) of syntactic/semantic recognition of sentences in text; experienced programmers retained semantic information better than syntactic elements, which is consistent with Sachs' (1967) and Begg and Paivio's (1969) studies. Shneiderman and Mayer (1969) further stressed their belief "that ability to memorize and recall a program is a strong correlate of program comprehension" (p.235).

Modal Model of Programmer Behavior. Kahney (1983) studied how novice programmers transform the verbal statement of a problem into a program. He describes the "modal model of problem solving" which has three phases:

1. A phase of problem understanding. Usually, a programming problem is not well defined, and some variables might be missing. Programmers have to be able to recognize the important aspects of the problem statement. Kahney states that the resolution of the problem starts with the understanding of the problem, since a specific tactical plan must be adopted.

2. A phase somewhere between problem understanding and the running of "solution" processes, usually called "method finding". Chi, Feltovitch and Glaser (cited in Kahney, 1983) found that novices are "stimulus bound": their problem solving strategies are influenced more by each text than by the general principles underlying all similar problems, while the experts are capable to extract these general principles from the problem. The ordered collection of information into structured knowledge is essential to the derivation of these general principles, and is the first step toward solving the problem. Kahney adds that to solve the problem, it is essential that they develop "a mental model of a problem" (p. 127). Kahney implies here that certain novice programmers develop a model usable in more than one situation, while others commit a segment of code to memory without understanding its underlying principles. He further says that programming and experience "interact to direct and constrain the mental models that are constructed". This mental model is obviously very similar to Shneiderman and

Mayer's (1979) model of hierarchical semantic knowledge.

3. A solution phase. This phase is often seen as a "solution framework into which elements of the problem are slotted" (Kahney, 1983, p. 123). Kahney found that novices did not experience any difficulty with writing code, although they had difficulty understanding code that was written by someone else. In the case of the written code the problem is reversed because the novice has the solution and must determine the problem. Novices fail to react to the written code in the appropriate way (i.e. by stating the purpose of the program), which may indicate that a different type of problem-solving behavior may be required for solving more complex problems.

In another experiment by Kahney (1983), talented and average novices were given a sample program to study, and then had to write a similar but more difficult program. Kahney found that talented novices worked problems the same way as the experienced programmers. They developed a mental model of how the principle of the sample program worked, generalized its application, then solved the more difficult program. Most of the average novices were not able to write the more difficult program, which could indicate that there may be certain "natural" abilities, aptitudes or skills influencing a programmer's ability to transfer a problem into workable code.

Kahney's modal model of problem solving is similar to Bresnan's model (1981) in which she equated cognitive processes in the mental representation of language to the solution of a problem of natural information processing, where computational theory (which establishes the constraints or limits imposed on the representation), algorithm (which properly interprets the information obtained), and process (in which the mental representation is compared with the "processes of the natural system" (p.40-41) are integrated.

In both models, before starting on the solution, the individual has (a) to identify the problem and its elements, (b) compare these elements to already acquired knowledge, then (c) code the answer.

These models, however, do not mention how these processes are "internalized", how this knowledge is structured within the individual, and how it is retrieved.

Programmer's Internal Program. Sengler (1983) proposed a model of program understanding which is also concurrent with Shneiderman and Mayer's model. He states that understanding a program requires two general capabilities: the individual must be able to predict what the program will do, and must be able to recognize that a certain part of the effect is caused by a certain part of the

program. The latter is especially necessary for modification purposes and debugging.

When an individual is reading text, not every character is considered separately: grouped together, they are seen as words; as groups of words are formed into sentences, the reader makes assumptions about what should follow. The individual goes through the process of "mapping the text into a new structure ... that represents all aspects of the text the reader assumes to be essential" (Sengler, 1983, p. 93). The same type of structure seems to be formed when a program is read by the experienced programmer. Sengler calls it the "internal program" and the components of this internal program are the semantic units of the programming language. Sengler's description could be more accurately termed as an "abstracted" program, since he is talking more about extracting general principles from what is learned than about an internal process which may be difficult to measure.

If we account for individual differences, it cannot be assumed that each programmer ends up with exactly the same mapping of the internal program. Similar to differences in the reader's selection of salient points from the text, it is more likely that there would be a variety of similar mappings, possibly due to a different repertoire of programming experiences.

In order to understand a program, Sengler proposes five necessary abilities. These are the abilities to (a) find, (b) associate, (c) recall, (d) evaluate and (e) abstract.

Because most programs are long and complex, the programmer must separate them into portions and then link them together in order to be able to understand what effect has the whole. The programmer portions out sections of the program through its syntactical separations (e.g. blocks such as "begin...end") which are arranged into a hierarchy, thus simplifying the process. This hierarchy is what the programmer calls the "structure" of the program.

In order to portion out the program, the programmer first has to find the elements of the portion. These elements are its "components [the semantic units of the programming language] and relations [the semantic relations between those semantic units] as well as its inner portions [portions inside the currently analysed portion] and its outer relations [elements affecting the interface between portions]" (Sengler, 1983, p. 95).

After this is accomplished, the programmer can associate (or connect) the semantics of components and relations (from a knowledge of the language and the system).

Then the programmer can recall the previously understood and memorized semantics of inner portions and outer relations, and "evaluate the resulting semantics of the portion [which includes] an imagining of the effect of the portion on the program's state space" (Sengler, 1983, p. 95).

Because the semantics derived from this general evaluation are very complex and are difficult to process, the programmer has to develop a concept of the portion in order to be able to "fit" it into the overall image of the whole program, therefore abstracting the semantics. This whole process would be impossible if the programmer attempted to recall portions of the syntax only.

Learner characteristics and individual differences. Sengler (1983) purports that although every experienced programmer internalizes programming concepts, there are individual differences in how the internal program is mapped. Concurrently, van der Veer and van de Wolde (1983) studied individual differences in the aspects of control flow notations. They started from the premise (derived from their own research observations) that students who had avoided math as a subject were more likely to use false or distorted imagery to solve algorithms than those students who enjoyed and were good at math (they "created maximum semantic transparency in the code", p.109).

Van der Veer and van de Wolde determined four factors affecting the learning process:

a. learner characteristics,

b. characteristics of the problem solution,

c. features of the programming language,

d. didactics.

This study concentrated on learner characteristics only. Van der Veer and van de Wolde distinguished three sources of variability in the learning of computer programming.

1. general abilities (such as intelligence) which are considered to be fairly stable over time,

2. educational background, for example extensive training in math,

3. cognitive style, or an individual's typical problem-solving approach, that is, the way a person approaches a problem.

They stressed three important factors that enable the individual to acquire, store and retrieve information:

-factor I: tendency to memorize (rote memory). This factor reflects a skill rather than an ability.

-factor II: "operation learning". Tendency to derive "specific rules and procedural details" (p. 111).

-factor III: "comprehension learning". Tendency to derive "general rules and descriptions, and to record relations between different or even remote parts of the domain" (p. 111).

As we can see, these factors are congruent with Shneiderman and Mayer's model: operation and comprehension learning can be equated to acquiring semantics or conceptual levels, whereas only syntax can be acquired through rote memory.

Soloway, Ehrlich, Bonal and Greenspan (1982) parallelled the above models in their study of programming proficiency in novices. They had novice and intermediate programmers write three programs in PASCAL, each problem requiring a specific looping construct; these constructs were not a matter of

individual taste, but were required logically. Soloway et al. started with the premise that experts use a high-level plan knowledge (or again high-level concepts) to manage their programming structure. Soloway uses Minsky's definition of a plan as an "encoded frame". A frame represents "a template, which is customized to the particular features of the concept being represented" (p. 27). The plans are linked together, forming specific relationships, reflecting a hierarchy of ordered relations.

The basic questions about the problem are asked through the Strategic plans. The programmer will (a) describe what the program should do, (b) determine the variables, (c) decide what the program will do, (d) decide what is to be tested, (e) determine the setup, (f) determine the action in the body of the program and (g) decide on the conclusion. The code itself is written through the Implementation plans; the specific technique used (regardless of the code) is selected and developed during the Strategic plans.

Soloway et al. found that choosing the appropriate looping construct was not a predictor of success in writing the program, but choosing the appropriate looping strategy was. The looping strategy is a type of strategic plan whereas a looping construct specifies an implementation plan.

This means that the technique must be not only used but understood. In

other words, one cannot only use the syntax properly and be guaranteed a correct program. If the programmer knows why a certain technique is used and how it can be applied in different situations, there is discrimination between different concepts or, in this case, the strategic plans. The present research attempted to demonstrate that computer programmers use semantics more efficiently than non-programmers.

Algorithmic vs. Heuristic problem-solving stragegies

Throughout the studies reviewed so far, the heuristic/algorithmic problem-solving behavior model has been taking shape along with the syntactic/semantic model. Along with the semantic interpretation of concepts, the approach to problem solution has been addressed. But how is knowledge tapped into and used? The theory of hypothesis testing by Bruner er al. (1957) becomes the cornerstone of the algorithmic/heuristic model.

As discussed in a previous section, Bruner et al. (1957) found that the efficient problem-solver used the focusing strategies rather than the scanning strategies. Conservative focusing, where the individual changes one attribute at a time, can be equated to the use of an algorithm for problem-solving; the algorithm methodically considers every possible solution and always arrives at the right

answer. Focus gambling, where the individual changes several attributes at a time, can be equated to heuristic strategies; heuristics usually provide a nontrivial approximate solution in a reasonable amount of time.

Following up on Bruner et al.'s experiment, Wickens and Millward (1971) tested subjects on a concept-learning task, after giving them large amounts of practice. They found that as a rule "subjects tend to consider a small number of dimensions simultaneously, that a dimension paired inconsistently with the correct response is eliminated, and that when all the dimensions in a set are eliminated the subject samples a new set" (Mayer, 1983). This behavior is very similar to focus gambling, or heuristic problem-solving, and supports the belief that most people solve problems through heurisitics.

Computers work on algorithmic principles, and even "heuristic programming" is based on algorithms, such as the trial and error algorithm, where "the amount by which the current approximation fails to satisfy the problem is used to determine the next approximation" (Gear, 1978, p.6).

Several studies not only corroborate Shneiderman and Mayer's syntactic/semantic model of programmer behavior, but also go further in asserting that experienced programmers have acquired a specific cognitive "set" with which

they solve problems: Soloway et al. (1981) have found that the understanding and judicious use of programming principles (or semantics) increase program quality; Kahney (1983) developed a modal model of problem solving processes, where the programmer builds a "solution framework into which the elements of the problem are slotted"; Sengler (1983) described the building of an "internal program", a method of breaking down a problem into smaller pieces to make it easier to manipulate, understand and remember; and van de Veer and van de Wolde (1983) identified a specific cognitive style where operations learning (the derivation of rules and procedures) is one of the necessary abilities to the programmer.

From these studies, there is evidence that programmers operate in a manner consistent with Shneiderman and Mayer's syntactic/semantic model. This partial model of programmer behavior explains how knowledge is generalized or structured, but does not address its retrieval or utilization. The algorithmic/heuristic problem-solving behavior model provides some answers to the latter concern. Most of the studies discussed in this paper, however, have addressed these concepts in an isolated fashion, and several questions have remained unanswered. The next section discusses these problem areas, and elaborates the statement of the problem.

Statement of the problem

Among the literature on programmers abilities reviewed above, none of the studies have tried to determine the cognitive model of their subjects prior to their first acquisition of a programming language, or tried to compare heuristically versus algorithmically oriented individuals. In addition, no baseline has been obtained in order to compare the general population with programmers to determine if there is a difference in their cognitive functioning.

Comparing novice and experienced programmers is a good way to examine the structure of the cognitive processing. However, there has not been a consistent measure of differentiation between levels of ability. Kahney's discrimination between average and talented students is vague, and there are no details on how to measure these differences. Van der Veer and van de Wolde (1983) discriminate between the alphas and the betas, the alphas being poor at math and disliking it, and the betas being the converse. They determined the profile of a programmer through the three learner characteristic (abilities, education, and cognitive style), the emphasis being on educational background, intelligence and a score on the three learning style factors. Unfortunately, it is difficult to see how these scores

were integrated together to form one of three categories, alphas, betas, and "?" (a category where students were good at math but disliked it). Although these make intuitive sense, the measure of learner characteristics is still uncertain.

Most studies were done with people that already knew programming. Shneiderman's research determined that recall performance increased with experience, but no evidence was presented to verify if practice has an equal positive influence on every programmer or if it only enhances already present abilities. Studies should also be conducted using naive individuals (i.e. non-programmers) in order to be able to predict performance prior to any training in programming.

Another difficulty encountered in these studies is the determination of the programmer's level of expertise. There are no established rules as to what is a novice, an intermediate or an experienced programmer. Although deciding on this issue is beyond the scope of this paper, it is important to keep in mind that any reserach will be influenced by the definitions given to the level of experience of programmers. As with all other studies the definitions used to describe novice and experienced programmers for this research are purely subjective.

In addition to a lack of definition for levels of experience, the studies have

not used the same programming languages, thus complicating the judgement as to what exactly is an experienced programmer. There is also no indication as to how many languages a programmer knows or has worked with, and whether this factor influences the programmers expertise.

Factors Influencing Programming Aptitude. Several studies have been performed attempting to find factors influencing programming aptitude. Petersen and Howe (1979) found that college GPA and general intelligence explained less than 40% of the variance in a programming course grade. Kurtz (1980) "has found little correlation between major, class level, and previous college courses ... and overall course performance" (p.110). He found that measures of abstract reasoning ability were a good predictor of low and high achievers, but could not predict well the average score. Kurtz's test was based on Piaget's theory of intellectual development, and tests items were all verbal in nature, although they tested abstract reasoning.

Cheney (1980) examined the "relationship between cognitive style and student programming ability". He defined cognitive style as "the problem-solving methodology employed by an individual in a decision situation" (p. 285). Cheney distinguishes between two different types of cognitive styles: (1) analytic, where a structured approach is used to solve a problem, and (2) heuristic, where intuition

and arbitrary judgement is used to reach a decision. He found that students with an analytic cognitive style tended to score higher on programming tests. Cheney stresses that "cognitive style does not depend on the opportunity for education and thus is not biased in favor of those applicants with an educational advantage in math" (p.287). Cheney's definition of analytical cognitive style is very similar to the definition of algorithmic thinking used in this study.

Konvalina, Stephens and Wileman (1983) found that the best predictors of success in a programming course were high school performance (20% of variance explained) and high school mathematics (23% of variance explained). Except for Cheney's study, these results are highly unsatisfactory and demonstrate that such factors are minimally useful in predicting success in programming.

Tests of Programming Aptitude. Several tests attempting to measure aptitude for computer programming have been developed through the years. However, none has satisfactorily predicted success consistently. McNamara and Hughes (1961) found that the Programmer Aptitude Test (PAT) was a good measure of reasoning ability; the PAT is composed of three subtests: (1) Number Series, (2) Figure Analogies, and (3) Arithmetic Reasoning. Howell, Vincent, and Gay (1967) found, however, that the PAT was significantly correlated with education level, indicating a "corruption" of the measure. Wileman, Konvalina,

and Stephens (1981) found that mathematical reasoning ability was an important factor in successful programming. The PAT might then measure mathematical knowledge rather than programming aptitude.

The "Aptitude Assessment Battery: Programming" was developed in 1969. It purports to measure the abilities necessary for programming in business, such as "accuracy, deductive ability, reading comprehension of a complicated and extended explanation of a kind found in programming reference manuals, ability to grasp new and difficult concepts from a written explanation, and ability to reason with symbols" (Wolfe, 1969). The sample used for validation was not representative of the general population, since it came from a selected few companies. The test was developed in order to help companies select people in their own organisation for training in programming. Those who scored the highest on the test were professional engineers and professional mathematicians and those who scored the lowest were clerical and secretarial workers. This indicates that the test is highly dependent on education; it possibly measures factors such as a level of knowledge in math, general intelligence, or again a combination of factors, rather than programming aptitude by itself.

One of the best and most used test of programming ability is the Computer Programming Aptitude Battery (CPAB). The CPAB was developed in 1964 as a

selection tool for computer programmers and systems analysts, to be used by managers of data-processing and computer programming sections. Compared to other tests, The CPAB was assessed as being the only one of "sufficient quality for use in 'screening computer programmers for training" (Buros, 1978). Predictive validity for job performance is low. Cronbach (1970) also argues that the validity of the CPAB is questionable. He states that the sample used for validation may not be generalizable, that computer training may influence the test scores of the subjects, and that the rating criteria were affected by intervening variables such as age or experience. In addition, the test has not been revalidated since its inception in 1964.

Several reasons can be proposed for the deficiencies of these tests. First, it is difficult to develop accurate tests when the underlying cognitive processes have not been examined properly. Before one can develop an appropriate measure, it is necessary to understand the constructs on which these aptitudes are based. In the programmers case, the research done to date indicates that the programmer must know how to utilize information in order to solve the problem, before that problem is translated into a program. Nevertheless, there has been limited work done on individual differences between programmers and non-programmers in relation to their cognitive processes.

Second, the low predictive validity of most programming tests seems to be due to the fact that two concepts are measured at the same time: verbal comprehension (or syntactic/semantic ability) and reasoning ability (or heuristic/algorithmic ability). Although it can be argued that no test of reasoning ability can be completely devoid of verbal components, it would be useful to test each ability separately as much as possible in order to establish their separate as well as their combined influence. At this time, no test is available to measure these two processes independently. It was therefore necessary to select new measures in order to attempt to separate the verbal components from the reasoning components when the cognitive processes of computer programmers are examined.

In other words, it was necessary, using new tests, to determine if group differences existed, in order to establish a baseline for both non-programmers and programmers, and to verify if the tests selected had some predictive validity. Three main theorems were derived from these arguments.

First, it was believed that, although each cognitive set may not be mutually exclusive, one or the other (syntactic or semantic, and heuristic or algorithmic) is more prominent in a person's cognitive style. Second, it was assumed that, although practice may influence programming efficiency, the individual's cognitive set would have precedence. Hence, experienced programmers would solve problems

through algorithmic procedures rather than heuristics, but an individual whose main cognitive set is based on algorithmic problem-solving strategies may be a better computer programmer than the individual whose main cognitive set is based on heuristic problem-solving strategies. Third, an individual who uses semantic and algorithmic problem-solving strategies, rather than syntactic and heuristic strategies, would learn computer programming more easily.

Selection of measures

As discussed above, the existing measures of programming ability are either inadequate or defective. It was therefore necessary to select two tests which could replace these measures, by separating verbal and logical reasoning elements.

Measure of Semantic Ability. In order to measure whether programmers and future programmers have better control over semantic elements (i.e. can understand meaning better), Begg and Paivio's (1969) test of concreteness and imagery in sentence meaning was selected. (Appendix A presents the test sentences used in the experiment).

Begg and Paivio used this test to demonstrate that subjects noticed semantic changes better in concrete sentences, because it disturbed the image they had

formed when they internalized the sentence. This effect, however, was weaker with abstract sentences, which led them to believe that abstract text was more dependent on verbal components only.

In the context of this study, it was conjectured that their test could provide the possibility of differentiating between programmers and non-programmers on semantic aspects, by determining the difference between the two populations on semantic comprehension, and by determining the differences within the programmer population (due to experience, as in the Shneiderman and Mayer model). If a subject can determine whether a sentence is changed or identical, in both abstract and concrete aspects, then he/she has semantic control. This rationale leads to the first two hypotheses of this research:

1. Experienced programmers will have more correct answers on the semantic ability test than non-programmers.

2. Novice programmers who score high on the semantic ability test will have a high grade on a BASIC programming course, and those who score low on the test will have a low grade.

Measure of Problem-Solving Processes. In order to measure algorithmic

thinking, it was necessary to separate, as much as possible, non-verbal and verbal elements in the problem-solving process. Johnson-Laird (1977) describes a study by Whitfield (1951), who whas examining the effect of negative information on problem-solving. Johnson-Laird indicated that a "variant of this task is a recently available game known as 'Master Mind'" (p.172). The game of Master Mind, developed by parker Bros., is a color coded equipment in which the subject must "guess" a secret code of four different colors.

In his study on mediating processes during discrimination learning, Marvin Levine (1963) developed a formula to predict the minimum number of correct responses a subject could make to arrive at the right answer. This formula is based on the "Win-stay-Lose-shift" principle, where a subject stays with an attribute if he/she is told it was correct but changes attributes if told it was incorrect. With algorithmic thinking, since only one attribute is changed at a time, the answer should be arrived at in minimum number of trials. Therefore, Levine's formula can be used to calculate the minimum amount of answers a subject would have to make in order to get the right answer: if only one dimension is manipulated, but all others are presented, then each dimension is added together then divided by two:

$$(a+b+c+d)/2$$

In the case of Master Mind, two dimensions are manipulated, that is, color and position, among 6 colors (white, black, green, blue, red, yellow), 4 positions (1, 2, 3, 4) and 3 attributes (right color, right color and position, wrong color and position). Thus, we can calculate the average minimum of answers required:

$$(6+4+3)/3 = 4.3$$

The dimensions are divided by three, because the subject has one chance in three to be right every time. Therefore, it would take on average 4.3 answers for the subjects using a perfect algorithmic problem-solving strategy to arrive at the hidden code, while the subjects using heuristic problem-solving strategies would take more answers.

The game of Master Mind appeared to be a good test of the algorithmic problem-solving strategy, since the problem includes a minimum of verbal elements. Using this measure, two additional hypotheses can be formulated:

1. Experienced programmers will solve the hidden code in fewer answers than non-programmers.

2. Novice programmers who have a low score in the game will have a high grade on a BASIC programming course, and those who score high in the game will have a low grade.

Method

Subjects

Table 1 gives a description of the two major groups used in the study. Subjects were divided as follows:

(1) Experienced Programmers: this group of 31 subjects was composed of 12 males and 19 females who were between 19 and 35 years of age. Experience in programming was defined as having worked for at least one year in programming and/or having a working knowledge of three or more programming languages: five had worked as a programmer for less than one year, one for one year, four for two to three years and 22 for four or more years; two knew only one language, six knew two languages, and 24 knew three or more languages. All subjects had a Computer Science-related degree or diploma, and volunteered for the experiment. Twenty-three of the subjects came from two software firms in Halifax; the other eight subjects were programmers working at Saint Mary's University, in Halifax, Nova Scotia.

(2) General population group: this group of 44 subjects was composed of 28 females and 16 males who ranged from 19 years to over 35 years of age. Their

involvement with computers was minimal: 20 subjects had never used a computer, ten were strictly system users (e.g. for word processing), nine had done programming as an accessory to another task (e.g. writing a program for trend analysis), and 5 had done some programming as a major task, but had not actively programmed for more than six months. All subjects were of university level, ranging from first year to graduate school; they were all volunteers for the experiment.

Table 1
Description of Major Groups (N = 75)

Experienced Programmers

Demographic variables		% of sample	N
Gender	female	61.3	19
	male	38.7	12
Education	graduate	32.3	10
	undergraduate	29.0	9
	diploma	38.7	12
Age	19-24	22.6	7
	25-34	67.8	21
	35 or older	9.7	3

General Population

Demographic variables		% of sample	N
Gender	female	63.6	28
	male	36.4	16
Education	graduate	9.0	4
	undergraduate	90.1	40
	diploma	0.0	0
Age	19-24	77.3	34
	25-34	20.5	9
	35 or older	2.2	1

The General Population group was separated into three sub-groups (see

Table 2):

a. Control Group: The 23 subjects in this sub-group were students enrolled at Saint Mary's university, and were about to start an Introduction to Psychology course. There were 17 females and six males, between 19 and 25 years old. They were all volunteers, and agreed to be retested after their course. They received one credit for each testing session.

b. Novice Programmer group: The 11 subjects in this sub-group were students enrolled at Saint Mary's or Dalhousie universities, and were about to start their first computer programming course (in this case, with the programming language BASIC). There were eight males and three females, between 19 and 25 years old. None of the subjects had undergone a selection or programming aptitude test prior to their enrolment into the BASIC course. Subjects received $5 for their participation, and all voluntarily agreed to be retested after their course. None were in a computer science program.

c. Non-Programmers. Ten remaining subjects from the General Population Group were added to the Control Group (to make up a group of 33 subjects) and used in analyses against the Experienced Programmers Group. They were not given a treatment, thus were not retested. In addition, they were never used in isolation.

Table 2
Description of Sub-groups (N = 44)

Demographic variables		% of sample	N
Novice programmers			
Gender	female	27.3	3
	male	72.7	8
Education	Graduate	0.0	0
	Undergraduate	100.0	11
	Diploma	0.0	0
Age	19-24	81.8	9
	25-34	18.2	2
	35 or older	0.0	0
Control Group			
Gender	female	73.9	17
	male	26.1	6
Education	graduate	0.0	0
	undergraduate	100.0	23
	diploma	0.0	0
Age	19-24	78.3	18
	25-34	17.4	4
	35 or older	4.3	1
Non-Programmers			
Gender	female	80.0	8
	male	20.0	2
Education	graduate	40.0	4
	undergraduate	60.0	6
	diploma	0.0	0
Age	19-24	70.0	7
	25-34	30.0	3
	35 or older	0.0	0

Materials

Test of Semantic Ability. In order to test semantic ability, Begg and Paivio's (1969) test of concreteness and imagery in sentence meaning was selected. This test appeared to provide the possibility of differentiating between programmers and non-programmers on semantic aspects, by determining the difference between the two populations on semantic comprehension, and by determining the differences within the programmer population (due to experience, as in the Shneiderman and Mayer model).

Design and Procedure. Begg and Paivio's design and procedure was followed, for the most part, with some slight modifications.

Each subject heard 25 sets of sentences with five sentences in each set; the 25 sets were divided into sixteen test sets and nine filler sets. There were two types of sentences (concrete and abstract) and two types of changes (non-semantic change and semantic change). After each set of five sentences was presented, another sentence was given to the subjects. This sentence was one of the original five in the set, or one that was similar to it. Subjects judged whether the test sentence was either changed from, or identical to the original sentence. Table 3 presents a sample

test set and its repeated sentence.

Table 3
Sample Test Set for Semantic Ability Measure

(Concrete Non-Semantic Change, Changed Sentence)

a. The vicious hound chased a wild animal*
b. The tortured slave uttered a deafening shriek
c. The destructive army pillaged a prosperous village
d. The talkative admiral attended a costume party
Test: The vicious dog chased a wild animal

The test sentence was either the first or second sentence (to avoid recency effects) in the set. In the filler sets, the sentences to be played back for comparison were the third, fourth or fifth sentence in the set, selected in random fashion. The sentence that was played back was identical to the original in eight sets and changed from the original in eight sets. More explicitly, there were two changed and two identical concrete "non-semantic change" sentences, two changed and two identical concrete "semantic change" sentences; the design was the same for the abstract sentences. Sentences were not changed in the nine filler sets.

Once they heard each set of sentences, the subjects had 10 seconds to respond by marking the answer sheet, following each playback. The subjects were

given the following instructions:

"The purpose of this experiment is to find out how well people can remember what they have just listened to. The experiment will take about one half hour. You will hear on tape 25 sets of sentences, with five sentences in each set. At the end of each set of five, you will hear the word 'test', and then one sentence from the set will be repeated. Ten seconds will separate each set of sentences, giving you time to write your answer.

Sometimes the test sentence will be repeated with exactly the same words as the original. But sometimes it will be changed in some small way. In some instances, the meaning of the sentence will be changed. For example, if you had heard 'The whiskered priest entered an ornate temple', and then 'The bearded priest entered an ornate temple', this would be a change, but not in meaning, since whiskered and bearded mean the same thing. If you had heard 'The innocent occasion promoted a useless illusion', and then 'The useless occasion promoted an innocent illusion', this would be a change in meaning.

If the words are just as they were in the original sentence from the set, put a check mark beside "Identical" on your answer sheet. (The experimenter points to the sheet). If there is a change, but the meaning is the same, check "Changed-Same Meaning". If there is a change, but the meaning is different, check "Changed-Different Meaning". Listen normally to the sentences. They go too fast to memorize anything, and anyway, you must attend to the meaning fully, as well as to the words used. Remember to pay close attention to both the meaning of the sentences and the words used in the sentences. Any of the five sentences in a set may be the test sentence, and any words in that sentence may be changed. Do not take notes, just listen carefully. I will play two examples to help you familiarize yourself with the format. (The experimenter plays the two sets of sentences, and the subject attempts to find the answer.)

Are there any questions?"

To summarize, the subjects had to contend with five elements in the test: whether the sentences were concrete or abstract, and whether the test sentence was either identical, changed semantically or non-semantically. The following are examples of changes in test sentences:

1. Non-semantic change (Concrete): 'The whiskered priest entered an ornate temple', changed to 'The bearded priest entered an ornate temple'.

2. Semantic change (Abstract): 'The innocent occasion promoted a useless illusion', changed to 'The useless occasion promoted an innocent illusion'.

The original sentences were obtained from Dr. Begg, and are included in Appendix A, which presents the test sentences and their code. Ten filler sentences were constructed to complement the actual test sentences given by Dr. Begg.

Scoring Method: The scores obtained from both types of changes from both types of sentences were added together, giving one overall score of semantic ability. A perfect score was 16, since there were 16 test sets, corresponding to the number of "hits" or correct responses of semantic similarity or semantic change.

Master Mind Game. In order to measure algorithmic thinking, it was necessary to separate, as much as possible, non-verbal and verbal elements in the problem-solving process. The game of Master Mind appeared to have the potential

to meet this requirement. In the game of "Master Mind", one of the two players must "guess" a secret code of four different colors that has been selected by the other player. It is composed of:

1. a de-coding board, with ten rows of large holes (Code Peg holes) and ten groups of small holes (Key Peg holes), and four shielded holes (for the hidden code),

2. a shield, to hide the hidden code,

3. code pegs, round-headed, of six different colors (twelve each): white, black, blue, red, green, yellow,

4. key pegs, 40 thin flat-headed pegs (20 each black and white).

Appendix B shows the organisation of the de-coding board.

Design and Procedure. The experimenter explained the rationale behind the experiment to the subject, then explained the rules of the game, by giving the following instructions:

> "The purpose of this game is for you to duplicate the secret code of colored pegs behind this shield. Any combination of six colors (blue, yellow, white, black, green and red) can be used, although no color can be repeated twice in the secret code. You must duplicate the exact color and position of each colored peg.

To begin, you place any combination of pegs you want in the first row close to the shield. Each time you will place a row of code pegs, I will give you the following information beside that row by placing white or black key pegs for a hit, or nothing:

a. white key pegs mean that you have a right color but it is not in the right position;

b. black key pegs mean that you have a right color and a right position;

c. nothing means that you do not have the right color or right position.

For example, if the secret combination was blue, red, yellow and green, and you placed blue, black, green, and yellow, I would leave one whole empty (for the black peg), give you two white key pegs (for the misplaced yellow and green) and one black key peg. You can then use the colored pegs, which stay on the board and the feedback I gave you to place your second row of colored pegs, and so on.

Your answers will be timed, but there is no penalty for the time you take. Therefore, take as much time as you need to select your colored pegs after you have had feedback from me. The maximum number of rows you can fill is ten. If you have not found the solution by then, I will reveal to you the secret code.

You will attempt to duplicate four different secret codes. This should take approximately one half hour, but do not worry if you take longer.

Are there any questions?"

After the experimenter selected a combination of four colors for the hidden code (from a set of four combinations, used in a counterbalanced fashion) the subject was told to start "guessing" the right combination of colors. Immediately

after the subject had decided on four colors for a row, he/she was given feedback through white and black key pegs as outlined in the instructions.

The experimenter repeated this procedure until the subject had guessed the right combination of colors, and the experimenter revealed the hidden code. Subject's reaction time was timed between feedback periods, and the number of rows it took to arrive at the combination was recorded. The "game" was repeated four times for each subject, each time with a different combination of colors.

Scoring Method. Two dependent measures were obtained from the Master Mind Game:

(a) Number of Rows. The number of rows it took each subject to solve the code on each trial, and the mean number of rows over the four trials were recorded. The average scores were compared over groups to determine differences in overall performance; the scores for each trial were also used in a repeated measure analysis to determine if there was an interaction between experience level and/or practice. If practice played a role in solving the problem, then the subjects would take less rows to arrive at the solution at the end of the four games.

(b) Response Time. The mean response time per "game" (each trial time divided by the number of rows) and the average time it took to respond for each

subject (total time divided by four) was also measured. Repeated measures analyses were performed on time in order to determine whether the time taken to arrive at a solution changed with experience and/or practice.

Experimental Design

As was mentioned in the statement of the problem in the previous section, the aim of the present study was twofold: first, to determine whether experienced programmers would perform better than the non-programmers on the Master Mind game and the Semantic Ability test; second, to determine if those two tests could predict success on a BASIC programming course.

To achieve this, the experiment was divided into two sections: comparisons were made first between the General Population and the Experienced Programmers Groups, then between the Novice Programmers and the Control Groups.

Both the General Population and the Experienced Programmers were administered the Master Mind Game (MM) and the Semantic Ability Test (SEMAB). The Novice Programmers and Control sub-groups were retested after

treatment, which was either a course in BASIC programming language for the Novice Programmers, or in Introduction to Psychology for the Controls. Table 4 presents the design of the experiment.

Table 4
Experimental Design

Group	N	Test	Treatment	Retest
General Pop.	44	MM/SEMAB		
a. Novice	11		Crse in BASIC	MM/SEMAB
b. Control	23		Crse in PSYCH	MM/SEMAB
c. Non-progr.	10			
Exper. Progr.	31	MM/SEMAB		

The first part of the experiment compared the General Population to the Experienced Programmers to determine if their performance differed on the two measures; the Novice Programmer and Control sub-groups were then used to attempt to predict success in a Computer Programming course. The Novice Programmers were therefore tested after having taken a course in BASIC, and the Controls after a course in Psychology. In addition to test scores, academic grades

were obtained for the subjects course performance. The methodology also permitted a partial analysis of the validity and reliability of the two test measures. In addition to the Master Mind game and the Semantic ability test, a Background Information questionnaire (included at Appendix C) was filled out by all subjects; it consisted of seventeen questions on demographics, education and computer experience.

Coding. There were three dependent variables in this experiment: Semantic Ability Scores, Master Mind Game Scores, and Course Grade. Semantic Ability test scores ranged from 1 to 16; in increments of one integer, 16 being the maximum number of correct answers possible. Master Mind game test scores ranged from 1 to 10, for the number of rows it took to solve the problem, 10 being the worst score. Test scores were an average of the sum of the number of rows per trial for four trials, for a maximum of 10. Grades were coded from 1 to 8, 1 being equivalent to an "A", 8 to an "F". Thus, grade and Master Mind scores went in the same direction, and Semantic Ability scores ran contrary to grade. Grades from the two courses were considered equivalent.

Results

This results section is divided into three parts. The first analysis compares Experienced Programmers and General Population groups and examines the response patterns of the major groups on the two performance measures. The second part examines the two measures for their predictive potential as measures of computer programming ability. The third analysis evaluates the psychometric properties of the tests (reliability and validity).

Comparison of Major Groups

Based on an analysis of demographics, the General Population and Experienced Programmers groups were considered acceptable samples. The number of female and male subjects in both groups was essentially the same (X^2 = .042, p. > 25) although the Experienced Programmers group was generally older than the General Population Group (X^2 = 19.3, p. < 001); concurrently, there were more subjects having completed a degree (i.e. holding a diploma or attending graduate school) in the Experienced Programmers Group (X^2 = 25.91, p. < 001). However, because of the nature of the experiment, where experience was a differentiating factor, it was decided that the differences were inherent to the groups studied. (Results were tested for gender differences which were all non significant.)

Test means shown in Table 5 indicate that the Experienced Programmers performed better on both tests than the General Population, as the first two hypotheses predicted. SEMAB results for the Experienced Programmers group were not significantly different from those of the General Population ($t(73) = -1.10$, $p. > 16$).

Table 5
Test Means and Standard Deviations for the Experienced Programmers and General Population Groups

	N	Mean	SD
Experienced Programmers	31		
MM		5.168	.862
SEMAB		7.855	2.241
General Population	44		
MM		5.743	1.110
SEMAB		7.330	1.930

Master Mind Game

ANOVAs with repeated measures were used to compare the performance of the General Population and the Experienced Programmers. (All ANOVA tables are included in Appendix D). Separate analyses were performed for the number of rows per trial and for the average time per trial to solve the color code problem. These analyses revealed that although there was a significant difference between their performance ($F(1,74) = 9.42, p < .003$), both groups performed similarly from one trial to another in the number of rows they took to solve the problem ($F(3,219) = .06, p > .98$), and their performance remained the same between trials, regardless of experience ($F(3,219) = .82, p > .49$). Notwithstanding these results, MM mean scores were significantly different from the projected 4.3 discussed in the literature (Experienced Programmers: $t(30) = 5.5, p. < 01$, General Population: $t(43) = 8.52, p. < 01$). Figure 1 shows the average number of rows needed for the solution for each trial. The average time per row per trial was the same for each group ($F(1,74) = .90, p > .34$). The average time per row per trial decreased over successive trials ($F(3,219) = 2.53, p. < 06$); However, the decrease in average time per row per trial was not the same for each group. The Experienced Programmers remained more constant across trials, while the General Population decreased ($F(3,219) = 2.63, p. < 05$). Figure 2 represents the average time per row per trial needed for the solution.

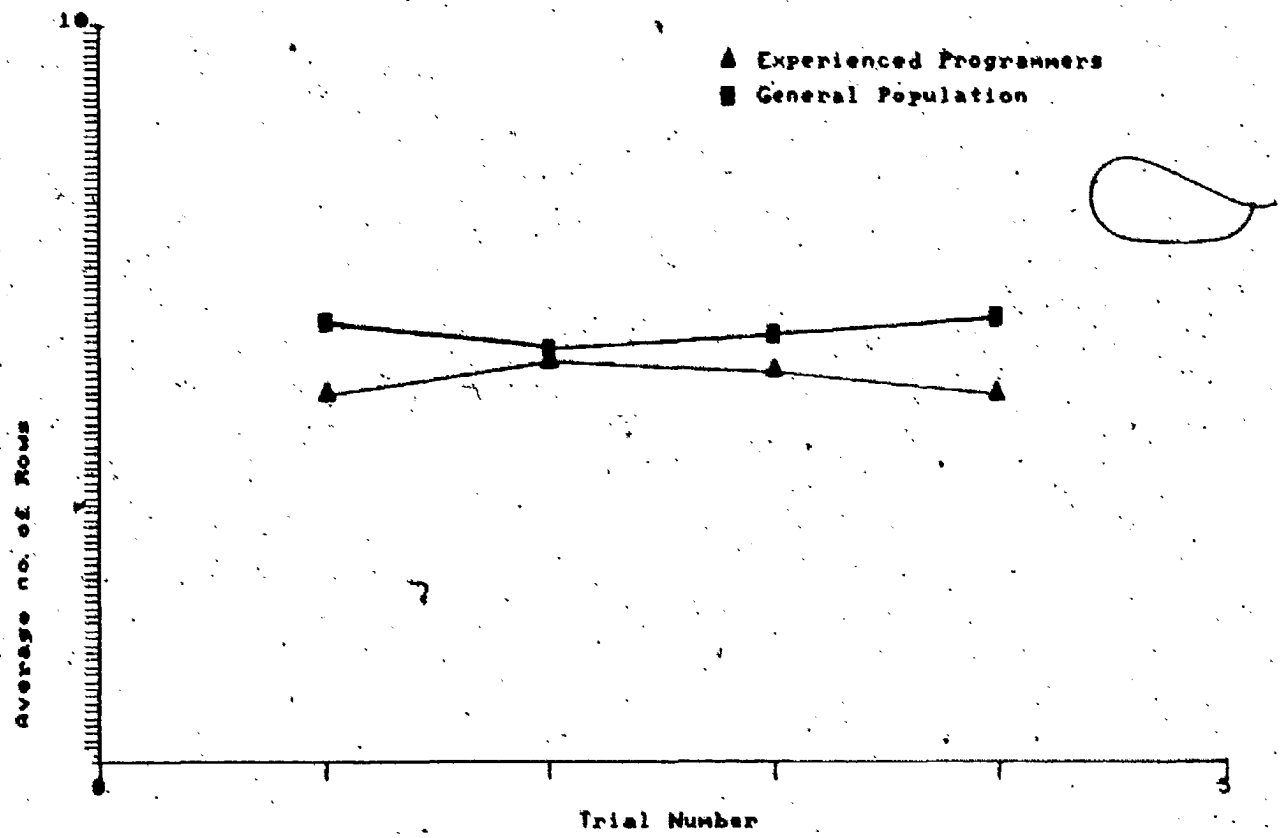

Figure 1. Average Number of Rows Per Trial needed for MM Solution for Experienced Programmers and General Population Groups.

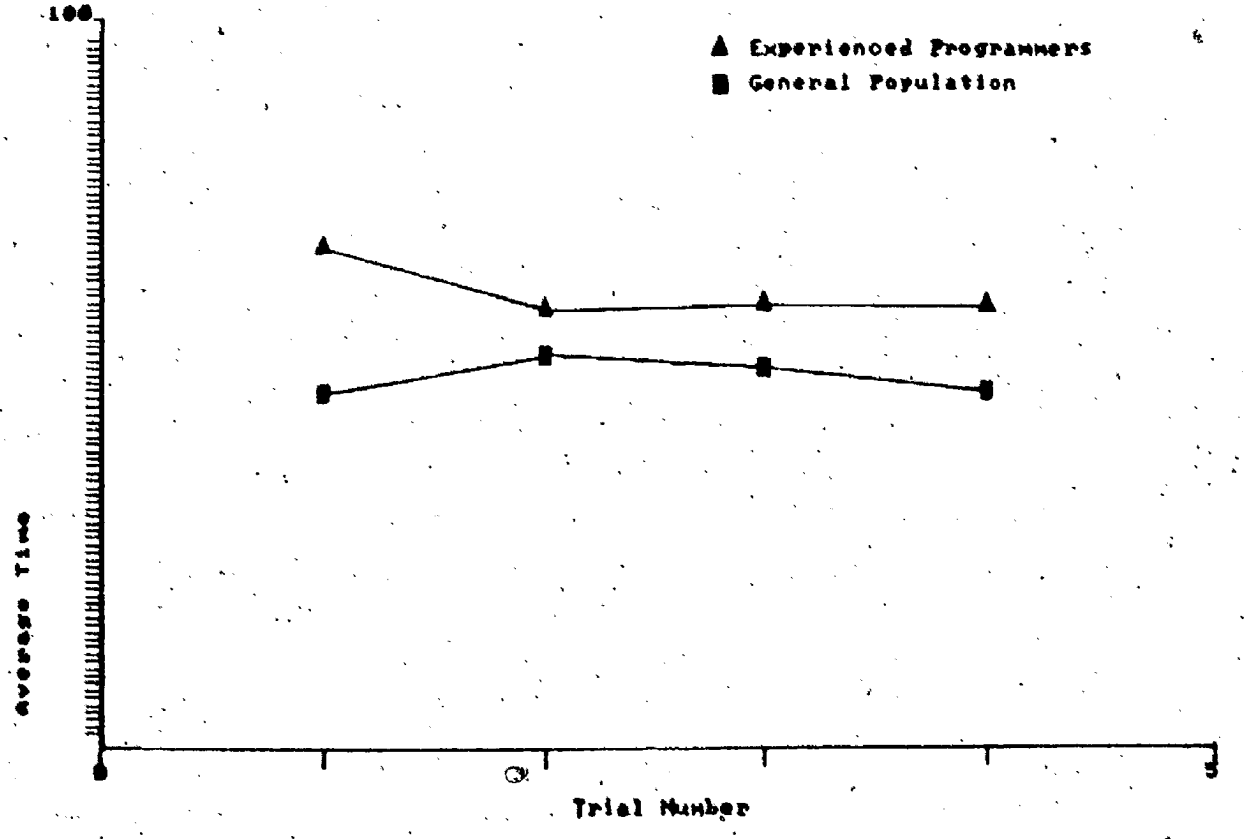

Figure 2. Average Time per Row per Trial for MM
Solution for Experienced Programmers and General Population

Semantic Ability Test

Answering patterns on the Semantic ability test were examined. Crosstabulations on every question for the entire group (n = 74) were performed to determine the most difficult questions in the questionnaire, and to establish whether answering patterns existed notwithstanding the lack of significant difference between the groups. Seven out of the 16 test sentences were missed by over 60% of the whole group. Five were from identical sets and one was a semantic change. Three were concrete and four were abstract. Only two sentences were answered correctly by over 60% of the group. Both sentences were concrete semantic changes. A summary of these results is shown in Table 6.

Table 6
Semantic Ability Test--Answering Pattern (Filler Sentences Excluded) in Percentages (n = 74)

Question no.	Type	Miss	Hit
1	Abstract non-semantic change	55	45
2	Concrete identical	65*	35
4	Concrete semantic change	19	81*
6	Abstract non-semantic change	45	55
7	Abstract identical	66*	34
8	Abstract semantic change	61*	39
10	Concrete non-semantic change	53	47
13	Concrete identical	61*	39
16	Abstract identical	74*	26
17	Abstract identical	55	45
19	Concrete semantic change	24	76*
20	Concrete identical	69*	31
21	Concrete non-semantic change	42	58
23	Abstract semantic change	41	59
24	Concrete identical	56	44
25	Abstract identical	66*	34

* indicates percentage of answers 60% or higher

In addition, differences in success of responses (i.e. how many hits) were examined by group. The Experienced Programmers and General Population groups differed significantly on three questions; each involved a concrete test sentence. In two cases, the Experienced Programmers performed better. Table 7 describes the findings for these groups.

Table 7
Difference in Answering Patterns--Experienced Programmers vs. General Population

Question no.	Type	t-test
4	Concrete semantic change	-1.82*
19	Concrete semantic change	2.59***
21	Concrete non-semantic change	2.22**

* .05 level, ** .01 level, *** .001 level of significance

The response patterns were also explored for the Novice Programmers and the Control Groups. There were three sentences on which these two groups differed significantly. Two abstract identical sentences were answered better by the Control group, and one concrete non-semantic change was answered correctly more often by the Novice programmers. Table 8 describes the findings for this group.

Table 8
Difference in Answering Patterns--Novice Programmers vs. Control Groups

Question no.	Type	t-test
16	Abstract identical	1.97*
17	Abstract identical	2.87**
21	Concrete non-semantic change	-4.11***

* .05 level, ** .01 level, *** .001 level of significance

Predictive Potential of Measures

The second part of the study attempted to show that a grade in a computer programming course could be predicted by the MM and SEMAB tests. The two sub-groups from the General Population, the Novice Programmers and the Control Groups, were used for these specific analyses.

Table 9 represents test means and standard deviations for test (MM1, SEMAB1) and retest (MM2, SEMAB2).

Table 9
Test Means and Standard Deviations for the Novice Programmers and Control Groups

	N	Mean	SD
Novice Programmers	11		
MM1		5.705	1.224
MM2		5.068	.936
SEMAB1		7.545	2.115
SEMAB2		7.091	.831
GRADE (GPA)		5.273	1.737
Control Group	23		
MM1		5.717	.945
MM2		5.728	.862
SEMAB1		7.000	2.000
SEMAB2		7.957	1.522
GRADE (GPA)		3.7826	1.6776

ANOVAs with repeated measures were used to compare the performance of the Novice Programmers and Control Group. Separate univariate analyses were carried out for the MM and SEMAB data. These analyses revealed that the Novice Programmers performed at a significantly better level on the MM test than the Control Group ($F(1,33) = 1548.9$, $p < .001$). With regard to SEMAB data, the Control group performed at a higher level, on average ($F(1,33) = 917.87$, $p < .001$); however, in this case, while both groups started at about the same level, the control group improved while the Novice programmers did not ($F(1,33) = 3.96$, $p < .03$). For both interactions, comparisons showed that the differences between groups on MM1 and SEMAB1 were not significant ($t(32) = .03$, $p > .48$, $t(32) = -.73$, $p > .23$), but were on MM2 and SEMAB2 ($t(32) = 2.03$, $p < .025$, $t(32) = 1.76$, $p < .04$).

Master Mind. A significant relationship was found between the second MM score and the final course grade for the Novice Programmers ($r = .71$, $p < .01$). Novice Programmers who performed well on MM2 also did well in the BASIC programming course. No other relationships, including those taken at the beginning of the course, were significant. Similarly, this correlation was the only one whose 95% confidenct interval did not include zero.

Inspection of Table 10 also shows that the magnitude of the correlations for the Novice Programmers Group were greater than those for the Control Group, reaching an acceptable level, $r = .30$, in the case of MM1.

Table 10
Correlations Between Grade and Measures
and 95% Confidence Intervals

Groups	Measures			
	Master Mind		Semantic Ability	
	pre	post	pre	post
Novice Progr.	r = .30 (-.36,.77)#	r = .71* (.19,.92)	r = -.26 (-.74,.40)	r = -.23 (-.42,.72)
Control	r = .20 (-.24,.55)	r = -.13 (-.51,.29)	r = -.11 (-.50,.31)	r = -.09 (-.48,.33)

* indicates significance at the .05 level
\# intervals are separated by a comma

Fisher z' transformations (Table 11) were used to compare the difference between the Novice and Control groups for each correlation. The only significant difference between correlations for the two groups was found for MM2 scores.

Table 11
Significance of Differences Between Correlations of Two Groups

Groups	Measures			
	Master Mind	Semantic Ability		
		pre	post	pre	post
		Fisher z'.			
Novice programmers with Control		.61	.97*	.64	.63

* indicates significance at the .05 level

Further analysis revealed that the Novice Programmers improved their performance from $\overline{X} = 5.7$ to $\overline{X} = 5.1$, on the second Master Mind task ($t(10) = 1.89, p < .04$), but that the Control group did not improve ($t(22) = -.06, p > .48$). The scores for both groups at the beginning of the study on MM1 were approximately the same. This interaction can be seen in Table 9.

Semantic Ability. Although none of the correlation coefficients were significant, Table 10 shows that the relationship between grade and SEMAB was stronger for the Novice Programmers ($r = -.23$) than for the Control group ($r = -.09$). In comparing pre- and post-test differences, the Control Group's SEMAB scores improved ($t(22) = -2.33, p < .01$), while those for the Novice Programmers Group did not ($t(10) = .81, p < .21$).

A stepwise multiple regression analysis was performed using MM1, SEMAB1, age, performance in high school, performance in university, number of math courses in high school, and number of math courses at university as predictors of course performance. None of the variables entered in the equation predicted grade in either the Novice Programmers or the Control groups.

Psychometric Properties of Measures

This section of the study attempted to assess whether the two tests do in fact measure different constructs (a partial discriminant validation) and whether they are reliable. Some of these analyses have been performed on data obtained from all the subjects used in this study, that is, all of the groups from the two experiments ($n = 75$).

General Population. There were no gender differences in MM performance ($t(31) = -1.03, p > .31$) or in SEMAB results ($t(31) = .44, p > .67$) for the General Population group. However, those who were in graduate school ($n = 4$) scored better on MM ($t(31) = -2.52, p > .04$) but not on SEMAB ($t(31) = 2.48, p > .07$).

Experienced Programmers. Although males were significantly older than females ($X^2 = 9.32, p < .05$), both genders performed equally well on MM ($t(29) = -1.19, p > .24$) and on SEMAB ($t(29) = -.86, p > .43$). Subjects without a degree or with a

diploma in computer programming (n = 12) performed as well as subjects holding a university degree (n = 19) on both MM ($t(29) = -.74$, $p > .47$) and SEMAB ($t(29) = .64$, $p > .53$)

There was no significant relationship between MM1 and SEMAB1 ($r = .115$, $p > .20$) over all the subjects or for each of the Experienced Programmers and General Population groups considered separately (Experienced programmers: $r = .072$, $p > .35$; General Population: $r = -.257$, $p > .07$). In addition, there was no relationship between the MM2 and SEMAB2 scores obtained for the Control Group ($r = 0.1984$, $p > .182$) for Controls, or for the Novice Programmers group ($r = .0876$, $p > .40$).

The reliability coefficient, $r = .49$, between pre- and post-test (n = 35) on MM was significant ($p < .001$); as was that, $r = .36$, on the SEMAB test ($p < .018$). When the data was analysed separately for Controls and Novice Programmers, the reliability of both tests remained significant for the Control group (Master Mind: $r = .487$, $p < .009$; Semantic ability: $r = .403$, $p < .03$). Although the magnitude of the reliability coefficients for both tests for the Novice Programmers remained constant, the significance of the coefficients was only marginal ($p > .06$ and $p < .07$ respectively).

Discussion

Comparison of Major Groups

The first part of the experiment attempted to show that Experienced Programmers would perform better than the General Population on the MM and SEMAB tests. This relationship was found for MM but not for SEMAB. Although the Experienced Programmers solved the problem in more rows than the minimum number 4.3 that was needed, their overall score on MM was higher than the General Population's score. However, the Experienced Programmers took longer on average to solve the problem than the General Population. These results confirm Pellegrino's argument (1985) about individuals using algorithmic thinking to solve a problem: "...the best reasoners are often slower at encoding...slower, more accurate encoding of information at the outset speeds up subsequent processes" (p.52). The absence of variation between the average number of rows per trial and between the average time per row per trial indicates that the test was not affected by intervening variables. If the number of rows had decreased, or if it had taken less time per row for the subjects to solve the problem from one trial to the next, practice would have influenced the results. If the number of rows per trial had fluctuated, or if the time per row had increased, results could have been influenced by fatigue or boredom.

However, scores from one trial to the next were similar enough to show that the test may not be sensitive to external or internal factors.

On the other hand, Semantic Ability test results were inconclusive; both major groups performed similarly.

The second part attempted to determine if a grade in a computer programming course could be predicted by the MM and SEMAB tests. Although some of the results were encouraging, the relationship could not be fully established.

Master Mind. Novice Programmers performed better on MM than the Controls, although it is their improvement on the second testing of MM (MM2) which determined their better performance. This seems to indicate that the treatment given to the Novice Programmers (the BASIC programming language course), may have had an effect on MM scores. It could also be conjectured that the Novice Programmers used the problem-solving strategies acquired through programming in BASIC and transferred them to the Master Mind problem. The interaction supports this argument by showing a significant treatment effect for the Novice Programmers.

Therefore, although the MM1 score did not predict grade, a definite relationship between the MM score and the type of course was established. It

appears that learning how to program could have helped the Novice Programmers to solve a logical reasoning problem such as Master Mind. The instruction in Introduction to Psychology apparently had no effect on MM performance. These results are in agreement with Mayer, Dyck and Vilberg (1986), who found that "there is an important...relationship between a person's thinking skills and ability to learn Basic" (p.610).

Semantic Ability. Test results indicate that although the Control group's MM scores stayed the same, their SEMAB scores improved. It could be argued that Introduction to Psychology leads to improvement in verbal skills, while a course in BASIC does not. However, this interaction could be an artifact due to the nature of the test.

Psychometric Properties of Measures

Discriminant validity. The MM game and SEMAB tests show good discriminant validity: data analysis seems to indicate that two different processes are tested, since the correlation coefficients between the two tests at either testing session for the four groups are all below 30 percent and are not significant.

Test-retest Reliability. Both measures have good test-retest reliability: 49 per cent for MM and and between 36 and 48 per cent for SEMAB. These reliability

measures are applicable to all groups. These results are encouraging, because although treatment may have caused an increase in results, this increase was similar for each individual in the group, rather than being haphazard. Thus, it supports the discriminant validity data by suggesting that a particular process is measured through each test, although in the case of the SEMAB test these results are more doubtful.

Semantic Ability Test.

The results obtained in this study contradict Begg and Paivio's results who had found that subjects have more difficulty detecting semantic changes in abstract sentences than in concrete sentences. These findings were not duplicated: subjects were equally successful with both types of sentences, although they generally all performed poorly. The results also seem to indicate that this test is difficult (group $X = 7.5$), less sensitive to change, and less able to differentiate between individuals than the Master Mind game.

Three main arguments can be proposed for the results obtained with this test. First, the test may not be able to discriminate between levels of ability because the subjects used for the study "may essentially represent the upper ranges..." of language ability (Dillon, p.91), and they may have already mastered the processes implied in the differentiation of levels of language comprehension. Dillon states:

> Those whose only or main problem in elementary school was decoding are either not in the college population or they have mastered decoding-related processes to an extent that the limiting performance factor lies elsewhere...in a sense, their reading ability matches their general intellectual ability and within the latter, there is a fairly narrow range--above average to superior...Studies of adult readers have seldom used sufficiently difficult decoding tasks. The sensitivity of the tasks thus are in question (p.87). Considering the above, the Semantic Ability Test may not be sensitive enough to tap into distinct abilities.

A second argument could be that the test is a good test, and that the Control Group's language ability did in fact improve; it may be that by learning psychological concepts, the subjects reinforced their ability to differentiate concrete sentences from abstract sentences. Clark (1974) states that "concepts are organized into semantic fields that have a "conceptual core". A conceptual core is "an abstract entity that in reflecting a deeper conceptualisation of the world integrates the different concepts within the semantic fields" (p.183). The subjects may therefore have acquired experience in using semantics to understand concepts, thus being able to transfer this experience to the SEMAB test. This argument, however, does not explain why the Novice Programmers group did not improve, since they were also learning concepts, albeit of another nature.

A third argument seems to explain somewhat better the results that were obtained. In this study, Begg and Paivio's test was adapted to the purpose of this

particular research. Similar instructions as in Begg and Paivio's experiment were used, but instead of being asked if the sentences had been changed in terms of words or of meaning, the subjects were asked to indicate if the sentence was changed from the original. If the subjects answered yes, then they were required to indicate if it was a change in meaning. Begg and Paivio also asked their subjects to recognize lexical changes. In this case, however, recognizing a change as non-semantic was considered identical as recognizing semantical similarity through the change. Therefore, recognizing that difference was also considered a test of semantic recognition and ability.

The format of the test was also modified slightly. In Begg and Paivio's experiment, subjects were tested separately on concrete and abstract sentences; that is, one group listened to changes in concrete sentences only, and another group listened to changes in abstract sentences only. In this experiment, the two types of sentence sets were combined and mixed in a random fashion.

By modifying the test in this fashion, the results were not duplicated in either testing session; they also seemed to be affected by the type of treatment applied to the subjects. These results are confusing, since it seems difficult to imagine that language ability would significantly improve in a short time, especially at the age level of the subjects. A more plausible explanation could be that the Semantic Ability Test suffers from important procedural limitations, and that the apparent

effects are due simply to the manipulation of the test, that is, the combination of concrete and abstract set of sentences into one test. The test then becomes unreliable in testing the syntactic/semantic behavior of subjects, although to determine whether it still can test concrete imagery would be the subject of another study

Overall Discussion

Only one of the four hypotheses has been confirmed through this study; however, the results obtained are considered encouraging.

Implications of Results

Part of Shneiderman and Mayer's results have been confirmed: Experienced Programmers did perform better than Novices, albeit not in their ability to recognize semantic changes. It may be that semantic elements -and their comprehension- have less importance than their hierarchical organisation. The results raise the possibility that, as programmers gain experience, their capacity to reorganize programming structures also increases. Consequently, experienced programmers may be better able to recognize the need for an algorithm in solving problems. (In fact, it may be interesting to research whether experienced programmers approach

all problems in the same manner, or whether they choose an algorithmic strategy only for specific problems.)

Semantic Ability

Begg and Paivio's (1969) test of concreteness and imagery in sentence meaning did not provide an adequate measure of verbal ability. This problem does not negate the necessity to find or develop a test which measures this ability (to which we intuitively can agree), perhaps through the use of complex processes for which Perfetti (1983) has argued. The complex processes would tap into the different levels of language ability and confirm Shneiderman and Mayer's hierarchical organisation from low level constructs to a higher level domain. Abstracts constructs vary in difficulty, and although most people can understand some of them, others are more subtle and intricate to grasp: accordingly, individuals with superior verbal ability would be able to better manipulate and understand the more complex concepts.

This research into complex processes raises another question: Can a measure of verbal ability be separated from problem-solving processes? Once language becomes more complex, its comprehension may always involve a form of problem-solving, with the information gathered and integrated into the "working memory" (Feigenbaum, 1970). The quality of the integration may be what should be

measured; it may be an elemental process, underlying both verbal and problem-solving abilities. In other words, a problem may not be understable without the integration and redefinition of its semantics. The hierarchical organisation of semantics may mean that existing verbal ability tests tap the low level details, but never touch the high level concepts which require a more abstract understanding. This could mean that the integration of the semantics is more important for language manipulation than the imagery or concreteness aspects of language.

Throughout the literature review, it was stressed that understanding (and in this case integration of information) is crucial to the cognitive process. When the argument of hierarchical organisation is taken into account, results on SEMAB could be interpreted two additional ways: either the abstract level was low enough that it did not need to be understood to be remembered, or the sentences were easy to understand and were measuring a low level of integration, a level which would have been attained by all subjects in the study simply because of the pre-selection (university) they had gone through (Dillon, 1983). Either way, it is difficult to reject the hypotheses formulated on language ability on the basis of this test, although it is also impossible to confirm them.

In short, it cannot be established whether the Semantic Ability Test is an accurate measure, whether it is too sensitive, or whether it is not discriminative

enough due to the language sophistication of the subjects. The results are intriguing enough, however, to warrant further examination.

Algorithmic Thinking

The integration of information has not been tested adequately, but its retrieval and utilisation was measured fairly well through MM. Another research question raised by MM results stems from observations made by the experimenter while subjects were solving the color code.

Bruner et al.'s findings that most people use focus gambling (or heuristics) to solve a problem were confirmed. Most subjects had no discernible method, and were taking chances in changing several attributes at a time, sometimes without using the feedback given by the experimenter. This resulted often in repeating wrong color combinations, although, in general subjects succeeded in "guessing" the code.

Those who used a method used an approach closer to conservative focusing. However, although they did use some type of method, they did so with various degrees of efficiency. For example, instead of starting with four different colors (which is the best way to start), some subjects would start with four pegs of the same color, or two pegs of one color and two pegs of another color. By process of

elimination, they would know which colors were not included in the code; this did not eliminate position at the same time as color, however. These procedures work, and one can arrive at the solution more easily than with no method, but they are costly in terms of rows; they seem to be non-optimal algorithms combined with heuristic thinking, because they are more dependent on chance for arriving at the solution in the minimum number of rows (4.3). They used the feedback given to them to change attributes methodically, and generally took less rows to solve the problem. From a qualitative and observational point of view, it was clear that certain subjects attempted to find the secret code in a methodical -even if imperfect- way, whereas others just tried out some combinations until they "happened" on the right one. The use of a method, rather than guessing, would also explain why the experienced programmers took longer on average to solve the problem: the method in itself is time-consuming but more accurate, which is the way they may have learned to work in order to write computer programs.

The Master Mind Game seems to measure a process related to the problem-solving strategies required to program: the Novice Programmers did better on that test after learning how to program in BASIC, and the Experienced Programmers did better than the General Population. Master Mind is reliable; however, whether it can predict success in a programming course has not been possible to establish.

These observations raise other questions worthy of study: Can the use of any algorithm to solve a problem determine an ability in computer programming, or must the subject use the optimal algorithm? Is the use of non-optimal vs. optimal algorithms an indication of levels of programming ability? If the first question can be answered by stating that the use of any algorithm can predict computer programming ability, then Master Mind is not a good test, since it does not differentiate between the types of algorithms used. However, an observation on how people play the game may give indications as to the types of algorithms the subjects use. This could also be obtained by having the subjects think aloud while thay are solving the problem.

One way of alleviating these problems would be to raise the difficulty level of the task. By having a six-color instead of a four-color code (with ten colors to chose from), the problem becomes more complex, in the same way the Towers of Hanoi problem becomes more difficult by the addition of rings. Glass (1979) indicates that the more complex the problem, the more difficult it is to solve because of the "backtracking" (that is, the algorithm requiring an apparent reversal of the steps going towards solving the problem) necessary to arrive at the solution.

Using a more difficult problem may better separate the heuristically- vs. algorithmically- oriented individuals by removing the element of chance. Nevertheless, an important advantage to the MM game as a measure of algorithmic

process is that it takes only approximately 30 minutes to administer (for 4 trials). In addition, it has proven to be resistant to practice effects; it is also an interesting test for the subject because it gives immediate feedback, and is fun to play.

Finally, although results have been somewhat inconclusive, the Master Mind game is considered to be a worthwile tool for measuring algorithmic problem-solving processes, and merits further study.

Limitations of Study

The major limitation of this study was the small number of subjects obtained for the various groups, and especially for the Novice Programmers group. Because of this, it was not possible to determine if the tests selected could predict computer programming. It was also impossible to determine if practice had an equal influence on every subject in either group, or if it only enhanced already present abilities, because there were too few people to be able to perform solid comparisons. Since there seems to be a connection between programming and MM, further study should be conducted with larger groups of Novice Programmers to attempt to determine if MM can predict programming ability. In this study, it was felt that the groups were too small to obtain predictive data on either test.

A second difficulty was the lack of control over what was being taught in the

BASIC course, for example, which principles and concepts, or which problem-solving procedures or techniques were given in order to transfer a problem into code. Because the course was not tightly controlled (and the same can be said for the Introduction to Psychology course), it is difficult to replicate results based on similar treatments. It cannot be taken for granted that all BASIC or Psychology courses are the same. However, all studies that were reviewed for the purpose of this study and who used a programming course as treatment have this same deficiency.

For the purpose of further research, the content of a programming course should be controlled in order to establish what concepts are taught, and to determine if teaching these concepts improve algorithmic thinking. If this were the case, the implications for teaching computer programming are great, since it could be argued that if the proper problem-solving strategies and concepts were taught, then programming could be learned by anybody who has the intellectual capacity to understand these concepts.

A third limitation was the lack of commonality between the treatment for Novice Programmers and the treatment for the Controls. Treatment for the controls should have been a computer-related but non-programming course, such as learning how to use a word-processing program. Tying the Control treatment to a computer-related task would eliminate the intervening variables of learning how to

use a computer system.

Another deficiency which has been experienced throughout the literature as well as in this study is the lack of more solid definitions for the levels of experience for programmer, which make it difficult to classify subjects as well as replicate studies. The definitions given to the experience levels of the subjects in this study may have influenced, since the differences between experience levels were sometimes obscure or very slim.

The modifications made to Begg and Paivio's test of concreteness and imagery in sentence meaning have had drastic effects on the results expected. It is difficult to determine at this point whether the data obtained were an accurate reflection of the groups and the effects of the treatment, or whether they are due solely to the changes in the test. Before this measure is used again, an attempt to replicate Begg and Paivio's results with the original test should be made. The two test versions (one abstract and one concrete) should then be mixed together: similar results should be obtained, if the tests are not sensitive to procedural manipulation. These studies would establish a baseline and give credence --or infirm-- the hypothesis that semantic ability, and this test in particular, can help predict programming ability. In fact, a preliminary study should have examined this point before the test was used in its present form.

Conclusion

The results of this study have shown that the Master Mind game differentiates Experienced Programmers from the General Population, and that it seems to measure algorithmic thinking, albeit imperfectly. It is therefore considered a promising measure, that should be used in further studies on the cognitive model of computer programmers.

The Semantic Ability test, on the other hand, has provided doubtful results. It is felt that the hypotheses based on that test have not been disproved, and that a serious study on the validity of the test must be performed.

Although the study has limitations in several respects, it is felt that it served to reinforce the need to explore the processes underlying the computer programmers abilities to program. If these processes are essential, then the need for solid, reliable tests of programming abilities is imperative. But if these processes can be taught, the measure of computer programming ability becomes redundant. The implications are great and far reaching, and the consequences would affect a great number of domains.

References

Aaronson, D. and Scarborough, H.S. (1976). Performance theories for sentence coding: some quantitative evidence. Journal of Experimental Psychology: Human Perception and Performance, 2, 56-70

Adelson, Beth (1981). Probleming and the development of abstract categories in programming languages. Memory and Cognition, 9(4), 422-433.

Allen, Robert B. (1982). Cognitive actors in human interaction with computers. In Badre, Albert, and Shneiderman, Ben (Eds.). Directions in human/computer interaction. (pp.1-24) Norwood, New Jersey: Ablex Publishing Corporation.

Badre, Albert and Shneiderman, Ben (Eds.) (1982). Directions in Human/Computer Interaction. Norwood, NJ: Ablex Publ. Corp.

Begg, Ian and Paivio, Allan (1969). Concreteness and imagery in sentence meaning. Journal of Verbal Learning and Verbal Behavior, 8, 821-827.

Begg, I. and Wickelgren, W.A. (1974). Retention functions for syntactic and lexical versus semantic information in recognition memory. Memory and Cognition, 2, 353-359.

Benbasat, Isak, Dexter, Albert S. and Masulis, Paul S. (1981, November). An experimental study of the human/computer interface. Communications of the ACM, 24, 752-762.

Bever, T.G. (1970). The cognitive basis for linguistic structures. In J.R. Hayes (Ed.). Cognition and the Development of language, New York, Wiley.

Bourne, L.E. Jr. (1975) Human Conceptual Behavior, Boston, Mass.: Allyn and Baron, 175, 213, 513.

Bresnan, J. (1981). An approach to universal grammar and the mental representation of language. Cognition, 10, 39-52.

Briars, Diane J. (1988). An information-processing analysis of mathematical ability, 181-204. In Dillon, Ronna R. and Schmeck, Ronald R. (1983). Individual Differences in Cognition. Vol 1, New York: Academic Press.

Brooks, R. (1977). Towards a theory of the cognitive processes in computer programming. International Journal of Man- Machine Studies, 9, 737-751.

Bruner, Jerome S., Goodnow, Jacqueline J. and Austin, George A. (1956). A study of thinking. New York: Wiley & Sons.

Buros, O.K. (1978). The Eight Mental Measurement Yearbook (Vol.2). New Jersey: Gryphon Press, 1690-1695.

Card, Stuart K., Moran, Thomas P. and Newell, Allen. (1983). The Psychology of Human-Computer Interaction. Hillsdale, NJ: Lawrence Erlbaum Associates, 469 p.

Carroll, John B. (1983). Studying individual differences in cognitive abilities: through and beyond factor analysis, 1-35. In Dillon, Ronna R. and Schmeck, Ronald R. (1983). Individual Differences in Cognition. Vol 1, New York: Academic Press.

Cheney, Paul. (1980). Cognitive style and student programming ability: an investigation. AEDS, 285-291.

Clark, H.H. (1969). Linguistic processes in deductive reasoning, from: Psychological Review, vol.76, 387-404. In Johnson-Laird, P.N. (1977), Thinking: Reading in Cognitive Science. Cambridge: Cambridge University Press.

Cook, Thomas D., Campbell Donald T. (1976). The design and conduct of quasi-experiments and true experiments in field settings. In Dunnette, Marvin D. (Ed.) Handbook of industrial and organizational psychology. New York: John Wiley and Sons. (pp.223-326).

Cronbach, Lee J. (1970). Essentials of Psychological Testing. New York: Harper and Row, 417-421.

Dillon, Ronna R. and Schmeck, Ronald R. (1983). Individual Differences in Cognition. Vol 1, New York: Academic Press.

Di Persio, Tom, Isbister, Dan and Shneiderman, Ben. (1980). An experiment using memorization/reconstruction as a measure of programmer ability. International Journal of Man-Machine Studies, 13, 339-354.

Doyle, Lauren B. (1975). Information retrieval and processing. Los Angeles, Ca.: Melville Publishing Co.

Feigenbaum, Edward A. and Feldman, Julian (Eds.) (1963). Computers and thought. New York: McGraw-Hill, 257p.

Feingenbaum, Edward A. (1970). Information processing and memory, (451-468). In Norman, Donald A. Models of Human Memory. New York: Academic press, 537 p.

Glass, Arnold Lewis, Holyoak, Keith James and Sant, John Lester (Eds.) (1979). Cognition. Reading, Mass.: Addison-Wesley, 391-433.

Garrett, M. and Fodor, J.A. (1968). Psychological theories and linguistic constructs. In Dixon, T.R. and Horton, D.L. (Eds.). Verbal Behavior and General Behavior Theory. Englewood Cliffs NJ: Prentice-Hall, 451.

Gear, William C. (1978). Applications and Algorithms in Computer Science, Module A. Chicago: Science Research Associates Inc., 179 p.

Gough, P.B. (1965). Grammatical transformations and speed of understanding. Journal of Verbal Learning and Verbal Behavior, 4, 107-111.

Greeno, James G. (1974). Hobbits and Orcs: acquisition of a sequential concept. Cognitive Psychology, 6, 270-292.

Gregg, Lee W. (1974). Knowledge and Cognition. New York: John Wiley and Sons.

Johnson-Laird, P.N. (1977), Thinking: Reading in Cognitive Science. Cambridge: Cambridge University Press.

Johnson-Laird, P.N. (1975). Reasoning with quantifiers. In Johnson-Laird, P.N. (1977), Thinking: Reading in Cognitive Science. Cambridge: Cambridge University Press.

Johnson-Laird, P.N. (1977). A theoretical analysis of insight into a reasoning task. 150-153. In Johnson-Laird, P.N. (1977), Thinking: Reading in Cognitive Science. Cambridge: Cambridge University Press.

Howell, Margaret A., Vincent, John W., and Gay, Richard A. (1967). Testing aptitude for computer programming. Psychological Reports, 20, 1251-1256.

Kahney, H. (1983). Problem solving by novice programmers. In Psychology of Computer Use. (pp.121-141). London: Academic Press.

Kintsch, Walter (1977). Memory and Cognition. New York: John Wiley and Sons., 490 p.

Konvalina, John, Stephens, Larry J. and Wileman, Stanley A. (1983). Identifying factors influencing computer science aptitude and achievement. AEDS Journal, 16(2), 106-112.

Konvalina, John, Wileman, Stanley A. and Stephens, Larry J. (May 1983). Math proficiency: a key to success for computer science students. Communications of the ACM, 26(5), 377-382.

Kurtz, Barry L. (1980) Investigating the relationship between the development of abstract reasoning and performance in an introductory programming class. Communications of the ACM, 110-117.

Larkin, Jill, McDermott, John, Simon, Dorothea P. and Simon Herbert A. (Jun 1980). Expert and novice performance in solving physics problems. Science, 208, 1335-1342.

Levine, Marvin (1963). Mediating processes in humans at the outset of discrimination learning. Psychological Review, 70(3), 254-276.

Levine, Marvin (1975). A Cognitive Theory of Learning: Research of Hypothesis Testing. Hillsdale, NJ: Erlbaum. 175-260.

Marks, L.E. and Miller, G.A. (1964). The role of semantic and syntactic constraints in the memorization of English sentences, Journal of Verbal Learning and Verbal Behavior,3, 1-5.

Mayer, Richard E., Dyck, Jennifer L. and Vilberg, William (1986, Jul). Learning to program and learning to think: What's the connection? Communications of the ACM, 29(7), 605-610.

McNamara, W.J. and Hughes, J.L. (1961). A review of research on the selection of computer programmers. Personnel Psychology, 14(1), 39-51.

Newell, Allen and Simon, Herbert A. (1972). Human Problem Solving. Englewood Cliffs, New Jersey: Prentice-Hall.

Norman, Donald A. (1970). Models of Human Memory. New York: Academic Press, 537p.

Paivio, Allan (1971). Imagery and Verbal Processes. New York: Holt, Rinehart and Winston, 596 p.

Pask, G. (1976). Styles and strategies of learning. British Journal of Educational Psychology, 46, 128-148.

Perfetti, Charles A. (1983). Individual differences in verbal processes,65-104. In Dillon, Ronna R. and Schmeck, Ronald R. (1983). Individual Differences in Cognition. Vol 1, New York: Academic Press.

Petersen, Charles G. and Howe, Trevor G. (1979). Predicting academic success in introduction to computers. AEDS. 183-191.

Reed, Stephen K., Ernst, George W., and Banerji, Ranan. (1974). The role of analogy in transfer between similar problem states. Cognitive Psychology, 6, 436-450.

Rowe, Helga A.H. (1985). Problem Solving and Intelligence. Hillsdale NJ: Lawrence Erlbaum, 392 p.

Sachs, J. (1967). Recognition memory for syntactic and semantic aspects of connected discourse. Perception and Psychophysics, 2, 437-442.

Sengler, H.E. (1983). A model of the understanding of a program and its impact on the design of the programming language Grade. In Psychology of computer use (91-106). London: Academic Press.

Shneiderman, B. (1977). Measuring computer program quality and comprehension. International Journal of Man-Machine Studies, 9, 465-478.

Shneiderman, Ben. (1980). Software Psychology: Human Factors in Computer and Information Systems. Cambridge, Mass.: Winthrop Publishers, Inc.

Shneiderman, Ben and Mayer, Richard (1979). Syntactic/Semantic interactions in programmer behavior: a model and experimental results. International Journal of Computer and Information Sciences, 8(3), 219-238.

Slobin, D.I. (1966). Grammatical transformations and sentence comprehension in childhood and adulthood. Journal of Verbal Learning and Verbal Behavior, 5, 219-227.

Soloway, Elliot (1986, Sep). Learning to program = learning to construct mechanisms and explanations. Communications of the ACM, 29(9), 850-858.

Soloway, Elliot, Ehrlich, Kate, Bonal, Jeffrey, and Greenspan, Judith. (1982). What do novices know about programming? in Badre, Albert and Shneirderman, B. (Eds.), Directions in Human/Computer Interaction. (27-54). Norwood, New Jersey: Ablex Publishing Corporation.

Thomas, John C. Jr., (1974). An analysis of behavior in the Hobbits-Orcs problem. Cognitive Psychology, 6, 257-269.

Van der Veer, Gerrit C., and van de Wolde, Jan E. (1983). Individual differences and aspects of control flow notations. In Psychology of computer use, (107-120). London: Academic Press.

Vassiliou, Yannis (Ed.) (1984). Human Factors and Interactive Computer Systems. Norwood, NJ: Ablex Publ. Corp. 287 p.

Weinberg, G.M. (1971). The Psychology of Computer Programming. New York: Van Nostrand Rheinhold.

Whitfield, J.W. (1978). An experiment in problem solving. Quarterly Journal of Experimental Psychology, 3, 172, 184-97.

Wickens, T.D., and Millward, R.B. (1971). Attribute elimination strategies for concept identification with practiced subjects. Journal of Mathematical Psychology, 8, 453-480.

Wolfe Jack M. (Apr. 1969). Testing for programming aptitude. Datamation, 4, 67-72.

Appendix A

Annex 1

Semantic Ability -- Test-sentence Sets

(ANSC-C)
1. a. The additional fact settled a major disagreement
   b. The arbitrary regulation provoked a civil complaint*
   c. The remaining duty involved a standard payment
   d. The impartial source identified a hidden fault
   e. The national election indicated a secure future
T: The arbitrary regulation provoked a civil grievance

(CNSC-I)
2. a. The wonderful gift preceded an exciting kiss*
   b. The raging fire gutted a condemned building
   c. The young singer carressed a pretty girl
   d. The enthusiastic painter sketched an ancient temple
   e. The polite child presented an aromatic bouquet
T: The wonderful gift preceded an exciting kiss

(F-C)
3. a. The incredible machine produced a screeching noise
   b. The playful kittens shredded a new slipper
   c. The dying man blessed a mournful daughter
   d. The thunderous explosion shook a fragile hut*
   e. The ferocious dog devoured a meaty bone
T: The thunderous explosion shook a fragile hut

(CSC-C)
4. a. The cheerful artist entertained a lonely damsel*
   b. The stubborn proprietor opened an expensive restaurant
   c. The poor musician played a rusty trumpet
   d. The greedy attendant devoured a soft pudding
   e. The old professor occupied a comfortable seat
T: The cheerful damsel entertained a lonely artist

(F-A)
5. a. The revised procedure facilitated an expected outcome
   b. The latest evidence suggested an alternative version
   c. The minor change modified a basic measure
   d. The final decision nullified a prior commitment
   e. The current effort concluded a productive program*
T: The current effort concluded a productive program

(ANSC-C)
6. a. The dull description constituted a boring chapter*
   b. The advanced technology obtained a respectable reputation
   c. The main assembly noticed an unnatural pause
   d. The solemn creed encouraged an excessive devotion
   e. The awkward incident prevented a possible agreement
T: The dull account constituted a boring chapter

(ANSC-I)
7. a. The limited text explained a complicated formula
   b. The plausible address answered a contradictory reply*
   c. The mediocre demonstration inspired a select few
   d. The annual report recommended a complete renovation
   e. The tentative reason supplied an adequate explanation
T: The plausible address answered a contradictory reply

(ASC-C)
8. a. The entire episode prefaced a foreign affair*
   b. The strange mistake altered an established conclusion
   c. The recent speculation provided an acceptable solution
   d. The careful study resolved an open question
   e. The constant hope endured an attempted extinction
T: The entire affair prefaced a foreign episode

(F-A)
9. a. The indecisive argument depressed a waiting group
   b. The analytic review maintained an objective position
   c. The early civilization originated a feudal system*
   d. The habitual behavior acquired a noble quality
   e. The popular view raised a considerable discrepancy
T: The early civilization originated a feudal system

(CNSC-C)
10. a. The rich physician carried a black umbrella
b. The pompous monarch married a triumphant queen*
c. The skillful doctor soothed a flaming sunburn
d. The sharp arrow pierced a frantic bird
e. The rickety stagecoach crossed a winding river
T: The pompous sovereign married a triumphant queen

(F-A)
11. a. The inital attempt provoked a general enthusiasm
b. The shameful event induced a subtle change
c. The incredible exhilaration created a considerable incentive*
d. The sovereign authority rejected a civil exchange
e. The forceful declaration inspired a renewed interest
T: The unnatural exhilaration created a formidable incentive

(F-C)
12. a. The marching company attracted a noisy crowd
b. The thirsty traveler noticed a remote inn
c. The exhausted boxer administered a decisive blow*
d. The elegant gentleman cut a fine figure
e. The shining water reflected an early sunlight
T: The exhausted boxer administered a decisive blow

(CNSC-I)
13. a. The strong policeman ousted a seedy beggar
b. The loving mother served and excellent family*
c. The impulsive builder decorated a stylish cottage
d. The active volcano destroyed a majestic forest
e. The aggressive settler felled an immense tree
T: The loving mother served and excellent family

(F-C)
14. a. The tired passenger lifted a heavy suitcase
b. The small band played a colorful tune
c. The commanding beauty ignored an elegant dandy
d. The tall girl wore a green dress*
e. The brutal policemen arrested a drunk sailor
T: The tall girl wore a green dress

(F-A)
15. a. The foreign custom elicited a strained contact
b. The uncertain eventuality unnerved a selected few
c. Thelegitimateconcernsindicatedarespectable intellect
d. The impartial judgement avoided a disloyal solution*
e. The upstanding citizens eliminated a troublesome alternative
T: The impartial judgement avoided a disloyal solution

(ANSC-I)
16. a. The absolute faith aroused an enduring interest*
b. The previous calculation contributed a significant result
c. The extensive investigation furnished a reasonable criticism
d. The vague notion survived a renewed concern
e. The original location fulfilled a customary requirement
T: The absolute faith aroused an enduring interest

(ASC-I)
17. a. The passive majority defeated a listless opposition*
b. The unfair attitude destroyed a promising idea
c. The free country organized a private venture
d. The actual quotation lacked a rational foundation
e. The unpleasant atmosphere replaced a dismal silence
T: The passive majority defeated a listless opposition

(F-C)
18. a. The arrogant gentleman smoked a rancid cigar
b. The buoyant steamer sailed a tossing ocean
c. The carefree student climbed a high tower
d. The dynamic lecturer captivated an energetic committee
e. The noisy priest amused a lively infant*
T: The noisy priest amused a lively infant

(CSC-C)
19. a. The hollow tomb housed a decaying corpse
b. The rolling hillside surrounded a muddy valley*
c. The reckless baron flicked a shining coin
d. The caustic prosecutor accused a frightened prisoner
e. The rampaging caravan trampled an orderly caravan
T: The rolling valley surrounded a muddy hillside

(CSC-I)
20. a. The fat woman polished a red apple
b. The offensive performer cheered a zealous speaker*
c. The crippled juggler sported a gaudy costume
d. The fidgety wife folded a crinkled newspaper
e. The alert fisherman swatted a buzzing mosquito
T: The offensive performer cheered a zealous speaker

(CNSC-C)
21. a. The vicious hound chased a wild animal*
b. The tortured slave uttered a deafening shriek
c. The destructive army pillaged a prosperous village
d. The colorful snake crushed a screaming beast
e. The talkative admiral attended a costume party
T: The vicious dog chased a wild animal

(F-C)
22. a. The friendly banker purchased a blue automobile
b. The savage storm flattened a beautiful flower
c. The bright headlight illumined a gloomy street
d. The falling rock killed a sinful captive
e. The muscular blacksmith lifted a bulky hammer*
T: The muscular blacksmith lifted a bulky hammer

(ASC-C)
23. a. The plain alteration introduced an essential balance
b. The mistaken assumption preserved a naive rationale*
c. The thrifty business registered an average profit
d. The preliminary hypothesis predicted an unstable relationship
e. The available literature cited a useful article
T: The mistaken rationale preserved a naive assumption

(CSC-I)
24. a. The spirited leader slapped a mournful hostage*
b. The white foam topped a restless sea
c. The jagged stone shattered a clear window
d. The brutal officer snapped an abrupt salute
e. The delicate maiden watched a golden sunset
T: The spirited leader slapped a mournful hostage

(ASC-I)
25. a. The former custom abused a moral principle
b. The introductory statement promised a logical treatment
c. The rural community ensured a deprived childhood
d. The last crisis created a real necessity
e. The close supervision guaranteed a strict obedience
T: The introductory statement promised a logical treatment

Codes for Test Sets

CNSC-C: Concrete, Non-semantic Change, Concrete
CSC-C: Concrete, Semantic Change, Changed
CNSC-I: Concrete, Non-semantic Change, Identical
CSC-I: Concrete, Semantic Change, Identical
ANSC-I: Abstract, Non-semantic Change, Identical
ASC-I: Abstract, Semantic change, Identical
ASC-C: Abstract, Semantic Change, Changed
ANSC-C: Abstract, Non-semantic Change, Changed
F-C: Filler, Concrete
F-A: Filler, Abstract

Annex 2

SEMANTIC ABILITY TEST--ANSWER SHEET

1. Identical
   Changed-Same Meaning
   Changed-Different Meaning

2. Identical
   Changed-Same Meaning
   Changed-Different Meaning

3. Identical
   Changed-Same Meaning
   Changed-Different Meaning

4. Identical
   Changed-Same Meaning
   Changed-Different Meaning

5. Identical
   Changed-Same Meaning
   Changed-Different Meaning

6. Identical
   Changed-Same Meaning
   Changed-Different Meaning

7. Identical
Changed-Same Meaning
Changed-Different Meaning

8. Identical
Changed-Same Meaning
Changed-Different Meaning

9. Identical
Changed-Same Meaning
Changed-Different Meaning

10. Identical
Changed-Same Meaning
Changed-Different Meaning

11. Identical
Changed-Same Meaning
Changed-Different Meaning

12. Identical
Changed-Same Meaning
Changed-Different Meaning

13. Identical
Changed-Same Meaning
Changed-Different Meaning

14. Identical
Changed-Same Meaning
Changed-Different Meaning

15. Identical
Changed-Same Meaning
Changed-Different Meaning

16. Identical
Changed-Same Meaning
Changed-Different Meaning

17. Identical
Changed-Same Meaning
Changed-Different Meaning

18. Identical
Changed-Same Meaning
Changed-Different Meaning

19. Identical
Changed-Same Meaning
Changed-Different Meaning

20. Identical
Changed-Same Meaning
Changed-Different Meaning

21. Identical
Changed-Same Meaning
Changed-Different Meaning

22. Identical
Changed-Same Meaning
Changed-Different Meaning

23. Identical
Changed-Same Meaning
Changed-Different Meaning

24. Identical
Changed-Same Meaning
Changed-Different Meaning

25. Identical
Changed-Same Meaning
Changed-Different Meaning

Appendix B

Materials--Master Mind Game

Figure 3 shows the organisation of the de-coding board.

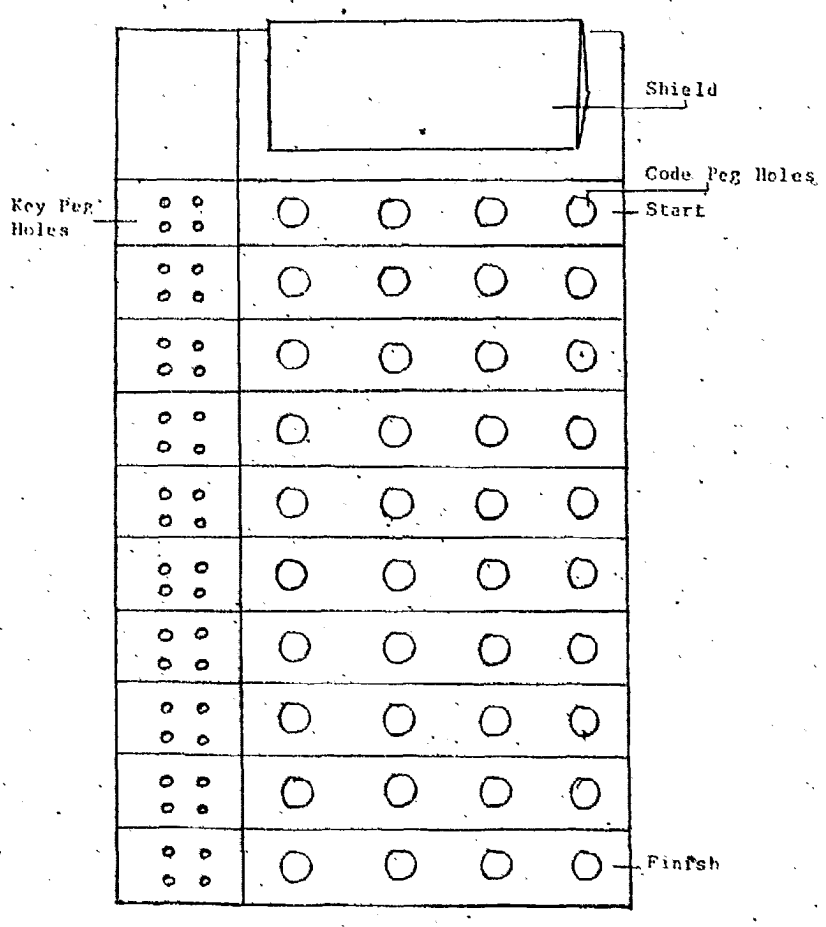

Figure 3. Master Mind De-coding Board.

Appendix C

Materials--Questionnaire

1. What is your gender?
   a)Male
   b)Female

2. What is your age?
   a)19 or younger
   b)20 to 24
   c)25 to 29
   d)30 to 34
   e)35 or older

3. Rate your high school performance using the following categories:
   a)A (86-100%)
   b)B (76-85%)
   c)C (66-75%)
   d)D (65% or below)

4. What is your current university level?
   a)First year
   b)Second year
   c)Third year
   d)Honors year
   e)Graduate School
   f)Graduated
   g)No Degree (give education level)_____

5. What is your degree major? (please write down major beside main area)
   a)Arts _____
   b)Science _____
   c)Commerce _____

6. If you are still in university, rate your current academic performance using the following categories:
   a)A (86-100%)
   b)B (76-85%)
   c)C (66-75%)
   d)D (65% or below)

7. Have you ever used a computer?
   a)Yes
   b)No

*If you have answered yes to question 7, please go on to question 8. If you have answered no, please go to question 14.

8. What type of experience do you have?
   a)user only
   b)programming as an accessory to other tasks
   c)programming as a major task and/or as a job

9. What type of computer do you usually use?
   a)a mainframe computer (such as VAX or CYBER)
   b)a mini or microcomputer
   c)both types

10. How much prior education have you had in computer programming?
   a)None
   b)1 to 2 courses
   c)3 to 5 courses
   d)degree in computer science

11. How many programming language can you use?
a)None
b)One
c)Two
d)Three or more

12. Please name the programming language(s) you know and can use:
_____
_____

13. How much work experience have you had that involved programming aspects of computers?
a)None
b)A few months
c)One year
d)Two to three years
e)Four years or more

14. How many years of high school math have you had?
a)0
b)1
c)2
d)3
e)4 or more

15. How many math courses have you had at the college or university level?

a)0
b)1
c)2
d)3
e)4 or more

16. Have you ever played the game of Master Mind before?
    a)Yes
    b)No

17. What is your mother tongue?
    a)English
    b)French
    c)Other (Please Specify)_____

(Adapted from Konvalina et al., 1983)

Appendix D

Table D-1
ANOVA of Rows Performance on Master Mind for General Population and Experienced Programmers

Source	df	SS	Mean Square	F
Group	1	44.49	44.49	9.42**
Error	74	344.65	4.72	
Trials	3	.63	.21	.06
Trials x Group	3	7.98	2.65	.82
Error	219	713.76	3.26	

* .05, ** .01, *** .001

Table D-2
ANOVA of Time Performance on Master Mind for General Population and Experienced Programmers

Source	df	SS	Mean Square	F
Group	1	1923.2	1923.2	.9
Error	73	155491.3	2130.0	
Trials	3	8526.76	2842.25	2.53*
Trials x Group	3	8847.31	2949.10	2.63*
Error	219	245602.86	1121.47	

* .05, ** .01, *** .001

Table D-3
ANOVA of Overall Performance on Master Mind for Control and Novice Programmers

Source	df	SS	Mean Square	F
Group	1	2143.13	2143.13	1548.93***
Error	33	44.27	1.38	
Experience x Performance	1	.67	.67	1.38
Error	33	15.49	.48	

* .05, ** .01, *** .001

Table D-4
ANOVA of Overall Performance on Semantic Ability Test for Control and Novice Programmers

Source	df	SS	Mean Square	F
Group	1	3750.37	3750.37	917.86***
Error	33	130.75	4.09	
Experience x Performance	1	4.25	4.25	2.27*
Error	33	59.84	1.87	

* .05, ** .01, *** .001