An Exploration of How a Technology-Facilitated Part-Complete Solution Method Supports the Learning of Computer Programming

Stuart Garner
Edith Cowan University, Joondalup, Australia

s.garner@ecu.edu.au

Abstract

This paper reports on the findings from a qualitative research study into the use of a technology-facilitated part-complete solution method (PCSM) that was used to support the learning of computer programming. The use of part-complete solutions to programming problems is one way in which the cognitive load that students experience during learning can be reduced.

A code restructuring tool, CORT, was built to support the PCSM and an inquiry into its effectiveness took place over a period of 14 weeks at an Australian university. Results suggest that: the system provided strong scaffolding for student learning; students engaged well with the system and generally used a thoughtful and considered cognitive strategy; and the highest level of support was for student semantic difficulties, although there was also strong support for algorithmic and structural difficulties.

Keywords: learning, programming, cognition.

Introduction

Learning to write computer programs is not easy (e.g., du Boulay, 1986; Scholtz & Wiedenbeck, 1992) and this is reflected in the low levels of achievement experienced by many students in first programming courses. For example, Perkins, Schwartz & Simmons (1988, p.155) state that: "Students with a semester or more of instruction often display remarkable naiveté about the language that they have been studying and often prove unable to manage dismayingly simple programming problems". Also, King, Feltham & Nucifora (1994, p.18) state that: "Even after two years of study, many students had only a rudimentary understanding of programming".

Some research has taken place into the use of part-complete solutions as a way of reducing the cognitive load on students in order to help them learn programming more effectively (e.g., van Merrienboer, 1990; van Merrienboer & De Croock, 1992). These studies demonstrated the potential of such a part-complete solution method (PCSM), but its success was never realised due to the absence of suitable electronic tools to support the process.

A software tool named CORT (Code Restructuring Tool) has been built by the author to support this completion method and this paper reports on the findings from a qualitative research

study concerning how the use of the PCSM and CORT helped support student learning.

CORT: A Tool to Support the PCSM

CORT has been described in other papers (e.g., Garner, 2003, 2005). The interface comprises two windows as shown in Figure 1. The right-hand window contains the part-complete solution to a given programming problem, and the left-hand window contains possible lines of code that can be used to complete the program. A CORT problem may have more lines in the left-hand window than are necessary, some lines acting as distracters to force students to think more carefully about the lines to choose.

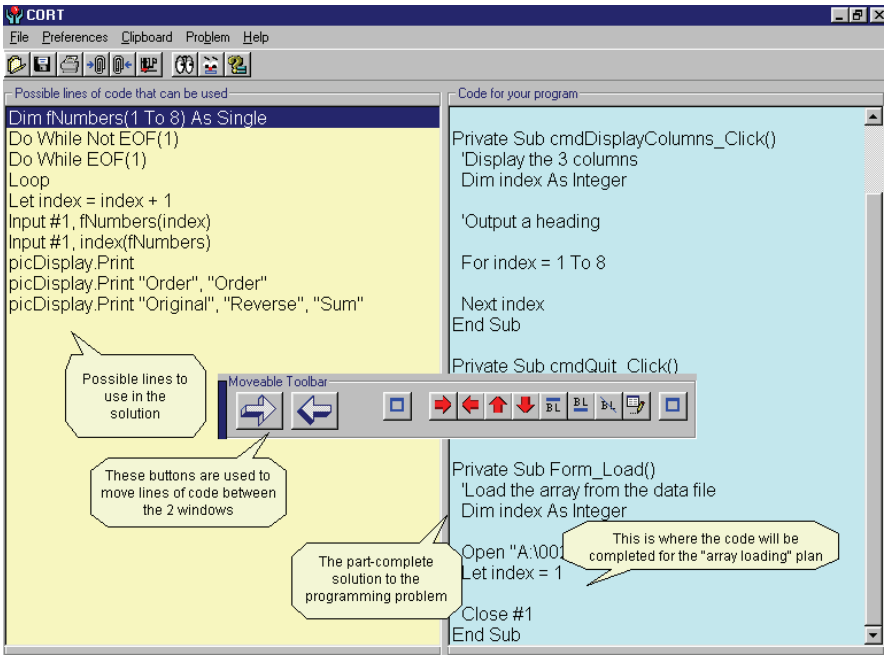


Figure 1: COde Restructuring Tool (CORT)

Lines can be moved between the windows by clicking on the large arrows on the toolbar. Other buttons on the toolbar can be used to rearrange lines in the right-hand window. When a student wishes to test the code, the contents of the right-hand window are copied to the windows clip-board and then pasted into a program development environment which, in this example, is Visual BASIC. The program can then be executed and, if necessary, amended back in the CORT pro-gram.

The set of possible lines of code that is given to a learner to be used in the completion of a part-complete program can be varied. This can be done by providing one of the following methods:

Method 1. All of the lines of code that are missing from the program are provided as options.

Method 2. All of the lines of code that are missing from the program, together with some extra lines of code that are not needed to complete the program, are provided. These extra lines act as "distracters".

Method 3. Some of the lines of code that are missing from the program might be provided, however some other missing lines must be keyed-in by the learner.

The important variable that affects which of the above methods is used for a given problem is the degree of difficulty of that problem. For example, if a problem was relatively simple then method 2 might be used, whereas method 1 might be used with a more difficult problem.

Research Design

An inquiry took place to investigate the ways in which the part-complete solution method (PCSM) within the CORT system supported the learning process.

Observation and participation by the researcher were used as a major method of collecting data and an action research methodology was utilised for the study. A quasi-experimental design framework was used within the action research methodology. The design was deemed to be quasi-experimental as it was not possible to achieve "randomisation" of exposures which is essential if true experimentation is to take place (Cohen & Manion, 1994).

The investigation took place over a period of one semester at an Australian university, a semester being 14 teaching weeks. The unit of study was in introductory programming and was for students within a school of Management Information Systems. Students are expected to gain fundamental programming knowledge in this unit including the three basic control structures, built-in functions, user-defined functions, event and general procedures, text file processing, and array processing.

A set of 17 programming problems was carefully designed in order to cover the objectives of the unit syllabus. The problems were a mixture of the three methods described earlier and were given to the students over a period of ten weeks. Eight students were observed over the period with the aim of determining patterns and trends in the usage of CORT and the part-complete solution method together with evidence of reflection and higher order thinking. The author acted as the observer and prompted students to try and make their thinking explicit when certain courses of action were undertaken.

Analysis of Results

Levels of Cognitive Strategy with the CORT System

The forms of learner cognitive strategy were tabulated and levels were identified from the forms of student activities observed. This determination of levels provided a discrete set for analyses and inquiry.

From the observations of student activities that took place by the author, five distinct levels of cognitive strategy were identified with respect to CORT. These ranged from the lowest level of cognitive strategy where a student demonstrated no planning and randomly moved lines from the left hand window into a part-complete solution, through to the highest level where a student demonstrated thorough planning and testing of a problem solution. These levels were classified by the researcher and are described in Table 1.

Table 1: Classification of Levels of Cognitive Strategy

Level of Cognitive Strategy	Solution Method
1	Unplanned and random. For example a student: • Does not read through the part-complete solution. • Chooses a line of code at random from the set of lines in the left-hand window and then moves it to a random position in the right-hand window. • Tests their code in an unplanned and random manner. • Does not trace code in the Visual BASIC debugger.
2	A low level of consideration in their approach. For example a student: • Partially reads a part-complete solution. • Chooses a line at random and then moves the line with some thought to a position in the right-hand window. • Identifies a subset of lines in the left-hand window, chooses a line to move from that subset and moves it to a random position in the right-hand window. • Demonstrates little planning in their testing. • Does not trace code in the Visual BASIC debugger.
3	Some levels of consideration in their approach. For example a student: • Thoroughly reads a part-complete solution. • Identifies a subset of lines in the left-hand window and then chooses a line to move from that subset. • Moves the line with some thought to a position in the right-hand window. However this is done at a micro level such that they move the line to be adjacent to lines which look similar, e.g. the line n = 1 is in the right hand window and therefore they move n = n + 1 to be close to this line • Tests and traces their program several times.
4	High levels of consideration and some evidence of strategy. For example a student: • Thoroughly reads and studies a part-complete solution. • Carefully selects lines of code. • Carefully and thoughtfully places the lines into the part-complete solution. • If a part-complete solution has more than one procedure, they work on one procedure at a time in the right hand window. Shows some evidence of testing the part-complete solution in a strategic manner, making extensive use of the Visual BASIC debugger.
5	Deliberate approach. For example a student: • Thoroughly reads and studies a part-complete solution and the lines of code in the left-hand window. • Demonstrates initial planning. • Carefully selects lines of code and thoughtfully places the lines into the part-complete solution. • If a part-complete solution has more than one procedure, they work on one procedure at the time in the right hand window. They might thoroughly test that procedure that they have completed the code for before they work on completing the code for other procedures. Tests the part-complete solution at appropriate points to check their hypotheses about the lines and the way the program should behave.

Support Types Identified and Scaffolded by the CORT System

The students were observed to use CORT in a number of different ways in response to the difficulties they experienced while attempting the programming problems. The use of CORT helped scaffold the students to various degrees and this section describes a classification of support types

and an explanation of how estimates were made of the scaffolding provided by CORT. The types of support provided by CORT were categorised as syntactical, semantic, structural, and algorithmic. These categories are standard forms within programming environments (e.g., Soloway, 1986; Winslow, 1996). Examples of the types are shown in Table 2.

Table 2: CORT Support Types

Support Type	Example
Syntax	A choice of two lines is available: Let txtName.ForeColor = vbRed Let txtName.ForeColor = Red A student initially chooses the second line which is syntactically incorrect and uses CORT to identify the error.
Semantic	A choice of two lines is available to place a string literal into a variable. Let personName = "Bill" Let personName = Bill A student initially chooses the second line which is semantically incorrect and uses CORT to identify the error.
Structural	**Example 1**: A student places variable declarations statements (DIM statements) in incorrect positions and uses CORT to identify the errors. **Example 2**: A student initially places lines of code in between (outside of) procedures and uses CORT to identify the errors.
Algorithmic	In solving an algorithm to determine the average of a set of numbers, a line of code is placed in the wrong position. For example: Do While Not EOF(1) Input #1, number Let total = total + number Loop Let count = count + 1 Let average = total / count In the above, the incrementing of the variable **count** should be within the loop. The CORT system helps the student identify the error.

In addition to determining the types of support that the CORT system provided for students for the problems that they attempted, it was important to determine the degree of assistance that CORT offered. Table 3 shows this classification of the scaffolding levels provided by CORT and their meanings.

Table 3: Classification of Levels of Scaffolding

Scaffolding Level	Meaning
1	The CORT system provided little help in solving the problem, but did help identify the errors.
2	The CORT system provided some help in solving the problem.
3	The CORT system provided a lot of help in solving the problem.

For each of the 17 CORT problems, the support type, cognitive strategy and level of scaffolding provided were recorded for each of the observed students.

Analysis of Summary Data

The collected data were summarised in a series of tables. This section presents those tables and discusses the trends that emerged from the data.

Analysis of Data by Student

Table 4 shows a summary of the learning support data for each of the eight students that were observed and includes the support types, levels of cognitive strategy and levels of scaffolding.

Table 4: Summary of Learning Supports, Levels of Cognitive Strategy and Levels of Scaffolding for Each Student

Stud. ID	No. of Problems Observed	=========Instances of Support Types=========								Level of Cognitive Strategy (max 5)		Level of Scaffolding (max 3)	
		Syntax		Semantic		Structural		Algorithmic		Median	Average	Median	Average
		No.	%	No.	%	No.	%	No.	%				
A	9	2	15	6	46	2	15	3	23	3	3.6	3	2.9
B	15	2	10	8	38	2	10	9	43	4	4.0	3	2.8
C	3			2	50	1	25	1	25	3	3.7	3	3.0
D	4	1	13	3	38	4	50			3	2.8	2.5	2.5
E	4			3	43	3	43	1	14	2.5	2.8	2	2.3
F	4	1	14	3	43	1	14	2	29	4	4.0	3	2.8
G	2			1	25	1	25	2	50	3	3.0	2.5	2.5
H	3	2	25	2	25	2	25	2	25	3	2.7	2	2.3
Total No. of Support Instances		8		28		16		20					
Overall % Support		11		39		22		28					
Averages										3.2	3.3	2.6	2.6

CORT scaffolded with an overall average of 2.6 (range 1 - 3) demonstrating that it provided considerable help for students. CORT supported a level of cognitive strategy of 3.3 (range 1 - 5). This revealed that students were generally engaged with CORT and that they nearly always applied some consideration in their approaches to the tasks that they attempted.

The overall levels of CORT's four support types were 11%, 39%, 22%, and 28% for syntax, semantics, structure and algorithms respectively. It was expected that CORT would provide a low level of support for syntax errors as methods 1 and 2 do not require students to key in lines of code. Because of this, the possibilities of students being confronted with syntax problems is relatively low for the CORT system. The majority of difficulties that the students had, and for which CORT provided support, were semantic. The probable reason for this is that most students attempted problems within CORT, made little use of the textbook, and made use of CORT to scaffold them with respect to any semantic difficulties that they were confronted with. If they had not used CORT to try and solve their problems then they would have been forced to use other resources, such as the textbook, in order to determine the meaning of various programming statements. By using the CORT system, students were usually able to determine the meaning and semantics of statements by experimentation and consideration of the feedback that the CORT sys-

tem provided them with. The levels of success achieved by students through their use of CORT confirmed the support CORT provided.

CORT also provided high levels of support for structural and algorithmic difficulties that students encountered. This is an important finding as there are generally few supports for these two areas when students learn to program using traditional techniques.

Analysis of Data by Problem Number

A summary of the data tabulated by problem number is shown in Table 5.

Table 5: Summary of Learning Supports, Levels of Cognitive Strategy and Levels of Scaffolding for Each Problem

Problem No.	No. of Students	CORT Method	Instances of Support Types								Level of Cognitive Strategy (max 5)		Level of Scaffolding (max 3)	
			Syntax		Semantic		Structural		Algorithmic		Median	Avrg.	Median	Avrg.
			No.	%	No.	%	No.	%	No.	%				
2	4	2	2	66.7			1	33.3			4.5	4.0	3	2.8
3	4	1			3	50.0	3	50.0			4.5	4.3	3	3.0
4	4	2			3	50.0	1	16.7	2	33.3	3.5	3.8	3	2.8
5	3	2			3	60.0	1	20.0	1	20.0	3	2.8	3	3.0
6	2	1			2	66.7	1	33.3			3.5	3.5	3	3.0
7	4	2			4	57.1			3	42.9	3	3.3	3	2.8
8	2	3	2	66.7	1	33.3					3.5	3.5	2.5	2.5
9	1	2					1	100.0			2	2.0	2	2.0
10	1	2			1	50.0	1	50.0			3	3.0	3	3.0
11	3	1			1	33.3			2	66.7	3	3.7	3	3.0
12	1	3							1	100.0	4	4.0	2	2.0
13	3	1			3	37.5	2	25.0	3	37.5	4	4.0	3	2.7
14	1	3	1	33.3			1	33.3	1	33.3	4	4.0	2	2.0
15	3	3	1	10.0	3	30.0	3	30.0	3	30.0	3	3.3	2	2.3
16	1	1							1	100.0	3	3.0	2	2.0
17	4	2							3	100.0	3	3.5	3	3.0
18	3	3	2	33.3	3	50.0	1	16.7			3	3.0	2	2.3

The table shows the support types and the levels of cognitive support and scaffolding for each of the seventeen problems that were attempted by the students during the research experiment. Two patterns clearly emerge from the data. The first concerns the instances of semantic support provided by the CORT system as shown in columns 6 and 7. In the first nine problems, problem numbers 2 - 10, there were seventeen instances of semantic support for the total of twenty-five student observations that took place. In the eight remaining problems, problem numbers 11 - 18, there were only seven instances of semantic support for the total of sixteen student observations that took place. These results indicate that most semantic help took place earlier in the course when students were attempting to acquire much of the necessary semantic knowledge of various programming statements. By the latter part of the course, most students had constructed much of this semantic knowledge and required less help from the system.

The second finding concerns the instances of algorithmic support provided by the CORT system. In the first nine problems, problem numbers 2 - 10, there were six instances of algorithmic support for the total of twenty-five student observations that took place. In the eight remaining problems, problem numbers 11 - 18, there were fourteen instances of algorithmic support for the total of sixteen student observations that took place. These results indicate that most algorithmic help took place in the latter part of the course as the problems became progressively more difficult.

These two findings suggest that the CORT system provided most support for semantic difficulties early in the course and most support for algorithmic difficulties in the latter part of the course. It could therefore be argued that the design of the 17 problems was fairly sound. The progressive increase in difficulty of the problems has ensured that most students have been supported early on in their learning of programming language semantics at a time when the algorithmic difficulties that they faced were relatively low. Later in the course there were more difficult CORT problems and students faced many more algorithmic difficulties. However they were well supported by CORT. It seems that students no longer had to be as concerned with semantic difficulties. The cognitive load had been kept low in the early part of the course by ensuring the problem solutions had relatively simple algorithms. As the course progressed, less semantic support was necessary and it had been possible to increase the level of difficulty of the algorithms that were required for solutions, whilst keeping the cognitive load steady and not overloading students.

Table 6 contains the same data as that of Table 5; however the rows have been sorted by the "CORT Method" column which is the third column. Some properties of the data that seem to emerge from this table included the relatively large number of syntax supports that CORT had provided in method 3 type problems.

Table 6: Summary of Learning Supports, Levels of Cognitive Strategy and Levels of Scaffolding for Each Problem – Sorted by CORT Method

Problem No.	No. of Students	CORT Method	==========Instances of Support Types==========								Level of Cognitive Strategy (max 5)		Level of Scaffolding (max 3)	
			Syntax		Semantic		Structural		Algorithmic		Median	Avrg	Median	Avrg
			No.	%	No.	%	No.	%	No.	%				
3	4	1			3	50.0	3	50.0			4.5	4.3	3	3.0
6	2	1			2	66.7	1	33.3			3.5	3.5	3	3.0
11	3	1			1	33.3			2	66.7	3	3.7	3	3.0
13	3	1			3	37.5	2	25.0	3	37.5	4	4.0	3	2.7
16	1	1							1	100.0	3	3.0	2	2.0
2	4	2	2	66.7			1	33.3			4.5	4.0	3	2.8
4	4	2			3	50.0	1	16.7	2	33.3	3.5	3.8	3	2.8
5	3	2			3	60.0	1	20.0	1	20.0	3	2.8	3	3.0
7	4	2			4	57.1			3	42.9	3	3.3	3	2.8
9	1	2					1	100.0			2	2.0	2	2.0
10	1	2			1	50.0	1	50.0			3	3.0	3	3.0
17	4	2							3	100.0	3	3.5	3	3.0
8	2	3	2	66.7	1	33.3					3.5	3.5	2.5	2.5
12	1	3							1	100.0	4	4.0	2	2.0
14	1	3	1	33.3			1	33.3	1	33.3	4	4.0	2	2.0
15	3	3	1	10.0	3	30.0	3	30.0	3	30.0	3	3.3	2	2.3
18	3	3	2	33.3	3	50.0	1	16.7			3	3.0	2	2.3

Because the number of student observations varied between the three CORT methods, the data from Table 6 has been summarised in Table 7 in order to extract more meaning from the data.

Table 7: Summary of Learning Supports, Levels of Cognitive Strategy and Levels of Scaffolding for Each CORT Method

Cort Method	No. of Student Observations	Ratio of support instances : No. Student Observations				Lvl. Cog. Support (average)	Lvl. Scaffolding (average)
		Syntax	Semantic	Structural	Algorithmic		
1	13	0.00	0.69	0.46	0.46	3.7	2.7
2	21	0.10	0.52	0.24	0.43	3.2	2.8
3	15	0.40	0.47	0.33	0.33	3.6	2.2

Analysis of Data by CORT Method

In Table 7, the ratios of support instances to the number of student observations have been determined for each support type within each CORT method. Also, the overall average of levels of cognitive support and of levels of scaffolding have been determined.

The data reveals several interesting findings. Firstly, CORT has not supported syntax errors when CORT method 1 was used. This is not surprising as such errors cannot be made in method 1 type problems. CORT does however provide good syntax support for method 3 type problems when students have to key in some lines of code. The data also reveals that there is less algorithmic support for the higher CORT methods, the ratios of support instances to the number of student observations being 0.46, 0.43, and 0.33 respectively for CORT methods 1, 2 and 3. This seems to indicate that it is easier for students to determine an algorithm to solve a problem when all the required lines are available and there are no distracter lines. It is most difficult for students to determine a required algorithm when they have to key in lines and then CORT provides less support for them.

The table also shows that the levels of cognitive support were fairly even and strong across all three CORT methods. However, that level of scaffolding provided by CORT was lowest for method 3 type problems. Again, this was probably to be expected as the lines of code together with distracter lines provide strong scaffolding in methods 1 and 2. However method 3 type problems do not provide students with all the necessary line of code.

Conclusions

This paper has reported on the observations that were made of students as they engaged with the part-complete solution process through the CORT system during the semester in which a research experiment took place. The results demonstrated the following outcomes:

- The system provided strong scaffolding for student learning.
- Students engaged well with the system and generally used a thoughtful and considered cognitive strategy.
- The highest level of support was for student semantic difficulties although there was also strong support for algorithmic and structural difficulties.
- The system support for semantic difficulties was higher in the early stages of the course.

- The system support for algorithmic difficulties was higher in the latter stages of the course.
- Students mainly had syntax difficulties with method 3 type problems when they had to key in lines of code. The system did provide support, if only by indicating that a difficulty existed.
- The system provides better algorithmic support for method 1 and 2 type problems than with method 3 type problems.
- The level of scaffolding provided by the system was lowest for method 3 problems.

The data suggest that the part-complete methods used in the CORT system have a strong influence on the supports and scaffolding levels that are provided. It is probably not unexpected that students receive lower levels of scaffolding when they have to key in lines of code themselves to complete a solution rather than choose lines of code that have been provided to them. Care is therefore necessary in the design of a set of problems for an introductory programming course so that the scaffolding is reduced gradually in order to try and keep the cognitive load that students experience fairly constant.

References

du Boulay, B. (1986). Some difficulties in learning to program. *Journal of Educational Computing Research, 2*(1), 57-73.

Cohen, L. & Manion, L. (1994). *Research methods in education* (4th ed.). N. Y.: Routhledge.

Garner, S. K. (2003). Learning to program using part-complete solutions. Paper presented at the *Computer Based Learning in Science 2003,* Nicosia, Cyprus.

Garner, S. K. (2005). The CLOZE procedure and the learning of programming. Paper presented at the *International Conference on Learning*, Granada, Spain.

King, J., Feltham, J. & Nucifora, D. (1994). Novice programming in high schools: Teacher perceptions and new directions. *Australian Educational Computing,* September, 17-23.

Perkins, D. N., Schwartz, S. & Simmons, R. (1988). Instructional strategies for the problems of novice programmers. In R. E. Mayer (Ed.), *Teaching and learning computer programming: Multiple research perspective* (pp. 153-178). Hillsdale, NJ: Erlbaum.

Scholtz, J. & Wiedenbeck, S. (1992). The role of planning in learning a new programming language. *International Journal of Man-Machine Studies, 37*, 191-214.

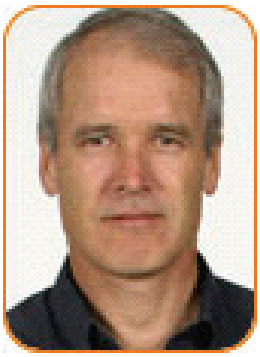
Soloway, E. (1986). Learning to program = Learning to construct mechanisms and explanations. *Communications of the ACM, 29*(9), 850-858.

van Merrienboer, J. J. G. (1990). Strategies for programming instruction in high school: Program completion vs. program generation. *Journal of Educational Computing Research, 6*(3), 265-285.

van Merrienboer, J. J. G. & De Croock, M. B. M. (1992). Strategies for computer-based programming instruction: Program completion versus program generation. *Journal of Educational Computing Research, 8*(3), 365-394.

Winslow, L. (1996). Programming pedagogy - A psychological overview. *SIGCSE Bulletin, 28*(3).

Biography



Stuart Garner has been a college and university lecturer for over 30 years and has also spent time working in industry as an analyst programmer. His main research interests include: the teaching and learning of programming; the teaching and learning of systems analysis and design; eLearning; personal knowledge management; and web based development.

Stuart is currently a senior lecturer in information systems at Edith Cowan University, Western Australia. His profile is available at: http://www.business.ecu.edu.au/schools/man/staff/sgarner.htm