A Conceptual Framework for Analyzing Students' Knowledge of Programming

Tanya J. McGill & Simone E. Volet

Murdoch University, Australia

Abstract

This article proposes a conceptual framework for analyzing students' knowledge of programming. The framework integrates three distinct types of programming knowledge identified in the educational computing literature (syntactic, conceptual, and strategic) with three distinct forms of knowledge proposed in the cognitive psychology literature (declarative, procedural, and conditional). Analysis of empirical data from a previous experimental study (Volet, 1991) provided support for the usefulness of the model and its educational potential for diagnosing deficiencies in the programming knowledge of novice programmers during a course of instruction and for designing appropriate instruction in introductory programming. (Keywords: cognitive skills, learning processes, mental models, programming, teaching methods.)

One of the recurring issues in the educational computing literature is the conceptual usefulness of distinguishing between different types of programming knowledge. A number of empirical studies have provided evidence that introductory courses in computer programming tend to overemphasize students' acquisition of the syntax of a particular programming language at the expense of their development of problem-solving strategies and their understanding of computer-programming principles.

The importance of examining the nature, structure, and function of domain-specific knowledge has also been emphasized in recent cognitive psychology literature. Different concepts have been proposed to categorize forms of knowledge, but similar claims have been made with regard to the significance of the higher order problem-solving or metacognitive knowledge and strategies that allow individuals to apply their knowledge flexibly across situations.

This article reviews the categories of knowledge emerging from each of these two bodies of literature and attempts to integrate them into a single conceptual framework. Empirical data are provided to support the usefulness of the model and its educational potential for diagnosing deficiencies in students' knowledge and for designing instruction in introductory programming courses.

EDUCATIONAL COMPUTING LITERATURE

Much of the educational computing research has focused on the cognitive processes involved in learning to program and on the types of programming knowl-

This research was supported by a grant from Murdoch University.

edge acquired by students during an introductory programming course. It has been argued that programming involves the acquisition and effective use of the following three interrelated types of programming knowledge (Bayman & Mayer, 1988).

- Syntactic: Knowledge of specific facts about a programming language and rules for its use.
- Conceptual: Understanding of computer programming constructs and principles.
- Strategic: Programming-specific versions of general problem-solving skills.

Bayman and Mayer's model builds on the earlier work of Shneiderman (1977), Shneiderman and Mayer (1979), and Linn (1985).

Shneiderman and Mayer (1979) proposed a syntactic–semantic model of programming knowledge to represent the knowledge involved in learning to program. In their model, syntactic knowledge is conceptualized as knowledge about the rules and procedural aspects of a programming language, such as the syntax of specific looping constructs. Semantic knowledge represents an understanding of the underlying principles and constructs and is considered to be independent of the particular programming language.

Linn's (1985) developmental model describes the cognitive skills and knowledge involved in learning to program in terms of a chain of cognitive accomplishments. The chain is derived from research analyzing the behaviour of expert and novice programmers as well as examination of exemplary textbooks (Linn, 1985). The chain is composed of the following three steps:

1. Acquisition of knowledge about language features.
2. Learning of design skills. This step consists of students' development of a repertoire of templates and the procedural skills to combine these templates to solve programming problems.
3. Development of problem-solving skills that can be applied to a variety of other formal systems.

Linn and colleagues' first step in the chain of cognitive accomplishments corresponds roughly to the acquisition of syntactic knowledge in Shneiderman and Mayer's (1979) model. Their second step was their description of the acquisition of semantic knowledge. However, Linn and colleagues make a further distinction between the ability to design solutions to closely related problems (second step), and the ability to develop solutions to novel and complex problems (third step). At that last stage of development, individuals are assumed to have developed a robust and integrated understanding of programming. They are capable of adopting a self-regulated approach to solving computer-programming problems and of using their problem-solving skills flexibly from one programming situation to another, regardless of the programming language used. These problem-solving skills are seen as becoming more and more general and even dissociated from the discipline in which they have been developed. Linn argues that the third step of her chain of cognitive accomplishments can be applied to nonprogramming problems, for example, Newtonian physics. Accord-

ing to Husic, Linn, and Sloane (1989), a critical component of this last step is "the ability to monitor one's own progress, reflect on alternative approaches for solving complex problems, and to autonomously seek information" (p. 571).

Bayman and Mayer (1988) developed a similar model to Linn's, but confined the domain of their version of the model to solving problems using the chosen language. Their model forms the basis of the most commonly used representation of types of programming knowledge—syntactic, conceptual, and strategic.

Syntactic knowledge is defined as knowledge of the language features or grammar (e.g., facts relating to the use of semicolons in Pascal or knowledge of the syntax of the "Repeat...Until" construct). This technical knowledge is necessary to write programs that will compile, but it is not sufficient for designing and developing appropriate programs to solve problems. Syntactic knowledge must be accompanied by a sound understanding of the programming principles necessary to develop logically correct programs.

The acquisition of conceptual knowledge involves the development of mental models of the system and the semantics of the actions that are executed by the program. Conceptual knowledge represents the understanding of program concepts and facilitates the combination of language features embedded in a program. Whereas syntactic knowledge relates to specific knowledge of individual language constructs, conceptual knowledge entails a full understanding of the semantics of these constructs and the ways in which they can be combined to solve a problem. The use of program-design skills requires conceptual knowledge.

Strategic knowledge is the ability to use syntactic and conceptual knowledge in the most appropriate and effective way to solve novel programming problems. Strategic knowledge is essential for recognizing that a problem can be solved and for identifying appropriate techniques for doing so. It is, therefore, necessary in the problem decomposition and design phases of the programming process. It is also crucial for testing programs and debugging logic errors.

A number of studies have provided empirical support for the usefulness of distinguishing among these three categories of knowledge. In a series of experiments with novice and expert programmers, Adelson (1981) demonstrated that novices used a syntax-based organization to represent programming concepts, whereas experts used a more abstract organization based on the program's function. She suggested that as expertise increases, the categorization of the language changes from being syntactically based to being semantically based. Her 1984 work provided further support for this claim.

Linn (1985) reported the results of an integrated set of studies that were designed to assess the levels of programming skill of middle school–level beginning programmers. The studies examined the types of knowledge hypothesized in her "chain of cognitive accomplishments." Original instruments were developed to assess the first two links of the chain; the third stage was assessed as students' ability to learn a new programming language. Her results supported the significance of identifying types of knowledge and revealed a wide range of achievement levels for each type of knowledge among individuals and among schools at the end of an introductory unit. For example, average scores on the instrument assessing syntactic knowledge ranged between 20% and 68%. Overall, Linn's studies emphasized the fact that in the majority of cases, after a 12-week introductory programming class,

students' achievement was still concentrated at the beginning of the chain, with little conceptual or strategic knowledge attained.

In an experimental study by Bayman and Mayer (1988), an attempt was made to examine the relationship between conceptual and strategic knowledge. In that study, novice programmers learnt BASIC programming from either a standard manual or one that emphasized conceptual models of the language. It was found that the teaching of conceptual models enhanced problem-solving performance (strategic knowledge) and that problem-solving performance was strongly related to measures of conceptual knowledge.

Overall, the distinctions among types of programming knowledge in the educational computing literature appear to be conceptually consistent across studies and to have strong empirical support. Such distinctions also have a promising educational potential. Empirical research that studies the teaching of programming have pointed to deficiencies in the types of knowledge acquired by students in many introductory programming courses. A close examination of the nature of the knowledge acquired by students who did not reach the desired level of achievement indicates that they did acquire some syntactic knowledge, but failed to develop conceptual and strategic knowledge (Linn, 1985; Oliver, 1990). This outcome is quite consistent with the emphasis that has been typically placed on the acquisition of syntactic knowledge in introductory classes and many introductory textbooks.

De Corte, Verschaffel, and Schrooten (1992) and Linn and Clancy (1992) argue that most introductory programming classes do not put enough emphasis on fostering students' understanding of the concepts being used. As a consequence, the development of conceptual and strategic knowledge is almost entirely left to unguided discovery. Students end up with a "fragile" knowledge of programming, which is described by Schwartz, Perkins, Estey, Kruidenier, and Simmonds (1989) as garbled or inert knowledge. This refers to either knowledge that is used inappropriately or knowledge that is not spontaneously accessed in the context of need. Studies in which the teaching focus is adjusted towards conceptual and strategic knowledge have demonstrated increased levels of student understanding and achievement (cf., Bayman & Mayer, 1988; Linn & Dalbey, 1985; Oliver & Malone, 1993; Volet, 1991). This is particularly the case for low-ability students who may be less able to develop their own conceptual models and, hence, benefit greatly from explicit instruction in this area (Bayman & Mayer; Snow & Lohman, 1984; Volet & Lund, 1994).

COGNITIVE PSYCHOLOGY LITERATURE

The distinction among different forms of knowledge is common in the cognitive psychology literature. The distinction between declarative knowledge, or "know that," and procedural knowledge, or "know how," is of major importance in many analyses of knowledge structure (Anderson, 1976, 1987; Gagné, 1985). More recently, some cognitive theorists (Paris, Lipson, & Wixson, 1983; Winograd & Chou Hare, 1988) have proposed conditional knowledge, or the "know when, where, and why" it is appropriate to apply a piece of knowledge, as a separate category of knowledge.

Declarative knowledge is commonly defined as knowledge about something, specifically about some facts, concepts, or principles. It can also refer to a conceptual awareness of the functions and purposes of that knowledge. It is generally assumed that an individual's declarative knowledge base is organized into meaningful knowledge structures called "schemata" (Rumelhart, 1980) or "propositional networks" (Anderson, 1983). These metaphors for describing the structure and function of declarative knowledge are particularly useful for explaining how people make sense of new information and how they acquire new knowledge. According to Rumelhart and Norman (1978), knowledge initially develops through "accretion," which refers to the process of encoding new information in terms of existing schemata. Then, in a second step, knowledge develops either through "tuning," which refers to the refinement of an existing schema, or through "restructuring," which involves the creation of a new schema. Applied to the domain of computer programming, declarative knowledge refers to an individual's knowledge about the syntax of a particular language, for example, the knowledge that a semicolon is needed at the end of a statement in Pascal (reference to a fact), or to an individual's knowledge about programming principles, for example the ability to explain what a fragment of pseudocode does (reference to concepts, rules, or principles). Declarative knowledge of programming can be taught through verbal or written explanations, or extracted from demonstrations and coded programs. Declarative knowledge is typically expressed through language. Declarative knowledge that is available is not necessarily useable. According to Flavell and Wellman (1977), cases of "production deficiencies," situations in which individuals appear to possess some knowledge that they are unable to use, are common. Karmiloff-Smith (1986) found that children who have been taught a rule to perform a mathematical operation may be unable to apply that rule, even though they have encoded and stored it in a declarative form and are able to recite it perfectly when required.

Procedural knowledge refers to the active use of the declarative knowledge base when solving problems. It is commonly defined as knowledge of how to do something. Procedural knowledge is not expressed through language; it is demonstrated in action. According to Anderson (1987), knowledge acquisition starts at the declarative stage, in which knowledge of facts and principles is acquired in a propositional form and must be consciously activated to be used. Gradually, after seeing, practicing, and reflecting upon examples, declarative knowledge is converted into procedures (or procedural knowledge) by way of a "proceduralization" process. With additional practice and experience, procedural knowledge becomes automatic and its use becomes less mentally demanding. Although Anderson claimed that his model applies to all complex learning, it is usually presented as a model of skill learning (Shuell, 1990). Not all cognitive theorists agree with Anderson's assertion that the proceduralization process is the second developmental step towards greater competence (the first representing the encoding of knowledge in a declarative form). Chi and Bassok (1989) argue that for successful proceduralization to take place, an individual must fully understand all the relevant declarative principles that govern the derivation of a solution. This does not appear to be the case for all learning. There

is evidence that many skills and procedural knowledge (e.g., algebraic or arithmetic computations) are algorithmic in nature. They can be learnt as a set of procedures simply by observation and imitation, and they can be performed correctly with only a syntactic understanding of the problem. Piaget (1978) argued and demonstrated that young children are able to perform complex cognitive activities that they are able to explain in terms of the underlying principle only at a later stage of development. In schools, a lot of emphasis is put on the development of propositional or algorithmic forms of knowledge, and, often, not enough is placed on the development of the declarative principles necessary to guide problem solving and derive an appropriate set of procedures in a meaningful way. It should be noted however, that although it is useful to distinguish declarative knowledge from procedural knowledge, the two forms of knowledge are seldom found in isolation from each other.

Applied to the domain of computer programming, procedural knowledge refers to people's ability to use their programming knowledge (syntactic, conceptual, or both) to write pieces of code or complete programs. Procedural knowledge can range from coding simple statements to complex solution processes. Anderson's studies of LISP learning identified about 500 production rules necessary to encode the skill of programming in LISP (Anderson, Conrad, & Corbett, 1989). Anderson's ACT theory discusses a process called knowledge compilation, which involves converting the initial interpretive use of declarative knowledge to a procedural production–rule form. Thus, according to Anderson, students must encode declarative representations of what particular LISP functions do and use them to guide their programming. The relevant information must be initially encoded in declarative knowledge structures and then transformed into procedural form through continued practice.

Most theoretical models of the nature of knowledge, such as Anderson (1982) or Gagné (1985), identify only two types of knowledge: declarative and procedural. In such models, procedural knowledge is implicitly assumed to include the know-how as well as the conditions of applicability of the production; that is, the knowledge of when and where it is appropriate to apply that knowledge. However, since the late 1970s, it has been claimed that "success at complex and challenging tasks requires more than just a great deal of declarative and procedural knowledge" (Goetz, Alexander, & Ash, 1992). Prominent cognitive and developmental psychologists, like Flavell (1979) and Brown (1980) argued that an individual's knowledge and control of one's cognition should be recognized as a separate category of knowledge. Paris et al. (1983) introduced the term conditional knowledge to indicate that the essence of metacognitive knowledge relates to the conditions of applicability of what has been learnt. According to Winograd and Chou Hare (1988) conditional knowledge is communicated "when the teacher explains to students why a strategy is important, when and where to use the strategy, and how to evaluate its effectiveness" (p. 134). Their review of the nature of teacher explanation in a number of experimental studies revealed that when conditional knowledge was fostered, significant gains could be seen in the use of the strategy that had been taught.

A CONCEPTUAL FRAMEWORK OF THE VARIOUS COMPONENTS OF PROGRAMMING KNOWLEDGE

Comparing the educational computing and the cognitive psychology bodies of literature on categories of knowledge reveals some interesting similarities and differences. The major similarity appears in the conceptualization of the highest category of knowledge, which is labeled strategic knowledge in the educational computing literature, and conditional knowledge in the cognitive psychology literature. Regardless of the discipline, it is expected at that expert level of knowing that the individual has developed a rich body of abstract, organized, and principled knowledge and effective cognitive skills for recognizing meaningful patterns, accessing relevant conceptual knowledge, and generating appropriate problem solutions. It is agreed that at the highest level, knowledge is flexible, transferable, and applicable across situations and tasks.

The major difference between the two bodies of knowledge lies in the conceptualization of the two lower level categories of knowledge. In the educational computing literature, the distinction is generally made between two forms of content (programming) knowledge, which are viewed as complementary but at the same time hierarchically related. It is assumed that syntactic knowledge is related to conceptual knowledge of programming, but that the former is of a lower level because it is more concrete than abstract and because conceptual knowledge of computer programming can subsume various types of syntactic knowledge. In contrast, in the cognitive psychology literature, the distinction between categories of knowledge is unrelated to the subject-matter content but refers to the nature, structure, and function of knowledge. Declarative knowledge and procedural knowledge are complementary and closely interrelated, but more in a developmental than in a hierarchical fashion.

Integrating these two sets of concepts into a single conceptual framework appears useful for a full understanding of the nature of computer-programming knowledge. Mandinach and Linn (1986) refer to the use of procedural skills necessary to combine language features (syntactic knowledge) and templates (conceptual knowledge) in a program. They define procedural skills as planning and testing skills. Anderson (1989) and Shih and Alessi (1993–1994) have applied cognitive psychology concepts to the educational computing domain, however they have not systematically explored the relationships among the cognitive psychology concepts—declarative and procedural knowledge—and the more common educational computing concepts—syntactic, conceptual, and strategic knowledge.

Palumbo (1990) and Reed and Palumbo (1992) addressed the issue of the potential relationship between programming language instruction and problem solving. They argued that the first two components of Linn's (1985) chain of cognitive accomplishments (namely, the acquisition of knowledge about language features and the development of design skills) "roughly parallel the distinction between declarative knowledge and procedural knowledge and the distinction between syntactic knowledge and semantic knowledge" (Palumbo, p. 79). Palumbo views declarative and syntactic knowledge bases as equivalent

terms (first component in Linn's model), and considers the development of design skills (second component in Linn's model) as equivalent to the development of procedural knowledge and semantic knowledge of the components in the declarative or syntactic knowledge base.

Oliver (1993) also makes reference to parallels between the categorizations of knowledge used in the educational computing and cognitive psychology literature and incorporates the distinctions in domain-specific knowledge into an elaborate understanding of the nature of conceptual knowledge. He proposes that conceptual knowledge incorporates the types of knowledge identified in the cognitive psychology literature. To further Oliver's distinction, a similar distinction could be applied to syntactic knowledge. A full model recognizes that each type of knowledge can subsume the other. For example, declarative knowledge includes both syntactic and conceptual knowledge. The same applies to procedural knowledge. Reciprocally, syntactic knowledge can be represented in a declarative and in a procedural form, as can conceptual knowledge. For example, the second link in Linn's (1985) chain of cognitive accomplishments, design skills, contains both knowledge of templates, which can be considered declarative knowledge, and the procedural skills that enable the use of templates. Soloway (1985) discusses the existence of both procedural and nonprocedural plans (templates) and expresses reservations about the value of nonprocedural plans, thereby acknowledging that conceptual knowledge may have both declarative and procedural components.

A two-dimensional model of the various components of programming knowledge distinguishes among four interrelated but conceptually distinct forms of programming knowledge. The four categories of knowledge are declarative-syntactic, declarative-conceptual, procedural-syntactic, and procedural-conceptual. It is assumed that together these knowledge categories form the basis of programming knowledge. The ability to know why, how, where, and when this knowledge can be used appropriately leads to a conceptually distinct higher form of knowledge called strategic or conditional. Strategic/conditional knowledge refers to the ability to integrate and orchestrate the use of all other forms of knowledge. This higher level of knowledge development is achieved when an individual is able to use procedural knowledge (syntactic and conceptual) flexibly and appropriately across novel situations and tasks in a way that is syntactically correct and that reflects a sound understanding of the semantics of the actions executed by the program (declarative-conceptual). Table 1 presents the conceptual framework of the various components of programming knowledge, with some examples within each category.

Declarative-Syntactic Knowledge

This category represents knowledge of syntactic facts related to one particular programming language. This form of knowledge is the one typically introduced at the beginning of introductory computer programming courses and then taught throughout as students are continually introduced to new syntactic constructs. It can be presented in lectures and learnt from books. No conceptual understanding of computer programming knowledge is assumed, nor is the ability to use that form of knowledge in a program.

Table 1
A Conceptual Framework of the Various Components of Programming Knowledge

	Declarative Knowledge	Procedural Knowledge
Syntactic Knowledge	Knowledge of syntactic facts related to a particular language, such as: • Knowing that a semicolon is needed to end each statement in Pascal. • The ability to explain the syntactic differences between a procedure and a function in BASIC	Ability to apply rules of syntax when programming, such as: • Ability to write a syntactically correct REPEAT statement in Pascal. • The ability to open a text file and read from it using BASIC.
Conceptual Knowledge	Understanding of and ability to explain the semantics of the actions that take place as a program executes, such as: • The ability to explain what a fragment of pseudocode does. • Knowing the way in which the result of a function activation is returned in a particular language.	Ability to design solutions to programming problems, such as: • The ability to design a procedure to compute the mean of some data. • The ability to modify a program that prints a 1-D array to print a 2-D array.
Strategic/Conditional Knowledge The ability to design, code, and test a program to solve a novel problem.		

Declarative-Conceptual Knowledge

This category of knowledge includes the understanding of and ability to explain the semantics of the actions that take place as a program is executed. This form of knowledge differs from both forms of procedural knowledge in that its availability does not ensure that students can apply it. Students may possess a

conceptual understanding of how a particular program works without having the ability to write the program themselves because the necessary knowledge may not be spontaneously accessed when needed (Schwartz et al., 1989). Declarative-conceptual knowledge can be taught in lectures and tutorials or extracted from observing programs and executing them.

Procedural-Syntactic Knowledge

This category of knowledge refers to the ability to apply rules of syntax when programming, that is to be able to produce syntactically correct statements in a programming language. This form of knowledge is emphasized throughout introductory computer programming courses by way of the practical laboratory component, during which students undertake programming exercises. Possession of this form of knowledge does not, however, guarantee understanding of the semantics of the syntactically correct statements (i.e., does not ensure conceptual knowledge). Research studies show that procedural-syntactic and declarative-syntactic knowledge are emphasized in many introductory programming courses, and, thus, students become familiar with the syntax of a programming language. However, the same students may be unable to comprehend the concepts being used. Linn and Dalbey (1985) suggest that some procedures can be used correctly with only a syntactic understanding of the problem, for example when using simple (surface-level) analogy with similar types of problems.

Procedural-Conceptual Knowledge.

This category of knowledge refers to the ability to use semantic knowledge of programming to write programs. This type of knowledge is frequently not taught explicitly; instead students are expected to acquire it as a result of the declarative knowledge presented in lectures and tutorials and their experiences undertaking hands-on programming exercises. However, in studies in which the teaching of conceptual models is emphasized, increased levels of student achievement are frequently reported (e.g., Bayman & Mayer, 1988; Oliver & Malone, 1993; Schwartz et al., 1989; Volet, 1991).

Strategic/Conditional Knowledge

This category refers to the ability to use syntactic and conceptual knowledge effectively to design, code, and test a program that solves a novel problem. The individual is also able to explain the semantics of the actions executed by the program; hence, he or she possesses both declarative and procedural knowledge. Again, this type of knowledge is often not explicitly taught in introductory programming classes; it is assumed that students will acquire it independently (De Corte et al., 1992; Linn & Clancy, 1992). Studies that have emphasized the teaching of conceptual and strategic knowledge have demonstrated increased levels of student understanding and achievement (e.g., Bayman & Mayer, 1988; Linn & Dalbey, 1985; Volet, 1991).

EMPIRICAL SUPPORT FOR THE TWO-DIMENSIONAL CONCEPTUAL FRAMEWORK

Empirical support for the conceptual and educational usefulness of distinguishing among five categories of programming knowledge was sought by reanalyzing data from a previous experimental study (Volet, 1991). In that study, Volet investigated the impact of an instructional package that emphasizes metacognitive instruction on the achievement and satisfaction of students enrolled in an introductory Turbo BASIC programming course.

The results of that study indicated that the instructional method had significant short-term and long-term effects on students' learning outcomes as reflected in their pass rates and overall course marks. However, although revealing the efficacy of the instructional method, the use of examination marks did not provide explicit information on the types of knowledge induced through this form of instruction. Because significant differences were observed between experimental and control students' scores on the examination question requiring them to design and write a program to solve a novel problem, it was implicitly assumed that the method had fostered students' development of procedural-conceptual and strategic/conditional knowledge; this effect, however, was not empirically examined. Identifying the types of programming knowledge that differentiated experimental and control students in that study was expected to provide empirical support for the conceptual and educational usefulness of distinguishing among the proposed five categories of programming knowledge.

VOLET (1991) STUDY

Method

The experimental group ($n = 28$) consisted of two intact tutorial classes (out of nine) and a control group of 28 matched students attending other tutorial classes. The two experimental classes were taught by the same tutor. Each control student represented the best possible match for an experimental student at the beginning of the course. Pairing was initially based on students' background in computing, and program of study (i.e., the discipline in which students were majoring). Gender, interest in computing, and initial study goals for the course were used as the second set of criteria for obtaining the best overall match because these factors have all been found to be significant in previous research on the achievement of first-year computing students.

The intervention was conducted weekly during tutorial time throughout a 13-week course. The instructional package consisted of an interactive teaching approach involving modeling, coaching, and collaborative learning and emphasized student use of a planning strategy for algorithm development and programming. The planning strategy was introduced in the first tutorial and used in all subsequent tutorials. Students were asked to use the strategy on all their programming exercises and were given feedback on their algorithm development and their completed programs. Demonstrations were initially performed by the

tutor, who acted as the expert, with the students gradually taking over and actively engaging in the teaching and learning process. The approach required a large amount of tutor–group verbal interaction. Collaboration among students was explicitly encouraged during tutorials and outside classes.

Students in all other tutorial groups were tutored in a traditional way, that is, they were required to work on set exercises while tutors acted as consultants, giving occasional group explanations. Control students (like experimental students) were introduced to algorithm development in the lectures, but algorithm development was not formally required in the weekly exercises or assignments, and no modeling or coaching techniques were used in tutorials to facilitate student acquisition of these skills.

Students' estimations of time spent on various activities during tutorials and students' ratings of the usefulness of tutorial activities provided evidence of major differences between the instructional method used in the intervention and normal practice in tutorials for this course. More time was spent overall within the experimental group on activities involving the tutor interacting with the whole group (on average 40 min, or 66.9% of the whole tutorial time, compared to 12 min, or 19.9% for control students). Consequently, less time was left for the more traditional activities (independent work on the weekly exercises and getting personal help from the tutor for programming problems). Students in the experimental group also rated the usefulness of tutor–group activities more highly than did students in the control group (Volet, 1991).

HYPOTHESES FOR THE PRESENT STUDY

Based on the nature of the experimental instructional method (with its emphasis on achieving a sound understanding of programming principles and the development of problem-solving skills) and on reports from other relevant experimental work on student learning in introductory programming courses three hypotheses were generated for the present investigation. It was expected that:

1. Experimental students in Volet's (1991) study would have achieved higher levels of procedural-conceptual knowledge than students in the control condition.
2. Experimental students would have achieved higher levels of strategic/conditional knowledge than students in the control condition.
3. There would have been no difference between experimental and control students' levels of declarative-syntactic, declarative-conceptual, and procedural-syntactic knowledge.

Students' levels of declarative-syntactic and declarative-conceptual knowledge were not expected to be significantly affected by the intervention because it was thought that declarative knowledge is typically the type of knowledge that students can acquire from reading their textbooks and studying their lecture notes or coded programs. Control students in that study had the same opportunities

as experimental students to develop these forms of knowledge by attending the lectures and studying outside class.

Procedural-syntactic knowledge was also not expected to be affected by the intervention, because the experimental instructional approach was not directly aimed at the improvement of syntactic knowledge. Instead, the instructional approach focused on application of conceptual knowledge.

When examining group scores on the different categories of knowledge, it should be remembered that experimental and control students' performance could be compared only within each category of knowledge, not across types of knowledge. The reason is that the level of difficulty of the examination questions had not been standardized across categories. Therefore, no hypotheses could be generated with respect to the relative levels of the five types of knowledge.

ANALYSIS OF EXPERIMENTAL AND CONTROL STUDENTS' PROGRAMMING KNOWLEDGE

Each final examination question was considered by the authors and two academics involved in teaching introductory programming to determine the types of knowledge required to answer that question according to the proposed conceptual framework. There was mostly initial agreement on categorization and following discussion full agreement was reached for all questions. Some questions could be categorized as requiring only one single type of knowledge, whereas others were partitioned into several relevant categories.

For example, the following questions were identified as reflecting one category of knowledge:

- How does a program activate a procedure and a function? (declarative-syntactic)
- Why do we use functions in the development of a program? (declarative-conceptual)

Questions such as the following required both procedural-syntactic and procedural-conceptual knowledge: "Write the code for a procedure or function to return the larger of 2 numbers."

The final question in the examination required that a complete program be written to help with the maintenance of a university's student record system. Strategic/conditional knowledge was required to design the solution, procedural-conceptual knowledge to design the program logic, and procedural-syntactic knowledge to create a syntactically correct BASIC program.

A scoring system was created to assess the students' levels of knowledge for each of the five proposed types of programming knowledge. The scoring scheme maintained the relative question weightings of the original marking scheme used when the course was conducted, but the marks for each question were allocated to the relevant category and partitioned across categories when necessary. For

example, in the example question given previously, in which code was written to return the larger of two numbers, half the marks were allocated based on the syntactic correctness of the code written (procedural-syntactic) and half on the conceptual correctness of the code (procedural-conceptual). Table 2 shows all the examination questions with their categorizations and the percentage weighting for questions that related to more than one category.

All examination papers were then re-marked by two independent judges blind to the experimental conditions and to the nature of the original study. The correlation coefficient between the two markers' scores on the individual categories was $r = 0.96$.

Table 2
Types of Knowledge Required to Answer the Examination Questions

Questions	Types of Knowledge
Why in the development of a program do we use:	
1. Program variables with meaningful names.	Declarative-conceptual (100%)
2. Program line indentation.	Declarative-conceptual (100%)
3. Procedures.	Declarative-conceptual (100%)
4. Uppercase sometimes and lowercase other times.	Declarative-conceptual (100%)
5. Commenting.	Declarative-conceptual (100%)
6. Block structured selection.	Declarative-conceptual (100%)
7. Functions.	Declarative-conceptual (100%)
8. Neither GOTO nor GOSUB statements.	Declarative-conceptual (100%)
9. A limited variety of iteration controls.	Declarative-conceptual (100%)
What is the difference between a procedure and a function?	Declarative-syntactic (50%), declarative-conceptual (50%)
Explain the meaning of formal and actual parameters	Declarative-syntactic (50%), declarative-conceptual (50%)
How is data transferred into a procedure?	Declarative-syntactic (50%), declarative-conceptual (50%)
What happens when the processing associated with a procedure or function is complete?	Declarative-conceptual (100%)

Table 2, cont.
Types of Knowledge Required to Answer the Examination Questions

How is the result of a function activation returned?	Declarative-syntactic (50%), declarative-conceptual (50%)
How does a program activate a procedure and a function?	Declarative-syntactic (100%)
Why are the variables used within a procedure referred to as local?	Declarative-conceptual (100%)
For each of the following situations: 1. Specify whether a procedure or a function is needed. 2. Specify the parameters. 3. Specify and special precautions needed to deal with potential problems. 4. Write code for the procedure or function. • Print a header for a program. • Find the square root of a number. • Obtain the name and age of the program user. • Retrieve a record from an opened file. • Print a warning message. • Solve a quadratic equation. • Return the larger of two numbers. • Compute the mean of some data. • Open a file.	 Procedural-conceptual (100%) Procedural-conceptual (100%) Procedural-conceptual (100%) Procedural-syntactic (50%), procedural-conceptual (50%)
Develop a program to help with the maintenance of a university's student record system. It should allow the user to: • Enter new records. • Delete existing records. • Alter existing records. • Show all records.	Procedural-syntactic (33.3%), procedural-conceptual (33.3%), strategic/conditional (33.3%)

RESULTS AND DISCUSSION

As shown in Table 3, two of the hypotheses were fully supported. Students in the experimental group attained higher levels of procedural-conceptual knowledge (60.2 versus 45.4, $t(44) = 2.39$, $p < .05$) and higher levels of strategic/conditional knowledge (66.7 versus 49.5, $t(44) = 2.53$, $p < .01$) than did the control students.

Table 3
Experimental and Control Students' Scores for Five Types of Computer Programming Knowledge

	Declarative Knowledge		Procedural Knowledge		
	Experimental x score (*sd*)	Control x score (*sd*)	Experimental x score (*sd*)	Control x score (*sd*)	*p*
Knowledge					
Syntactic	44.6 (26.2)	38.2 (25.7)	60.3 (21.6)	42.8 (24.4)	0.01
Conceptual	58.5 (16.2)	54.6 (17.2)	60.2 (21.8)	45.4 (19.9)	0.05

	Experimental x score (*sd*)	Control x score (*sd*)	*p*
Strategic/Conditional Knowledge	66.7 (22.1)	49.5 (24.0)	0.01

As expected, the levels of declarative-syntactic and declarative-conceptual knowledge were not significantly different across groups. However, contrary to our expectations, a significant difference was found between the experimental and control group scores for procedural-syntactic knowledge (60.3 versus 42.80, $t(44) = 2.58$, $p < .01$). This finding indicates that the intervention also had an impact on students' ability to produce syntactically correct program statements. This was not expected because the development of this form of knowledge was not directly targeted in the intervention, and was emphasized in the laboratory components of the course for all students. It is possible, however, that the requirement of externalizing procedural-conceptual problem-solving processes in the experimental classes had been extended to explicit justifications for the use of syntactic knowledge, which would explain the significant differences concerning this type of knowledge. An alternative explanation is motivational, and is related to the socially supportive form of instruction adopted in the experimental group. This form of instruction may have encouraged students to become generally more committed to learn more in this course.

The lack of experimental difference in declarative-conceptual knowledge, in contrast to the significant difference in procedural-conceptual knowledge, illustrates the issue of learning inert forms of knowledge, that is, knowledge not spontaneously accessed in the context of need (Schwartz et al., 1989). The form of instruction used in Volet's (1991) experimental tutorials emphasized the de-

velopment of conceptual knowledge in action (procedural-conceptual) rather than in a propositional form (declarative-conceptual).

Overall, the reanalysis of the experimental data using the conceptual framework proposed here clarified the differences between experimental and control students' learning outcomes in terms of the kinds of knowledge that were fostered in the study. The breakdown of students' achievement scores into five types of programming knowledge revealed that the experimental instructional method had a significant positive impact on students' development of the two types of procedural knowledge (conceptual and syntactic) as well as their development of the highest form of knowledge, strategic/conditional. What was not significantly affected by the intervention was students' amount of declarative knowledge, whether of a syntactic or conceptual type.

In view of these results, it could be argued that a simple distinction between declarative and procedural knowledge is sufficient for educational computing purposes and that there is no need for a more complex model. Yet, the educational computing literature provides substantial theoretical and empirical support for the usefulness of distinguishing between syntactic and conceptual knowledge of programming. As previously discussed in this article, the two sets of categories (syntactic and conceptual versus declarative and procedural) do not overlap and, in fact, can each independently subsume both terms of the other set of categories. However, more empirical research is needed to fully establish the conceptual usefulness of these four categories of programming knowledge. The present investigation could only attempt to discern differences in categories of knowledge resulting from the particular experimental instructional approach used. Different instructional approaches may target the development of other categories of knowledge.

The inclusion in the conceptual framework of strategic/conditional knowledge as a higher level type of knowledge was justified because of its theoretical importance in the educational computing and the educational psychology literature. The re-marking of Volet's (1991) experimental and control students' examination papers using the new knowledge-based coding scheme provided empirical support for the assumption that the experimental instructional package had facilitated students' development of strategic/conditional knowledge and procedural-conceptual knowledge.

IMPLICATIONS

The results of this study suggest that an instructional approach that emphasises the development and use of a planning strategy for algorithm development in conjunction with modeling, coaching, and collaborative-learning activities can have positive effects on students' development of introductory programming knowledge. As hypothesised, the categories of programming knowledge most enhanced were procedural-conceptual and strategic/conditional, with procedural-syntactic knowledge also significantly improved. As expected, students' development of declarative-syntactic and declarative-conceptual knowledge were not significantly affected, but the two groups' scores did move in the right direction.

It is argued that the success of the instructional approach was because of the explicit interface in the experimental tutorials, the declarative-conceptual knowledge presented to complete the practical exercises and write good programs. The ability to appropriately interface and use declarative and procedural forms of knowledge to solve complex and challenging tasks requires strategic/conditional knowledge (or metacognitive knowledge), which was precisely the target of the intervention. Winograd & Chou Hare's (1988) claim that when strategic/conditional knowledge is fostered, significant gains occur in appropriate use of declarative and procedural knowledge was empirically supported in the present study. Although control students were introduced to the same amount of declarative knowledge in lectures and given the same opportunities to develop procedural knowledge through the completion of practical exercises in tutorials, their development of strategic/conditional knowledge was not guided by structured interactive-modeling and cognitive-coaching instruction. These results support Oliver's (1993) assertion that computing educators should give attention not only to the mode of delivery but also to the use of appropriate rehearsal and consolidation learning activities. Recent work by Bielaczyc, Pirolli, and Brown (1993) also stresses the benefits of fostering computing students' development of metacognitive strategies relevant to the discipline in conjunction to their development of computing knowledge.

One important issue not addressed in this investigation relates to the sequencing of these different types of knowledge in computer programming instruction. Once the conceptual usefulness of the five categories of knowledge has been fully established, it will become important to examine how the different types of knowledge should be taught to optimize students' learning—whether progress through the categories should take place in a certain sequence, concurrently, or, possibly, in a cyclic fashion. The repeated finding in the empirical educational computing literature that the emphasis typically placed on the acquisition of syntactic knowledge in introductory classes and many introductory textbooks at the expense of conceptual knowledge leads to the development of inert knowledge provides support for concurrent or cyclic instructional approaches.

With regard to the declarative versus procedural dimensions, there is less agreement about which aspect of knowledge should be fostered first during instruction. Anderson's (1987) claim that procedural knowledge is acquired through a proceduralization process whereby declarative knowledge is acquired first and then gradually converted into procedures is endorsed by Chi & Bassok (1989), who argue that for successful proceduralization to take place, an individual must have fully understood the principles governing the derivation of a solution. If this is the case, more concrete improvements in declarative knowledge might be obtained by focusing on the pace at which declarative knowledge is introduced, ensuring that students have sufficient time to increase their procedural knowledge bases before further adding to their declarative knowledge bases. This is supported by Palumbo and Reed's (1991) work on the effects of BASIC programming on problem-solving ability.

However, there is also theoretical and empirical support for the notion that much procedural knowledge can be learnt through simple observation, and that

despite limitations regarding transfer of such knowledge, some skills can be performed correctly without full understanding of the underlying principles. According to Salomon & Globerson (1987), the initial acquisition of some low-level, automated types of knowledge can be very useful in some cases to facilitate the development of more complex and mentally demanding forms of learning, such as in the case of conceptual understanding and the development of conditional knowledge. Designing instruction in introductory programming courses is a complex task and requires further attention.

CONCLUSION

Three distinct types of programming knowledge emerging from the educational computing literature (syntactic, conceptual, and strategic) and three distinct forms of knowledge proposed in the cognitive psychology literature (declarative, procedural, and conditional) were integrated into a single conceptual framework. Empirical data from a previous experimental study (Volet, 1991) provided support for the usefulness of the model, its educational potential for diagnosing deficiencies in the programming knowledge of novice programmers during a course of instruction, and for designing appropriate instruction in introductory programming.

Contributors

Tanya McGill is a lecturer in information systems at Murdoch University in Western Australia. She teaches computer literacy, systems analysis and design, and programming. Her research interests include computing education, information-seeking behaviour, and end-user computing. Simone Volet is a senior lecturer in educational psychology at Murdoch University. Her research focuses on adult learning and education and the development of effective instructional models for teaching adults in academic and professional learning settings. (Address: Tanya McGill, School of Physical Sciences, Engineering and Technology, Murdoch University, Murdoch 6150, Western Australia; mcgill@murdoch.edu.au.)

References

Adelson, B. (1981). Problem solving and the development of abstract categories in programming languages. *Memory and Cognition, 9*(4), 422–433.

Adelson, B. (1984). When novices surpass experts: The difficulty of a task may increase with expertise. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 10*(3), 483–495.

Anderson, J. R. (1976). *Language, memory, and thought.* Hillsdale, NJ: Erlbaum.

Anderson, J. R. (1982). Acquisition of cognitive skill. *Psychological Review, 89*, 369–406.

Anderson, J. R. (1983). *The architecture of cognition.* Cambridge, MA: Harvard University Press.

Anderson, J. R. (1987). Skill acquisition: Compilation of weak-method problem solutions. *Psychological Review, 94,* 192–210.

Anderson, J. R., Conrad, F. G., & Corbett, A. T. (1989). Skill acquisition and the LISP tutor. *Cognitive Science, 13,* 467–505.

Bayman, P., & Mayer, R. (1988). Using conceptual models to teach BASIC computer programming. *Journal of Educational Psychology, 80*(3), 291–298.

Bielaczyc, K., Pirolli, P., & Brown, A. (1993). *Strategy training in self-explanation and self-regulation strategies for learning computer programming* (Report No. CSM-5). Berkeley, CA: University of California.

Brown, A. L. (1980). Metacognitive development and reading. In R. J. Spiro, B. C. Bruce, & W. F. Brewer (Eds.), *Theoretical issues in reading comprehension: Perspectives from cognitive psychology, linguistics, artificial intelligence, and education* (pp. 453–481). Hillsdale, NJ: Erlbaum.

Chi, M. T. H., & Bassok, M. (1989). Learning from examples via self-explanations. In L. B. Resnick (Eds.), *Knowing, learning, and instruction: Essays in honor of Robert Glaser* (pp. 251–282). Hillsdale, NJ: Erlbaum.

Dalbey, J., & Linn, M. C. (1985). The demands and requirements of computer programming: A review of the literature. *Journal of Educational Computing Research, 1,* 253–274.

De Corte, E., Verschaffel, L., & Schrooten, H. (1992). Cognitive effects of learning to program in Logo: A one-year study with sixth graders. In E. De Corte, M. Linn, H. Mandl, & L. Verschaffel (Eds.), *Computer-based learning environments and problem solving* (pp. 207–228). Berlin: Springer-Verlag.

Flavell, J. H. (1979). Metacognition and cognitive monitoring: A new area of cognitive developmental inquiry. *American Psychologist, 34,* 906–911.

Flavell, J. H., & Wellman, H. (1977). Meta memory. In R. Kail & J. Hagen (Eds.), *Perspectives on the development of memory and cognition.* Hillsdale, NJ: Erlbaum.

Gagné, E. D. (1985). *The cognitive psychology of school learning.* Boston: Little Brown.

Goetz, E., Alexander, A., & Ash, M. (1992). *Educational psychology: A classroom perspective.* New York: Macmillan.

Husic, F., Linn, M. C., & Sloane, K. D. (1989). Adapting instruction to the cognitive demands of learning to program. *Journal of Educational Psychology, 81,* 570–582.

Karmiloff-Smith, A. (1986). From metaprocesses to conscious access: Evidence from children's metalinguistic and repair data. *Cognition, 23*(2), 95–147.

Linn, M. C. (1985). The cognitive consequences of programming instruction in classrooms. *Educational Researcher, 14*(5), 14–16, 25–29.

Linn, M. C., & Clancy, M. J. (1992). The case for case studies of programming problems. *Communications of the ACM, 35*(3), 121–132.

Linn, M. C., & Dalbey, J. (1985). *The cognitive consequences of programming: Influence of instructional setting, student access, and student ability* (ACCCEL Report). Berkeley: University of California.

Mandinach, E., & Linn, M. C. (1986). The cognitive effects of computer learning environments. *Journal of Educational Computing Research, 2*(4), 411–427.

Oliver, R. (1990, July). *The contextual model: An alternative teaching model for introductory computer programming.* Paper presented at WCCE '90: Informatics at the Secondary Level Stream conference, Perth, Australia.

Oliver, R. (1993). Measuring hierarchical levels of programming knowledge. *Journal of Educational Computing Research, 9*(3), 299–312.

Oliver, R., & Malone, J. (1993). The influence of instruction and activity on the development of semantic programming knowledge. *Journal of Research on Computing in Education, 25*(4), 521–533.

Palumbo, D. B. (1990). Programming language/problem-solving research: A review of relevant issues. *Review of Educational Research, 60*(1), 65–89.

Palumbo, D. B., & Reed, W. M. (1991). The effect of BASIC programming language instruction on high school students' problem solving ability and computer anxiety. *Journal of Research on Computing in Education, 23*(3), 343–372.

Paris, S. G., Lipson, M. Y., & Wixson, K. K. (1983). Becoming a strategic reader. *Contemporary Educational Psychology, 8*, 293–316.

Piaget, J. (1978). *Success and Understanding.* Cambridge, MA: Harvard University Press.

Reed, W. M., & Palumbo, D. B. (1992), The effect of basic instruction on problem-solving skills over an extended period of time. *Journal of Educational Computing Research, 8*(3), 311–325.

Rumelhart, D. E. (1980). Schemata: The building blocks of cognition. In R. J. Spiro, B. C. Bruce, & W. F. Brewer (Eds.), *Theoretical issues in reading comprehension* (pp. 33–58). Hillsdale, NJ: Erlbaum.

Rumelhart, D. E., & Norman, D .A. (1978). Accretion, tuning, and restructuring: Three modes of learning. In J. W. Cotton & R. Klazky (Eds.), *Semantic factors in cognition* (pp. 37–53). Hillsdale, NJ: Erlbaum.

Salomon, G., & Globerson, T. (1987). Skill may not be enough: The role of mindfulness in learning and transfer. *International Journal of Educational Research, 11*, 623–637.

Schwartz, S., Perkins, D. N., Estey, G., Kruidenier, J., & Simmonds, R. (1989). A "metacourse" for BASIC: Assessing a new model for enhancing instruction. *Journal of Educational Computing Research, 5*(3), 263–297.

Shih, Y., & Alessi, S. (1993–1994). Mental models and transfer of learning in computer programming. *Journal of Research on Computing in Education, 26*(2), 154–175.

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

Shneiderman, B., & Mayer, R. (1979). Syntactic/semantic interactions in programmer behavior: a model and experimental results. *International Journal of Computer and Information Sciences, 8*(3), 219–238.

Shuell, T. J. (1990). Phases of meaningful learning. *Review of Educational Research, 60*(4), 531–547.

Snow, R. E., & Lohman, D. F. (1984). Towards a theory of cognitive aptitude for learning from instruction. *Journal of Educational Psychology, 76*, 347–376.

Soloway, E. (1985). From problems to programs via plans: The content and structure of knowledge for introductory LISP programming. *Journal of Educational Computing Research, 1*(2), 157–172.

Volet, S. E. (1991). Modeling and coaching of relevant metacognitive strategies for enhancing university students' learning. *Learning and Instruction, 1*, 319–336.

Volet, S. E., & Lund, C. P. (1994). Metacognitive instruction in introductory computer programming: A better predictor of achievement than traditional factors. *Journal of Educational Computing Research, 10*(4), 297–328.

Winograd, P., & Chou Hare, V. (1988). Direct instruction of reading comprehension strategies: The nature of teacher explanation. In C. E. Weinstein, E. T. Goetz, & P. A. Alexander (Eds.), *Learning and study strategies: Issues in assessment, instruction, and evaluation* (pp. 121–139). San Diego, CA: Academic Press.