# The CLOZE Procedure and the Learning of Programming

Stuart Garner

School of MIS

Edith Cowan University
100 Joondalup Drive, Joondalup,
Western Australia
s.garner@ecu.edu.au

## ABSTRACT

Many students find great difficulty with the learning of programming. This paper discusses some of those difficulties and ways in which the cognitive load that students experience can be reduced. One way of reducing the load is to make use of the cloze method that is used in English comprehension testing. When used with the learning of programming, the cloze method requires students to fill-in and complete a part-complete solution to a programming problem that they have been given. A code restructuring tool, CORT, has been built by the author to support this method and utilised by students in an introductory programming course. Initial results suggest that students were encouraged and motivated whilst using CORT and that the time taken to complete problems and the help required were less than for students who did not use the tool.

## Keywords

Learning; programming; cloze; templates; schemata.

## 1. INTRODUCTION

It is widely accepted in the fields of computer science and information systems that the learning of computer programming is not an easy process for students. Many researchers have discussed this situation. For example, Deek et al [1] point out that students face several challenges, including: deficiencies in problem solving strategies and tactical knowledge; misconceptions about syntax, semantics, and pragmatics of language constructs; and ineffective pedagogy of programming instruction.

A large number of studies into the learning of programming conclude that novice programmers may know the syntax and semantics of individual statements but they do not know how to combine these features into valid programs. Winslow [2] also suggests that even when students know how to solve a problem by hand, they do not know how to translate it into a valid program.

The above should come as no surprise to educators as most recognise that there is a very high cognitive load imposed on students who are attempting to learn to program. Kennedy [3] cites Sweller [4] who states that learning to program demands considerable cognitive resources requiring skills of analysis and design, and that should the cognitive load be excessive then any learning will be inhibited. Kennedy also indicates that any methodology that can reduce this cognitive load will enable the student to produce more efficient solutions to problems in a shorter time frame.

This paper will look at how the cognitive load that students experience when learning to program can be reduced by the utilisation of the cloze method. This method requires students to try and fill in the gaps within part-complete solutions that they are given to programming problems. By the judicious choice of programs, students can be encouraged to learn stereotypical programming plans or schema which are the building blocks of computer programs. A tool has been produced by the author to support this method of learning and this is discussed together with some of the results of field trials.

## 2. Cognitive Load and Programming

It is well known that some subjects are inherently difficult to learn for many students and they often complain about the high cognitive load of their learning experience. Cognitive load theory has been succinctly described by Maguire [5]:

*All conscious processing occurs in working memory which is limited to a few elements. On the other hand, long term memory is almost boundless and relatively permanent. Once a problem solving procedure is learned it is incorporated into a construct called a schema which is stored in long term memory. Schemas are recalled from memory as a single chunk. They allow different types of problems to be classified according to problem type and method of solution. A human becomes an expert in a particular domain once a particular schema has been automated.*

Cognitive load can be divided into two areas, these being intrinsic and extraneous. The intrinsic cognitive load is that imposed by the difficulty of the knowledge to be learned and this is fixed. The extraneous cognitive load is dependent on the instructional and learning design. This is not fixed and a good design can reduce this load.

Programming is known to have a high intrinsic cognitive load and it is therefore necessary for instructors to attempt to ensure that the extraneous cognitive load is as low as possible.

## 3. The Cloze Method

The term "cloze" is derived from "closure", a Gestalt psychology term referring to the human tendency to complete a familiar but not quite finished pattern [6]. It was first used in teaching and learning by Taylor [7] who studied the effectiveness of cloze as an instrument for assessing the relative readability of written materials for school children. Cloze is now often used to measure comprehension in English readability [8, 9].

Cloze has also been used in the teaching and learning of programming as a way of measuring student understanding of programs [10, 11]. Such program comprehension tests are constructed by replacing some of the "words" or tokens by blanks and requiring students to fill in the blanks during a test. The use of the cloze procedure in testing was found to correlate well with conventional comprehension, question - answer, type quizzes and is also much easier to create and administer, see for example [6].

Other researchers have experimented with the testing of program comprehension by omitting complete lines of code from programs and requiring students to fill in those lines. Norcio [12, 13, 14, 15] found that students were more likely to supply correct statements if they had been omitted within a logic segment rather than from the beginning of a segment. This is consistent with the chunking hypothesis of Miller [17] that specifies that the first element of a chunk provides the key to the contents of the entire unit. Ehrlich [16] looked at the differences between experts and novices in filling in missing lines within various programming plans and, as expected, found that the experts filled in the lines correctly taking into account the surrounding plan whereas novices had more difficulty.

Cloze has also been utilised in an educational program diagnosis system [18] for the Internet. In the system, part complete programming exercises are presented to students who are then expected to try and key in correct lines of code. The aim of the system is to then diagnose a solution and give student feedback. The success of this system is as yet unclear.

In the various experiments in program comprehension using the cloze procedure, the students had to fill in the lines of code without being given a selection of lines to choose from. In some work done by Edwards [19] in an area unrelated to programming, students were expected to create an essay using a file of statements, only some of which were relevant to the topic. The students were expected to copy and paste only the statements which they believed to be relevant and then to link them with their own text and it was suggested that learners would consolidate their understanding of the topics by having to actively evaluate all possible statements. The file of statements was acting as a scaffold to student learning.

Edward's method of helping students to learn essay writing has been developed further by the author as a way of scaffolding student learning of programming. An incomplete solution to a programming problem can be given to a student together with a choice of statements that might be used in the solution. The student can then study the incomplete solution and the choice of statements and decide which statements to use and where to put them. A computerised tool has been developed that uses this idea, making the mechanics of placing the statements into the incomplete solution very straightforward. This eliminates typing errors, reduces syntax errors and also the cognitive load on students.

## 4. Programming Plans and Schema

Expert programmers have the necessary cognitive schemata to easily perform familiar programming tasks and also to interpret unfamiliar situations in terms of their generalised knowledge [20]. In the domain of programming these specific schemata are known as programming plans and they are learned programming language templates, or stereotyped sequences of computer instructions, that form a hierarchy of generalised knowledge. After most introductory programming course, students have a serious lack of programming language templates or programming plans [21].

An example of a program with its plans identified is shown in figure 1. The challenge for programming instructors is how to facilitate student learning so that students can learn such plans and then apply them to new problem situations.

In the figure, three plans have been identified, these being: a running total loop plan; a counter variable plan; and a skip guard plan. Such plans are second nature to experienced programmers and can be extracted and applied to other problems almost automatically. Such plans have been categorised to be high-level, medium level and low level, examples being respectively: a general input - process - output plan; a running total loop plan as in the above; and a statement to print the value of a variable. It is suggested that, within the programming domain, programming plans provide a hierarchy of increasingly context dependent strategies that may guide a process of "templating" in the creation of solutions to posed problems [20].

Such patterns, plans or templates that exist within computer programs need to be somehow learned by students who are learning to program in order to develop appropriate programming schemata. Self-discovery, by attempting to solve problems, is often problematic for sudents. Work done by Van Merrienboer and Krammer [22] suggests that a program completion strategy is superior to a program generation strategy when attempting to learn such templates. A completion strategy involves students having to complete a solution to a problem whereas a generation strategy requires students having to develop a solution "from

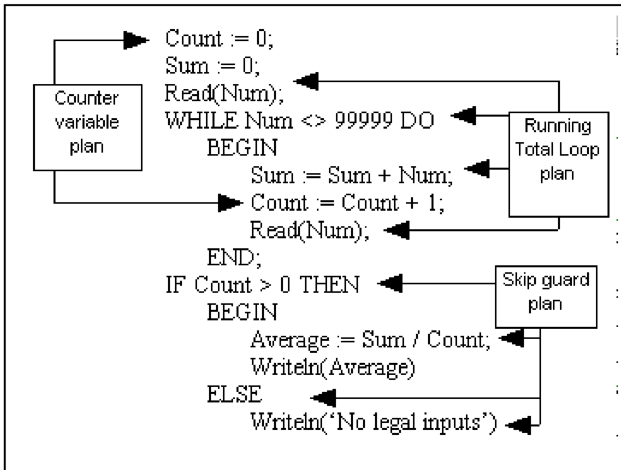


**Figure 1: Example of Programming Plans**

scratch".

It is proposed that students can be helped to learn such plans by utilising the cloze method in conjunction with the program completion strategy.

## 5. CORT: a Tool to support the Cloze Method

CORT (Code restructuring tool) has been designed to provide learner support for the use of the cloze procedure in conjunction with a program completion strategy to help students learn to program. The tool allows part-complete solutions to programming problems to be displayed in one window and possible lines of code to be inserted into the solution within another window. The interface is shown in figure 2.

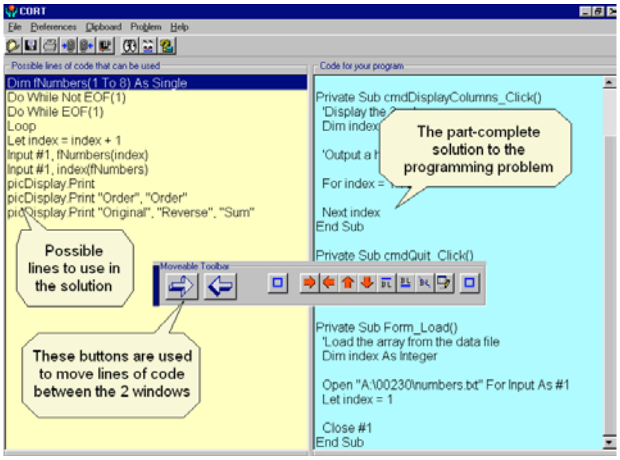


**Figure 2:**

The tool is currently being used in an introductory programming unit that uses Visual BASIC as the programming language. It can however be used with any programming language with the constraint that a Wintel machine is necessary. The use of CORT by a student is as follows:

A student firstly loads a CORT file, the two windows displaying a part-complete solution to a problem together with possible lines that can be used in that solution. The student can also view the problem statement and the solution interface by clicking on appropriate buttons on the fixed toolbar.

The student then moves certain lines of code from the left hand window to the right hand window in an attempt to complete the solution. Lines can be moved up or down, and indented or out-dented in the right hand window. Also, lines can be moved back to the left hand window. Some of the programming problems have more lines of code in the left hand window than are necessary to complete the solution, some of those lines being incorrect. If necessary, the student can invoke a simple editor to amend, add or delete lines of code.

When the student has "completed" the solution, they click on the appropriate toolbar button to copy the contents of the right hand window to the Windows clipboard. The student then invokes the programming language's IDE (integrated development environment) and loads into it a file that contains the graphical user interface for the correct solution. This file has been created by the instructional designer and contains, in effect, the solution to the problem without the lines of code.

The student then pastes the contents of the Windows Clipboard into the Visual BASIC editor and tests the program to determine if it works correctly. Use is made of the trace and de-bugging facilities of the IDE, these facilities providing an insight to the workings of the notional machine. If the student finds a problem with the working of the program, they can return to CORT and make the changes to the code there. They would then repeatedly amend and test the code until they had a working program.

## 6. Evaluation of CORT

CORT was used in over a period of one semester in an introductory programming unit at a university in Western Australia. Students had to enroll in one of four computer laboratories and they selected the particular laboratory at the time of enrollment. In the investigation, two of the laboratories were chosen to use the CORT program and the other two laboratories to have "conventional" programming exercises without CORT. In order to reduce possible bias, at the time of enrollment the students did not know whether they would be in the CORT or non-CORT group. Of the students who completed the unit, the number in the CORT group was 26 and the number in the non-CORT group was 27. The problems that the CORT and non-CORT students had to solve were of a similar nature and degree of difficulty.

The students in both the CORT and non-CORT groups had to fill-in journals after they had completed each problem. These journals were analysed and at the end of the semester as was the final closed-book examination. The results of the closed book examination revealed that there was no difference in the performances between the CORT and non-CORT groups. However, the student journals revealed that the CORT students took significantly less time than the non-CORT students to complete their programming exercises and that they also required significantly less help. These results are shown in figures 3 and 4 respectively.

Interviews were also carried out with some of the students and some of the comments made by students are as follows:

- With CORT it was good as it enabled me to finish something and therefore I prefer to use it. Getting programs working makes you feel better.
- If I was just asked to study code, then I would not do it properly, so CORT really helps.
- Very happy with CORT because lines are there to help. About the right amount of help is provided. If I did not use CORT then I would find it very difficult to know where to start.

## 7. Conclusions

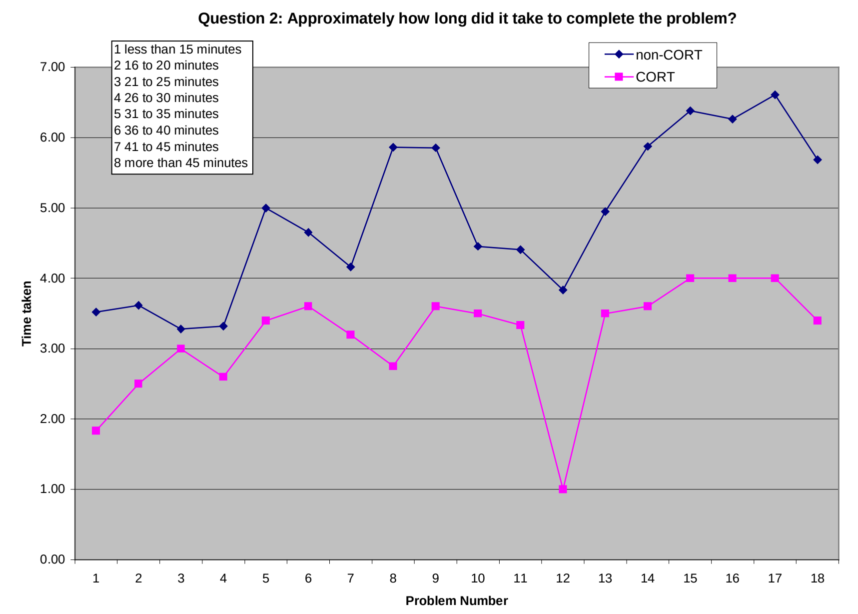


**Figure 3:**

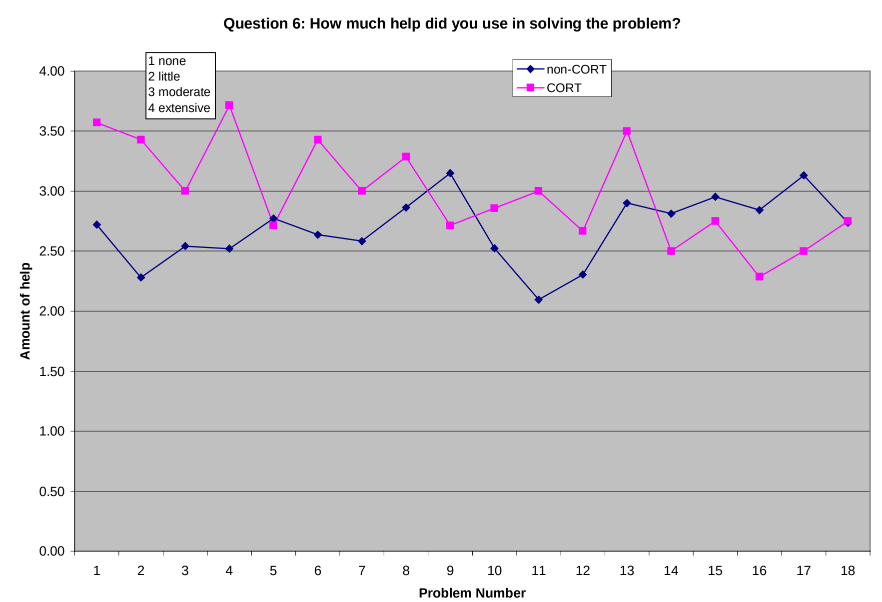


**Figure 4:**

This paper has discussed the use of a tool to support the cloze procedure in conjunction with the learning of introductory programming. The aim has been to reduce the cognitive load that students are under when learning programming and initial results indicate a degree of success.

Although there is no difference in the outcomes between CORT and non-CORT students, it was seen that the CORT students took less time to complete their work and that they required less help.

It could therefore be argued that if the CORT students spent more time solving problems, such that their total time matched that of the non-CORT students, then they would most likely be more successful.

The interview data suggested that students were very comfortable using CORT and they believed themselves that it was a very useful aid for learning programming.

## 8. REFERENCES


[1] Deek, F.P., McHugh, J.A. and Hiltz, S.R. Methodology and Technology for Learning Programming. The Journal of Systems & Information Technology, 4 (1). 25-37.

[2] Winslow, L. Programming Pedagogy -- A Psychological Overview. *SIGCSE Bulletin*, *28* (3).

[3] Kennedy, D.M. and McNaught, C. Design Elements for Interactive Media. *Australian Journal of Educational Technology*, *13* (1). 1-22.

[4] Sweller, J. and Chandler, P. Evidence for cognitive load theory. *Cognition and Instruction*, *8*. 351-362.

[5] Maguire, M. Measuring cognitive load using computational models, Australian Association for Research in Education, 1996.

[6] Cook, C., Bregar, W. and Foote, D. A Preliminary Investigation of the use of the Cloze Procedure as a Measure of program Understanding. *Information Processing & Management*, *20* (1-2). 199-208.

[7] Taylor, W.L. Cloze procedure: a new tool for measuring readability. *Journalism Quartely* (30). 415-433.

[8] Klare, G.R. Assessing Readability. *Reading research quarterly* (10). 63-102.

[9] Brown, J.D. Do Cloze Tests Work? Or, Is It Just An Illusion? *Second Language Studies*, *21* (1, Fall 2003).

[10] Hall, W.E. and Zweben, S.H. The Cloze Procedure and Software Comprehensibility Measurement. *IEEE Transactions on Software Engineering, May 1986*. 608-623.

[11] Thomas, M. and Zweben, S. The Effects of Program-Dependent and Program-Independent Deletions on Software Cloze Tests. *Empirical Studies of Programmers*. 138-152.

[12] Norcio, A.F., Human Memory Processes for Comprehending Computer Programs. in *Cybernetics and Society*, (Cambridge, Massachusetts, 1980), 974-977.

[13] Norcio, A.F., Comprehension Aids for Computer Programs. in *American Psychological Association Annual Meeting*, (Montreal, 1980), 1-6.

[14] Norcio, A.F., Chunking and Understanding Computer Programs. in *Human-Machine Systems Symposium*, (Boston, USA, 1981), 1-6.

[15] Norcio, A.F., Indentation, Documentation and Programmer Comprehension. in *Human Factors in Computer Systems*, (Gaithersburg, Maryland, 1982), 118-120.

[16] Ehrlich, K. and Soloway, E. An Empirical Investigation of the Tacit Plan Knowledge in Programming. in Thomas, J. and Schneider, M.L. eds. *Human Factors in Compter Systems*, Ablex, Norwood, New Jersey, 1984, 113-133.

[17] Miller, G.A. The Magical Number Seven, Plus or Minus Two: Some Limits on our Capacity to Process Information. *Psychological Review* (63). 81-97.

[18] Kaijiri, K. Program Diagnosis System using World Wide Web, 1998.

[19] Edward, N., Development of a cost effective computer assisted learning (CAL) package to facilitate conceptual understanding. in *CAL97*, (University of Exeter, UK, 1997).

[20] Van Merrienboer, J.J.G. and Paas, F. Automation and Schema Acquisition in learning elementary computer programming. *Computers in Human Behavior* (6). 273-289.

[21] Dalby, J. and Linn, M.C. The demands and requirements of computer programming: A literature review. *Journal of Educational Computing Research* (1). 253-274.

[22] Van Merrienboer, J.J.G. and Krammer, H.P. The "Completion Strategy" in Programming Instruction: Theoretical and Empirical Support. in Dijkstra, S., Van Hout-Wolters, B. and van der Sijde, P. eds. *Research on Instruction*, Educational Technology Publications, Englewood Cliffs, New Jersey, 1989, 45-61.