In M. Petre & D. Budgen (Eds) Proc. Joint Conf. EASE & PPIG 2003 Pages 333-346

Applying Cognitive Load Theory to Computer Science Education

Dale Shaffer
Lander University
Greenwood, SC
USA

Wendy Doube
Monash University
Churchill, Victoria
Australia

Juhani Tuovinen
Charles Sturt University
Wagga Wagga, New South Wales
Australia

Abstract

Cognitive Load Theory provides a theoretical basis for understanding the learning process. It uses an information processing model to describe how the mind acquires and stores knowledge, and to provide an explanation for the limitations imposed by working memory.

This paper describes Cognitive Load Theory, discusses its application in a number of areas, and explores its potential uses in understanding and improving novice programming and computer science education. A number of research directions are suggested.

Introduction

Cognitive science deals with the mental processes behind learning, memory, and problem-solving. It identifies, among several methods, an information-processing approach to explain how the mind works. In the simplest form of this model, the mind is assumed to be divided into three portions, sensory, working, and long term memory (Cooper, 1998). An adaptation of the model is shown in Figure 1, but is incomplete as it omits all but the two most important senses for learning, sight and sound.

Memory

Sensory memory receives stimuli from the senses, including sight, sounds, smell, taste, and touch. It is short-lived, and if the mind is not able to identify and assign meaning to the input, the information is lost. For example, the sense of smell can detect a particularly inviting and unidentifiable fragrance, and a few moments later the fragrance is gone and is no longer able to be recalled in detail.

Long-term memory holds a permanent and massive body of knowledge and skills. Examples include how to ride a bicycle, multiplication facts, how to walk, and where we live.

Working memory is the portion of our mind that allows us to think both creatively and logically, and to solve problems. It provides our consciousness (Baddeley, 1993). Working memory is the interface between long-term memory and sensory memory. After being filtered through sensory memory and before being stored in long-term memory, knowledge must pass through working memory.

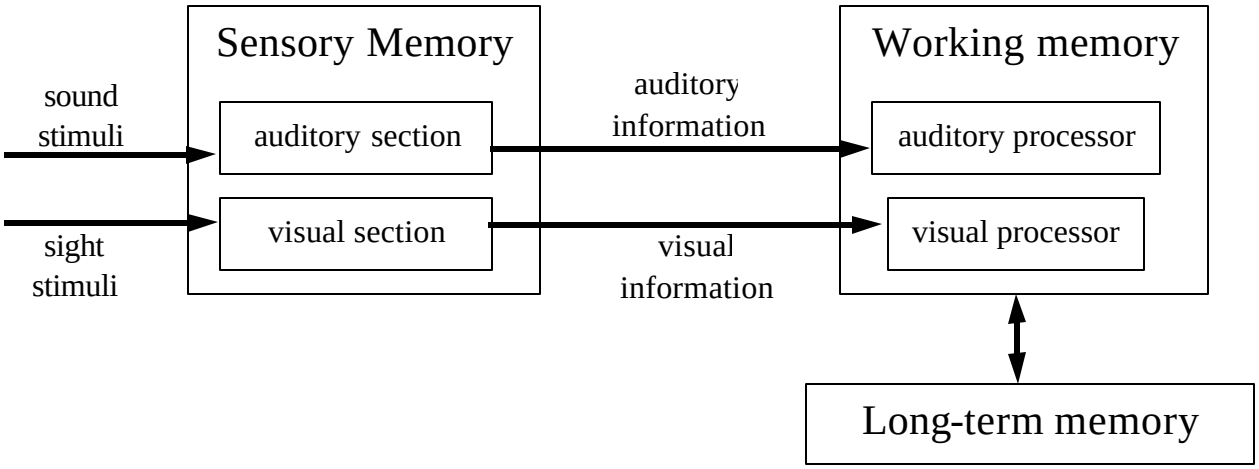

Figure 1 A Simple Model of Memory

Working memory, however, has two limitations. Firstly, it is short-lived. Secondly, it is only capable of processing up to approximately seven things at a time (Miller, 1956). If this capacity is exceeded, then some or possibly all of the information is lost. Cooper (1998) gives the following example of this phenomenon. Attempt to add the following in your head without the assistance of a calculator, pencil, or paper.

1) 46	2) 83,468,446
+ 37	93,849,937
	+ 58,493,900

Most individuals are capable of performing the first addition without difficulty. However, the second problem is almost impossible to solve mentally, even though the same process, addition, is involved. The capacity of working memory was exceeded. If one were to employ paper and pencil for the second problem, success would be likely since the paper formed an extension of working memory, allowing one to record intermediate values and to not overload working memory.

Learning

Learning can be defined as the "encoding (storing) of knowledge and/or skills into long-term memory in such a way that the knowledge and skills may be recalled and applied at a later time on demand" (Cooper, 1998). Introductory computer programming, like mathematics, requires *declarative learning* of abstract concepts and *procedural learning* acquired by practice. Procedural learning retrieves declarative knowledge into working memory, refines it, and strengthens cognitive structures. Declarative learning, in the field of computer science, can be thought of as skills acquisition, and procedural learning can be thought of as learning to execute those skills.

Visual knowledge appears to be encoded and processed differently from verbal knowledge (Pavio, 1990). Learning can be increased when the same content is presented simultaneously in verbal and visual representations in a way that facilitates associations between the two, especially if the verbal presentation is in an aural medium and the visual medium is graphical. This suggests that memory is partitioned into an auditory and a visual system and that working memory capacity can be expanded when both systems are employed (Cooper, 1998). Other partitions are possible, too.

During recall, knowledge is retrieved into working memory. Retrieval plays a major role in increasing the efficiency of the encoding and storage processes. With "rehearsal", or repeated and regular retrieval, recall processes gradually become "automated" and novices can progress towards expert status. Experts employ automated retrieval processes to recall information from highly organized structures in long term memory.

Cognitive load theory

Cognitive load theory is an instructional theory developed by psychologist John Sweller to describe "the learning process in terms of an information processing system involving long-term memory, which effectively stores all of our knowledge and skills on a more-or-less permanent basis, and working memory, which performs the intellectual tasks associated with consciousness" (Cooper, 1998). This theory, based on the above view of the mind (Sweller, Van Merriënboer, & Paas, 1998), provides a model of how the mind processes information based on two tenets:

- "Human working memory is limited; we can only keep in mind a few things at a time. This poses a fundamental constraint on human performance and learning capacity.
- Two mechanisms to circumvent the limits of working memory are:
    1) schema acquisition, which allows us to chunk information into meaningful units, and
    2) automation of procedural knowledge." (Wilson & Cole, 1996)

The first mechanism provides information about how the mind understands and processes information. For example, one typically chunks portions of a phone number to help remember it. The phone number 235 4827 is much easier to remember than 2 3 5 4 8 2 7.

The telephone example suggests that chunking does not need an underlying meaning associated with the elements that were chunked. One does not have a special meaning for 235, yet chunking can occur. However, if meaning can be identified and used to define the chunks, remembering is significantly enhanced. Consider memorizing the following chunks of information.

Vers esth atdo notte achme nnew andtou ching trut hsdon otdes erveto bere ad.

If Voltaire's quotation is chunked differently, we would have

Verses that do not teach men new and touching truths do not deserve to be read.

The process of chunking information into meaningful units is quite similar to the way a computer programmer would combine steps in a program into an abstraction (Shaffer, 2003). For example, the individual programming steps of finding the sum of a series of numbers, then dividing the sum by the number of numbers would be combined together (i.e. chunked). The abstraction has a convenient name, average and, for those individuals able to form the abstraction, only one of the approximately seven things that can be retained in working memory is accounted for.

Schema acquisition involves more than chunking. The information that is chunked is further processed and placed into a schema. These hierarchical networks of information in long-term memory are used to associate and store things in memory. For example, the schema in Figure 2 is hierarchical in that "Cars" is a higher-order concept when compared to "Traffic lights." However, some links between nodes are not hierarchical; "Speed kills" and "Speed limits", for example.

Schema formation is a very dynamic operation. Through building ever more complex schema by assimilating portions of lower-level schemas into higher level schemas, skills are developed (Sweller, Van Merriënboer, & Paas, 1998). Children learning to read, for example, have to build schemas for letters that allow them to classify an infinite variety of shapes into the limited number of characters in the alphabet (Sweller, Van Merriënboer, & Paas, 1998). This process continues throughout life as words are formed from this collection of squiggles and meaning is interpreted from words.

The second mechanism that circumvents working memory, automation of procedural knowledge, deals with skills acquisition. Once a particular skill is acquired, automatic processing can bypass working memory (Sweller, Van Merriënboer, & Paas, 1998). In other words, with enough practice, an activity can be carried out without conscious processing. For example, many individuals reach a stage in driving where their conscious, or working, memory is free for other activities.

This automation of procedural knowledge only occurs when one has acquired a skill. Sweller, Van Merriënboer, and Paas (1998) gives the example of solving the following algebraic equation for a.

$$\frac{a+b}{c} = d$$

Individuals who have acquired the needed algebraic skill will automatically know that one needs to multiply both sides of the equation by c, then subtract b. Some individuals, students studying elementary algebra for example, would need to use working memory in an effort to recall the appropriate rules.

Building problem-solving skills, be it in the domain of algebraic equations or computer programming, have at its roots automation of procedural knowledge. Once a particular problem-solving skill has been chunked and stored in long-term memory in a way that enables prompt retrieval, the skill can be considered as being learned.

Cognitive load is the load placed on the cognitive system by performing a specific task. It can be measured in several indirect ways, including

- self-reporting (assigning load values on a scale),
- measurement of physiological activities (heart rate, for example),
- learner performance following treatment, and
- learner performance when a second task is performed concurrently with the primary task (Sweller, Van Merriënboer, & Paas, 1998).

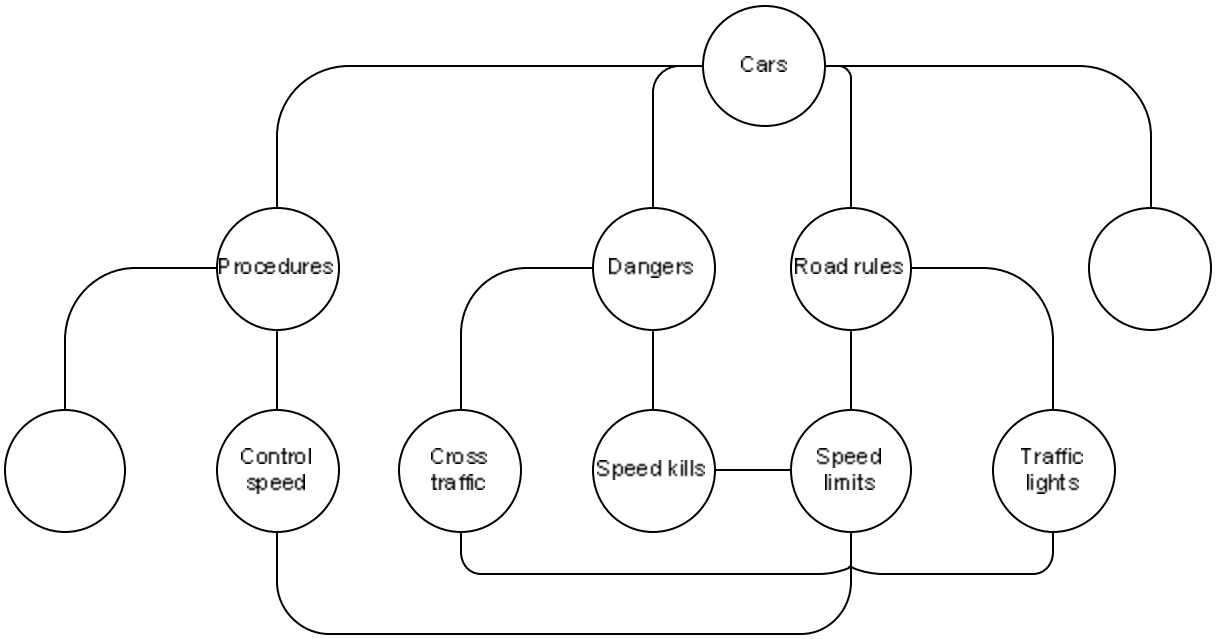

Figure 2 Portion of a schema in long-term memory (adapted from Cooper, 1998)

Cognitive load theory and learning

The extent to which curriculum materials impose a load on working memory varies widely. It "depends on the number of elements that must be processed simultaneously in working memory, and the number of elements that must be processed simultaneously, in turn, depends on the extent of element interactivity" (Sweller, Van Merriënboer, & Paas, 1998).

For example, compare a student who is learning multiplication facts to one learning how to solve equations in algebra. The student learning multiplication facts has quite a large number of facts to place into long term memory, but most of the facts are independent of each other. However, students solving equations in algebra have a larger load placed on working memory since they have to recognize (1) that they can divide both sides of the equation by the same value, (2) that this process does not change the equality, and (3) that it is more sensible to do this operation in some situations before adding a quantity to or subtracting a quantity from both sides. The student solving algebraic equations is dealing with facts with higher element interactivity which imposes a greater load on working memory.

Cognitive load and skills acquisition

Skills in computer programming can be classified as recurrent and non-recurrent (van Merrienboer, 1992). Recurrent skills exhibit minimal variation in differing problem situations, for example, selection of a basic language command. Non-recurrent skills vary from problem to problem, for example, structured decomposition. According to van Merrienboer, instructional approaches should take these differences into account.

Teaching in technical areas is often based on the formula of presenting a new topic, showing a few examples, and assigning practice exercises. Some simple adjustments to the presentation-examples-practice formula can accommodate cognitive load theory.

To learn recurrent skills, declarative knowledge of the abstract concepts on which the skills are based, as well as procedural knowledge of the procedures or rules necessary to perform the skills, should be simultaneously in working memory. Both the concepts and the procedures that use those concepts to solve problems should be presented together. One approach to this is to *partition* the information into small

segments to prevent overload, then *demonstrate* application of the concepts within each segment (van Merrienboer, 1992).

In contrast, non-recurrent skills can be taught by *presenting heuristic approaches and strategies*, independent of domain reference (Robertson, 2000). For example, consider the following repetition rule.

> In indeterminate looping,
>
> > use a WHILE loop when 0 iterations are possible, and
> >
> > use a REPEAT UNTIL loop when at least 1 iteration is necessary.

In this situation of high element interactivity, *presentation in the form of knowledge structures* can aid elaboration of existing schemas. For example, decomposition into a hierarchy of component goals or plans is a form of knowledge structure (Figure 3).

Examples provide explicit information to facilitate initial and correct schema formation (Cooper, 1998) and can initiate association of declarative and procedural knowledge. Following *presentatio*n, students can work through several *examples* with the teacher or examine multiple problems with integrated solutions, then immediately *practise* problems of the same type. By repeating this process with each type of problem, students can build schemas, and reinforce learning by retrieving and refining those schemas. Eventually

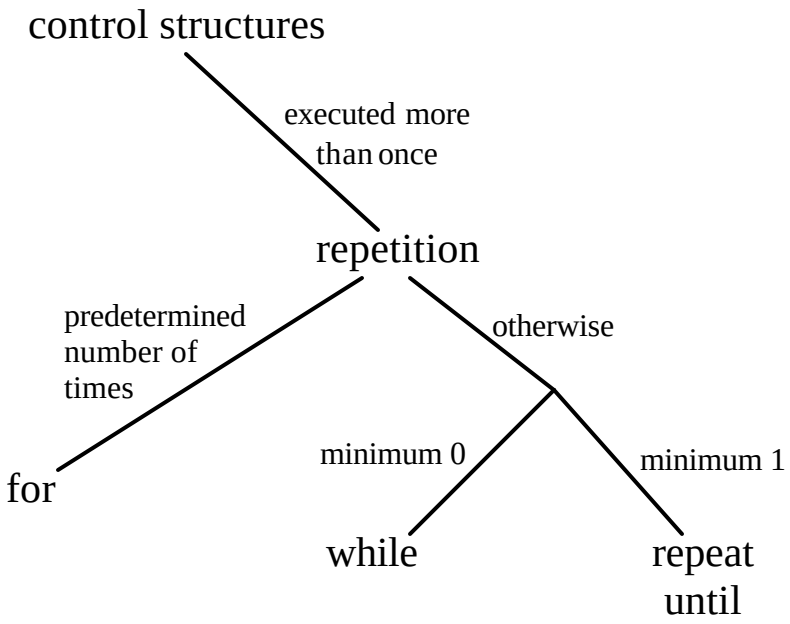

Figure 3 Hierarchy as a form of knowledge structure

practice with a mixed set of problems "elaborates" the schema even further by strengthening associations. A variation on this approach is to *provide partially-completed examples* for students to finish. Another is to *gradually reduce the amount of information and support for each task*, so that as learning increases with practice, less information is required.

One approach to practicing recurrent skills is to *decompose them into successively smaller components in the form of domain-specific, algorithmic steps*. Low element interactivity simplifies initial schema construction. Each component is practiced individually, and then parts of the whole are practiced until the entire skill can be integrated. For example arithmetic expressions are learned with presentation of entire expressions, then separate presentation and examples of firstly constants, then variables followed by operations. Next arithmetic expressions with constants and operations are practised. Finally, the three components of constants, variables and operations in complete arithmetic expressions are practised together.

It is generally agreed by computer science educators that introductory computer science courses tend to have high attrition rates, and that the lack of problem-solving skills is a contributing factor. This phenomena is discussed from various viewpoints in Beaubouef , Lucas, and Howatt (2001), Shultz (2000), Roberts, Kassianidou, and Irani (2002), Thweatt (1994), and in Wilson and Shrock (2001).

Problem-solving is usually approached using means-ends analysis. With experts who can rely on extensive schema acquisition and formation in long-term memory, this heuristic is applied by reducing differences between the current problem state and the goal state. This strategy seems especially effective in computer programming.

However, novices often apply it by working backwards from the goal to the initial state, then forward to the goal, resulting in a high cognitive load (Sweller, 1988). Cognitive load could be reduced if novices learning to apply means-ends analysis are given *problems that have no goal* (Owen & Sweller, 1985). Rather than focusing on the desired result, one is forced to focus on the initial state. Since much of computer programming is goal-oriented, this learning approach would appear to be difficult to implement, but might have significant value in an intermediate learning stage.

Another approach to reducing cognitive load is to increase working memory capacity by *utilizing verbal and visual channels*. However, redundant information can increase cognitive load by increasing the number of associations that have to be made between the different sources of information. (Chandler & Sweller, 1991).

One effective approach is to integrate text into an associated graphic. By placing the text directly within a diagram, the student is not forced to split attention between multiple sources. Modern software, in the form of algorithm animation and multimedia presentation packages where text and graphics can be added a piece at a time, gives much promise for this approach. In addition, algorithm animation and multimedia give students' power over the speed in which the material is presented, and therefore redundant learning elements can be reduced.

Bearing in mind that verbal material can be in the form of text and can place demands on visual resources if combined with graphics, the final approach to improve learning is by providing *multiple sensory input*. Partitioning information to come from visual and aural sources can further extend working memory (Pavio, 1990). Again multimedia and algorithm animation show promise in developing this approach but also show that working memory can be overloaded by multiple media, especially if it is superfluous to the learning content or asynchronous.

In summary, some of the possible strategies that could reduce cognitive load in introductory computer programming courses are:

- partitioning of information into small segments, them simultaneous presentation of concepts and procedures in demonstrations
- presentation of heuristic approaches and strategies, independent of domain reference
- organization of presentation of non-recurrent skills into knowledge structures
- presentation – repeated example and practice – broad practice
- broad practice using partially-completed examples
- gradual withdrawal of supporting information from practice tasks
- decomposition of practice tasks into small steps then gradual integration into a complete skill
- goal-free problem solving
- careful integration of text and graphics
- visual and aural sources for learning

Research applying the cognitive load theory to computer science education

There seems to be scant research in how the cognitive load theory and these suggested changes apply to computer science. Tuovinen and Sweller (1999) found that novices learning to apply a database to specified problem domains substantially benefited from the inclusion of worked examples when compared to students using discovery learning techniques if students had poor content schema. However, discovery learning was found to be at least as useful, and possibly even superior, for students who have well-developed content schema knowledge. Another study (Harvey & Anderson 1996), by measuring the transfer of knowledge learned in a Prolog language course to a second course using Lisp, found that transfer of cognitive skills between complex domains is dependent on the level of schema acquisition.

Two studies (Van Merriënboer & De Croock, 1992; Van Merriënboer, 1990) examined the effect that two approaches to learning computer programming had on learning outcomes. One group of students used an instructional strategy of working on modifying and completing existing programs while the second group focused on developing new programs. In both studies, students who were in the group completing existing programs demonstrated significantly improved program development skills, including developing new programs. There was considerable evidence that students in the completed programs group experienced an optimal cognitive load. This study should have profound effects in computer science education, but other than a push for both approaches in a lab setting, it seems to have little effect.

Perhaps the lack of clear support for completion tasks results from different levels in prior knowledge in programming classrooms. In contrast to van Merrienboer's findings, Nicholson and Fraser (1997) found no significant difference between the two approaches when students already familiar with programming learned a new language. In Van Merriënboer's studies, the novice students may have used the partially completed programs to construct new schemas whereas in Nicholson's study, the students may have had existing schemas to build on.

Although cognitive load theory appears to have received little interest in computer science education research, it has been applied with considerable success in the associated field of multimedia learning (Moreno, 2000; Mayer, 2001). At the very least, it would make sense to incorporate multimedia learning theories, based on cognitive load theory, into the ubiquitous fields of on-line learning and software visualization.

Conclusion: A research agenda

The connection between cognitive load theory and the challenges faced by novice computer science students has not been fully addressed. Research is needed to determine how the previously-mentioned potential changes can improve computer science education (Tuovinen, 2000). Possibilities for future research in this area include:

- using partially-completed examples of the form (presentation – repeated example and practice – broad practice)
- exploring ways to apply means-ends analysis without goals
- replicating the Van Merriënboer studies (1992; 1990) in university environments
- investigating the use of multimodal instruction in learning complex computer programming concepts
- investigating the identification of students' prior schematic computing content knowledge and the consequent adaptive teaching of further concepts and procedures using alternative approaches best suited for students with differential expertise and prior knowledge
- applying the cognitive load theory to the process of programming by professionals: Can language constructs be designed to be more cognitively efficient in terms of the programmers using them?
- Studying the effect of the use of abstraction on cognitive load.
- Identifying and classifying sources of high element interactivity in introductory computer programming and developing strategies to reduce it.

References

Baddeley, A. (1993). Working memory and conscious awareness. In A. F. Collins, S. E. Gathercole, M. A. Conway, & P. E. Morris (Eds.), Theories of Memory (pp. 11-28). Hillsdale: Lawrence Erlbaum.

Beaubouef , T., Lucas, R., & Howatt, J. (2001). The UNLOCK system: enhancing problem solving skills in CS-1 students. ACM SIGCSE Bulletin 33(2).

Chandler, P. & Sweller, J. (1991). Cognitive load theory and the format of instruction. Cognition and Instruction, 8:293-332.

Cooper, G. (1998). Research into Cognitive Load Theory and Instructional Design at UNSW, http://www.arts.unsw.edu.au/education/CLT_NET_Aug_97.HTML.

Harvey, L. & Anderson, J. (1996). Transfer of declarative knowledge in complex information-processing domains. Human-Computer Interaction, 11:69-96.

Mayer, R. (2001). Multimedia Learning. Cambridge University Press, Cambridge:.

Moreno, R. & Mayer, R. (2000) Meaningful design for meaningful learning: applying cognitive theory to multimedia explanations. Proceedings Ed-Media 2000, Montreal, Canada.

Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. The Psychological Review, 63:81-97.

Nicholson, A & Fraser, K. (1997). Methodologies for teaching new programming languages: a case study for teaching LISP. Proceedings 2nd ACSE Conference, Melbourne, Australia.

Owen, E. & Sweller, J. (1985). What do students learn while solving mathematics problems. Journal of Educational Psychology, 77:272-284.

Pavio, A. (1990). Mental Representations: A Dual Coding Approach. New York: Oxford University Press.

Shultz, G. (2000). Using the "boxes" support software in teaching CS1. The Journal of Computing in Small Colleges 16(1):133-141.

Roberts, E., Kassianidou, M., & Irani, L. (2002). Encouraging women in computer science. ACM SIGCSE Bulletin, 34(2):84-88.

Robertson, L.A. (2000) Simple Program Design: A step by step approach Nelson Thomson Learning, Melbourne.

Shaffer, D. (2003). Cohesion, coupling, and abstraction. In H. Bigoli (Ed.) Encyclopedia of Information Systems (pp. 127-139). New York: Academic Press.

Sweller, J. (1988). Cognitive load during problem-solving: Effects on learning. Cognitive Science, 12(2):257-285.

Sweller, J., Van Merriënboer, J., & Paas, F. (1998). Cognitive architecture and instructional design. Educational Psychology Review 10(3):251-294.

Thweatt , M. (1994). CSI closed lab vs. open lab experiment. ACM SIGCSE Bulletin, 26(1):80-82.

Tuovinen, J. (2000). Optimising student cognitive load in computer education. Proceedings of the Fourth Annual Australasian Computing Education Conference, Melbourne: ACM. pp. 235-241.

Tuovinen, J. & Sweller, J. (1999). A comparison of cognitive load associated with discovery learning and worked examples. Journal of Educational Psychology, 91(2):334-341.

Van Merriënboer, J. (1990). Strategies for programming instruction in high schools: Program completion vs. program generation. Journal of Educational Computing Research, 6:265-285.

Van Merriënboer, J. & De Croock, M. (1992). Strategies for computer-based programming instruction: Program Completion vs. Program Generation. Journal of Educational Computing Research, 8:365-394.

Wilson, B. & Cole, P. (1996). Cognitive teaching models. In D. Jonassen (Ed.) Handbook of Research in Instructional Technology. New York: Scholastic Press.

Wilson, B. & Shrock, S. (2001). Contributing to success in an introductory computer science course: a study of twelve factors. ACM SIGCSE Bulletin, 33(1):184-188.