A Quantitative Study of a Software Tool that Supports a Part-Complete Solution Method on Learning Outcomes

Stuart Garner
Edith Cowan University, Joondalup, Australia

s.garner@ecu.edu.au

Executive Summary

This paper reports on the findings from a quantitative research study into the use of a software tool that was built to support a part-complete solution method (PCSM) for the learning of computer programming. The use of part-complete solutions to programming problems is one of the methods that can be used to reduce the cognitive load that students experience during their learning.

The tool that was built is a code restructuring tool known as CORT. It permits the display of a part-complete solution to a programming task together with a set of possible lines of code that can be used to "fill-in" the solution. Students can then easily manipulate the programming lines within CORT and then test their solutions within a traditional development environment such as Microsoft's Visual Studio.

An inquiry into CORT's effect on student learning outcomes took place over a period of 14 weeks at an Australian university. Two student groups participated in an introductory programming unit, one of which utilized the CORT system whilst the other acted as a control group. Data were collected on student background, time taken to complete programming tasks, the amount of help that students required, and student learning outcomes.

The data were analysed using a statistical package, and it was found that there was no significant difference in the relative levels of achievement between students in the two groups. However the results did indicate that students in the CORT group spent significantly less time and required significantly less help than the students in the control group. This suggests that if the CORT students had spent extra time on further programming tasks such that their overall time had been equal to that of the non-CORT students, then they may have achieved higher learning outcomes. Also the fact that the CORT students required less help than the control group suggests that the use of CORT might be beneficial to students studying programming in external mode where access to tutor help is more problematic.

The data were also analysed to determine if there were any differences with respect to student achievement between the sub-groups of: previous achievement level; age; computer literacy level; previous programming experience; and gender. The analysis revealed that the CORT students who were either younger, only moderately computer literate, or without any previous pro-

Editor: Dion Goh

gramming experience, performed at a lower level than their non-CORT counterparts in reading and comprehending computer programs. This might be because these sub-groups have ill-developed mental models of program execution and therefore have greater difficulty comprehending code. It suggests that such students completed their programs relatively quickly with CORT, with less experimentation, and that this hindered their mental model development.

Finally the paper makes suggestions for future research.

Keywords: learning, programming, cognition.

Introduction

Most teachers of computer programming know that many students find learning to program a very difficult process. For example, Perkins, Schwartz, and Simmons (1988, p.155) stated, "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."

Jenkins (2002) suggests that the learning of programming is a perennial problem. Students struggle as they try to master the subject, and it is not uncommon for a student's first experience of programming to be so negative and stressful that it leads to academic failure or withdrawal. In a study into the teaching and learning of first year programming, it was found that the main concerns were high failure rates, a low flow of students into higher degrees, and a perception of a wide variation of teaching skills (Carbone, Hurst, Mitchell, & Gunstone, 2000).

Although the number and variety of students that attempt to learn to program has increased, high failure rates are a major problem and much of the literature provides many examples of new teaching approaches that have been used to try and overcome this problem (Bruce et al, 2004). A review of the literature on the teaching and learning of introductory programming reveals that there has been little, if any, research on how students go about learning to program (Bruce & McMahon, 2002). The review also points out that there have been many examples of innovative teaching practice implemented, but that these usually appear to have been developed independently of any research into the students' experience of learning programming. In practice, however, the ways in which teaching and learning takes place in the domain of programming have changed little and many students still find the learning of programming a very difficult process.

One strategy of learning programming with particular promise that could help address the problem is known as the part-complete solution method (PCSM). Earlier studies demonstrated its potential (e.g., van Merrienboer, 1990b; van Merrienboer & De Croock, 1992) but its success was never realised due to the absence of suitable electronic tools to support the process. In this method, students are provided with part-complete solutions to programming problems and their task is then to try and complete the solutions. The aim of the method is to reduce the cognitive load on students and help them learn more effectively.

A software tool named CORT (Code Restructuring Tool) has been built by the author to support the completion method, and the findings from a qualitative research study concerning how the use of the PCSM and CORT helped support student learning has been reported previously (Garner, 2007). Some of the findings from that study included: CORT provided strong scaffolding for student learning; students engaged well with the system and generally used a thoughtful and considered cognitive strategy; and CORT particularly supported semantic difficulties in the early stages of a course.

This paper reports on the second, quantitative part of the research study which explored the impact of the PCSM and CORT on students' learning outcomes and achievements. The paper briefly describes the CORT, outlines the research design, and finally discusses the results of the study.

The Use of Part-Complete Solutions in the Teaching and Learning of Programming

A lot of the work in the area of incomplete programming examples has been carried out by van Merrienboer and his colleagues (e.g., van Merrienboer, 1990a, 1990b; van Merrienboer & De Croock, 1992; van Merrienboer, Krammer, & Maaswinkel, 1994; van Merrienboer & Paas, 1990). They argue that the traditional approach to the teaching and learning of programming is ineffective and that the "Reading" approach is a better one to follow. However, they also suggest that presenting worked examples to students is not sufficient as the students may not "abstract" the programming plans from them. "Mindful" abstraction of plans is required by the voluntary investment of effort, and the question then arises as to how students can be motivated to study the worked examples properly. In practice, students tend to rush through the examples, even if they have been asked to trace them in a debugger, as they often believe that they are only making progress in their learning when they are attempting to solve problems.

One suggestion that has been put forward is that students should annotate worked examples with information about what they do or what they illustrate (Lieberman, 1986). Another suggestion is to use incomplete, well-structured, and understandable program examples that require students to generate the missing code or "complete" the examples. This latter approach forces students to study the incomplete examples as it would not be possible for their completion without a thorough understanding of the examples' workings. An important aspect is that the incomplete examples are carefully designed as they have to contain enough "clues" in the code to guide the students in their completion. It is suggested that this method facilitates both automation, students having blueprints available for mapping to new problem situations, and schemata acquisition as they are forced to mindfully abstract these from the incomplete programs (van Merrienboer & Paas, 1990).

In one study, two groups of 28 and 29 high-school students from grades 10 to 12 participated in a ten lesson programming course using a subset of COMAL-80 (van Merrienboer, 1990b). One group, the "generation" group, followed a conventional approach to the learning of programming that emphasised the design and coding of new programs. The other group, the "completion" group, followed an approach that emphasised the modification and extension of existing programs. It was found that the completion group was better than the generation group in constructing new programs. It was found that the percentage of correctly coded lines was greater and that looping structures were more often combined with correct variable initialisation before a loop together with the correct use of counters and accumulators within the loop. It would appear that the completion strategy had indeed resulted in superior schemata formation for those students within that group. In addition, the completion group used superior comments in connection with the scope and goals of the programs, indicating that they had developed better high-level templates or schemata. It was noted in the study, however, that both groups were equal in their ability to interpret programs and that this might indicate that students in the completion group do not understand their acquired templates. It is then suggested that future completion strategies should include the annotation of the examples by students with details of what they are supposed to do and details of the templates (plans) that are being used.

A side effect of the research was also noted. The drop-out rate from the completion group was found to be lower than for the generation group, particularly for female students with low prior knowledge. This is important as other studies have concluded that females are more anxious and less confident than males with respect to computer skills (e.g., Staehr, Martin, & Byrne, 2001; Werner, Hanks, & McDowell, 2004). van Merrienboer (1990b) suggested that the generation of complete programs is perceived as a difficult and menacing task and that the use of the completion strategy may help reduce the anxiety for some of the less confident students.

Another study was undertaken in which 40 undergraduates, undertaking a short course in turtle graphics programming, were divided into completion and generation groups (van Merrienboer & De Croock, 1992), both learning activities and learning outcomes being investigated. The course was divided into four parts, each part having three modules. Each of the three modules was presented as an incomplete solution to the completion group and the group had to complete the solutions. The first two modules were presented as completed solutions to the generation group and the third module required the group to construct a solution from scratch.

In the area of learning activities, it was found that the generation group often had difficulties in finding or coding a solution to their programming problems as they had to undertake frequent searches for useful information or examples. It was also found that the completion group took far fewer notes about the programming commands and their syntax than the generation group. It was hypothesised that the reason was the incomplete programs provided to the completion group contained a lot of this information.

With regard to learning outcomes, it was found that students in the completion group had acquired better low and high-level programming templates and that the semantic correctness of their constructed programs was superior to the generation group. As with the previous study, there was no difference between the groups in their ability to comprehend programs, however both groups' levels of comprehension were found to be high.

The "degree" of completion of the solutions is an important aspect within the completion strategy, and in some later work (van Merrienboer, Krammer, & Maaswinkel, 1994) examples are given of completion assignments that might be used early and later in a programming course. In an early part of a course, an example may indeed be complete and include explanations and a question on its inner workings. In the latter part of a course, an example may be largely incomplete and include a question on its workings and an instructional task. Between these two extremes, examples will have varying degree of completeness and, in all cases, the incomplete examples are acting as scaffolds for the students. A similar strategy was used in a study that investigated the effectiveness of fading within problem solving examples (Renkl, Atkinson, Maier, & Staley, 2002). Problem solving and example study were combined as follows:

- A complete example was presented to students;
- An example was given to students such that one solution step was omitted; and
- More steps were omitted until just the problem to be solved was left, i.e., independent problem solving.

It was found that this method produced reliable results with students on near-transfer items, i.e., for similar problem types, but not on far-transfer items.

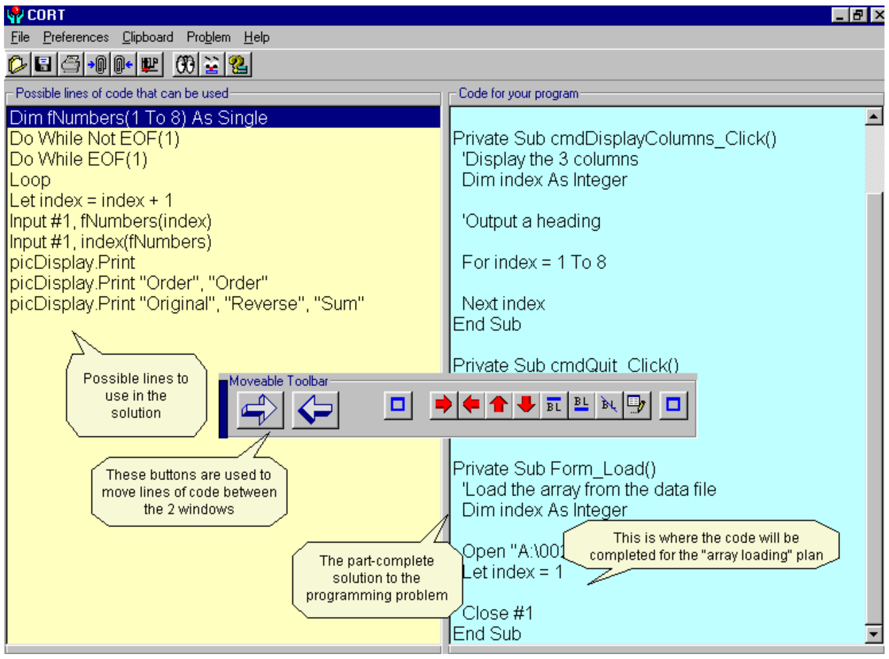
In summary, the research into the use of part-complete solutions and problem solving strongly suggests that this method of teaching and learning has great merit. In the learning of programming, the evidence suggests that the completion strategy is superior to the conventional generation strategy. By using the completion strategy:

- Students are better able to construct programs and abstract appropriate programming plans or schemata;

- There is a lower drop-out rate as students are not immediately faced with the daunting task of having to construct programs; and
- It results in a reduced cognitive load for students.

The CORT System

CORT has been described in other papers (e.g., Garner, 2003, 2005, 2007). 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.


CORT
File Preferences Clipboard Problem Help
Possible lines of code that can be used
Code for your program
Moveable Toolbar
Possible lines to use in the solution
These buttons are used to move lines of code between the 2 windows
The part-complete solution to the programming problem
This is where the code will be completed for the "array loading" plan

```
Dim fNumbers(1 To 8) As Single
Do While Not EOF(1)
Do While EOF(1)
Loop
Let index = index + 1
Input #1, fNumbers(index)
Input #1, index(fNumbers)
picDisplay.Print
picDisplay.Print "Order", "Order"
picDisplay.Print "Original", "Reverse", "Sum"
```

```
Private Sub cmdDisplayColumns_Click()
 'Display the 3 columns
 Dim index As Integer

 'Output a heading

 For index = 1 To 8

 Next index
End Sub

Private Sub cmdQuit_Click()

Private Sub Form_Load()
 'Load the array from the data file
 Dim index As Integer

 Open "A:\00
 Let index = 1

 Close #1
End Sub
```

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 clipboard 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 program.

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 impacted on student learning outcomes.

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. The programming language is Visual BASIC. The students participate in a two hour lecture / seminar and a one hour computer laboratory each week.

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 a group of students over a period of ten weeks. In addition, a control group attempted the same programming problems without the aid of CORT and the PCSM. All of the students were informed that it was not known if CORT and the PCSM would be an advantage or disadvantage to learning, and that they could move groups if they so wished. However all students chose to remain in their original groups with comprised 24 "CORT" students and 25 "non-CORT" students. The lecture was common to all students; however the groups attended different computer laboratories.

The data that were collected for this inquiry were as follows:

- Data concerning student background and past performance were obtained from a document study of the university's student record system. This enabled the collection of data concerning the average marks in units of study to date together with students' previous semester's average marks.
- Data concerning gender, age, previous achievement level, computer literacy level, and previous programming experience were collected from an initial questionnaire.
- The average time that students took to complete the set of problems and the average amount of help that students required were obtained from problem questionnaires that students filled out for each problem that they attempted.
- Data concerning student learning outcomes were obtained from a final examination. The examination was in two parts; the first part tested the students' ability to read and comprehend existing programming code, and the second part tested the students' ability to generate code having been given a problem specification.

The Data Analysis Method

The data analysis tool, SPSS (Morgan & Griego, 1998) was used to help with the statistical analysis of the data. SPSS is a well known statistical package that is widely used in quantitative research.

The data were analyzed in this research to determine if there were any significant differences in learning outcomes between the CORT and non-CORT students and, also, whether students from these two student groups differed significantly in the times that they took to complete problems and in the amount of help that they required. The student group (CORT or non-CORT) was an independent variable in the study and further analysis was also undertaken to determine if other independent variables, such as gender, significantly interacted with student group with respect to learning outcomes, and time and help required to complete problems.

The four dependent variables used in the analysis were:

- Exam Part A (reading and comprehension of existing programming code) which was taken by students at the end of the course;
- Exam Part B (generation of programming code to solve a problem) which was taken by students at the end of the course;
- Average time taken per problem for the set of problems that the students undertook during the semester; and
- Average help required per problem for the set of problems that the students undertook during the semester.

The main independent variable was student group. Within student group, data were collected across various groups or independent variables: previous achievement level, age, computer literacy level, previous programming experience, and gender. These variables have been previously shown as influences on programming achievement (e.g., van Merrienboer, 1990b) and were identified in this study to aid in the analysis.

An initial set of analyses was undertaken to explore the effect that the independent variable, student group, had on each of the four dependent variables. This set of analyses could be classified as basic difference tests (Morgan & Griego, 1998) in which t-tests or one-way ANOVA tests are used for the data analysis. Such tests are used to determine if there is a significant difference between the means of the dependent variable for the groups within the independent variable. As the student group was comprised of two possible values, CORT or non-CORT, t-tests could be used for the four analyses. One-way ANOVA tests are used when the independent variable has three or more possible values.

The second set of analyses concerned the significance of the interaction of student group with each of the five other independent variables, with respect to the value of each dependent variable. For example, "Did gender and student group interact significantly with respect to student performance in exam part A (reading and understanding programming code)?" As there were five other independent variables and four dependent variables, twenty such analyses were carried out. Each of the analyses was categorized as a complex difference question, such questions involving more than one independent variable (Morgan & Griego, 1998). A factorial ANOVA associational statistic is appropriate under such situations, and more specifically a two-way ANOVA was utilised as each question involved two independent variables.

The Data Analysis

Programming Achievement between Groups

Two t-tests were carried out to determine if there were any significant differences between CORT and non-CORT students for each of the two dependent variables which concerned programming achievement: Exam Part A, and Exam Part B. The first test is described in some detail in order that the reader might gain an understanding of the way in which the statistics are interpreted.

Differences in Exam Part A achievement among CORT and non-CORT students

Exam Part A was a test of the students' ability to read, trace, and understand programming code. CORT supports the "Reading" method of learning programming (van Merrienboer & Krammer, 1987), and this analysis was important to determine if the CORT group achieved higher marks than their non-CORT compatriots in a test of such knowledge. The maximum mark was 20. The t-test was carried out using SPSS and the group statistics that were output are shown in Table 1.

Table 1: Group Statistics for Student Group and Exam Part A

	Group (CORT / Non-CORT)	N	Mean	Std. Deviation	Std. Error Mean
Exam Part A	CORT	24	10.58	4.149	.847
	Non-CORT	25	10.88	3.655	.731

The table shows that 24 CORT students and 25 non-CORT students took the exam and that their mean marks were 10.58 and 10.88 respectively. The standard deviations of 4.15 and 3.66 suggested a similar spread of marks between the groups.

Table 2 shows the results of the t-test for Exam Part A.

Table 2: T-Test: Student Group and Exam Part A

		Levene's Test for Equality of Variances		t-test for Equality of Means						
									95% Confidence Interval of the Difference	
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	Lower	Upper
Exam Part A	Equal variances assumed	.364	.549	-.266	47	.791	-.30	1.116	-2.541	1.948
	Equal variances not assumed			-.265	45.720	.792	-.30	1.119	-2.549	1.956

This table indicates the results for two statistical tests. The first is the Levene test, which tests the assumption that the variances of the two groups, CORT and non-CORT, are equal. If Levene's F value is not statistically significant, then equal variances are assumed. In this particular case, the significance is 0.549 and, as this is greater than 0.05 ($p > 0.05$), equal variances are assumed. Because of this, the "Equal variances assumed" outcome is utilized in analyzing the t-test.

In this instance $t = -0.27$, with 47 degrees of freedom. The significance is 0.791. The result of the t-test

$$t(47) = -0.27, p > 0.05$$

revealed a non-significant difference between CORT and non-CORT students in their ability to read, trace and understand programming code.

Differences in Exam Part B achievement among CORT and non-CORT students

Exam Part B was a test of the students' ability to generate programming code in response to a problem statement. Generation of programming code is a much more difficult task than reading code (Linn & Dalbey, 1985) and CORT does not directly support it. In the weekly computer laboratories, non-CORT students had been required to generate all of their programs whilst the CORT students only had to complete part-complete solutions. The non-CORT students might therefore have had a learning advantage with respect to code generation.

The maximum possible mark for Exam Part B was 20. Table 3 shows that there were 24 CORT students and 25 non-CORT students who took the exam and that their mean marks were 11.63 and 11.36 respectively. The standard deviations of 3.54 and 3.28 were again quite similar showing a consistent spread.

Table 3: Group Statistics for Student Group and Exam Part B

	Group (CORT / Non-CORT)	N	Mean	Std. Deviation	Std. Error Mean
Exam Part B	CORT	24	11.63	3.536	.722
	Non-CORT	25	11.36	3.277	.655

Table 4 shows the results of the t-test for Exam Part B.

Table 4: T-Test: Student Group and Exam Part B

		Levene's Test for Equality of Variances		t-test for Equality of Means						
									95% Confidence Interval of the Difference	
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	Lower	Upper
Exam Part B	Equal variances assumed	.005	.941	.272	47	.787	.27	.973	-1.693	2.223
	Equal variances not assumed			.272	46.360	.787	.27	.975	-1.697	2.227

Levene's F value was not significant, and equal variances could be assumed. The result of the t-test

$t(47) = 0.27, p > 0.05$

revealed a non-significant difference between CORT and non-CORT students in their ability to generate programming code in response to a problem statement.

This finding indicates that the use of CORT had not disadvantaged the CORT group in their program generation, CORT students' achievement levels being as good as those of the non-CORT students.

Summary of programming achievement between groups

The tests that were carried out revealed that there was not a significant difference in learning achievement between the CORT and non-CORT student groups. However, the CORT students appeared to spend less time than the non-CORT students on the programming tasks and this may have limited the opportunities for the use of CORT to provide a learning advantage. Also, CORT may have only provided advantages to certain sub-groups of students. Both these possibilities were explored and are described later.

Programming Achievement Differences among Sub-Groups

Analyses were also carried out to determine if there were any significant interactions between student group, CORT and non-CORT, and each of the other five independent variables or sub-groups: previous achievement level, age, computer literacy level, previous programming experience, and gender with respect to student achievement. Two-way ANOVA statistics were produced for each of the two dependent variables associated with student achievement: Exam Part A and Exam Part B. The first tests are described in some detail in order that the reader might gain an understanding of the way in which the statistics are interpreted.

Previous achievement level

The CORT system has been designed to reduce cognitive load and provide scaffolding and learning supports for students. These two aspects have been associated with improved programming performance (e.g., Sweller, 1988) and analyses were undertaken of the interaction between the students' previous achievement, which was obtained from their course averages, and the student group (CORT or non-CORT), with respect to their student achievement in the final exam.

Interaction between Group and Previous Achievement Level for Exam Part A. The level of interaction between student group and previous achievement level was determined for Exam Part A: the students' ability to read and understand programming code.

Table 5 shows the numbers associated with the student group and previous achievement level. It indicates that 49 students took the exam and that there were 15, 21 and 13 students with corresponding previous achievement levels of "low", "medium", and "high".

Table 5: Student Group and Previous Achievement Level: Basic Statistics

		Value Label	N
Group (CORT / Non-CORT)	1	CORT	24
	2	Non-CORT	25
Previous achievement level	1	Low	15
	2	Medium	21
	3	High	13

Table 6 shows the descriptive statistics for this analysis. The data shows that for both CORT and non-CORT students, the marks were higher for those with greater levels of previous achievement. The standard deviations show a relatively larger spread of marks for the CORT, low previous achievers.

Table 6: Descriptive Statistics for Group, Previous Achievement and Exam Part A

Dependent Variable: Exam Part A

Group	Prev. Achievement	Mean	Std. Deviation	N
CORT	Low	9.50	5.732	8
	Medium	10.00	2.449	9
	High	12.57	3.599	7
	Total	10.58	4.149	24
Non-CORT	Low	9.14	4.140	7
	Medium	10.50	3.205	12
	High	13.67	2.658	6
	Total	10.88	3.655	25
Total	Low	9.33	4.880	15
	Medium	10.29	2.849	21
	High	13.08	3.121	13
	Total	10.73	3.866	49

The results of the two-way ANOVA test that was undertaken to explore results in Exam Part A across achievement are shown in Table 7.

Table 7: Two-way ANOVA for Group, Previous Achievement and Exam Part A

Dependent Variable: Exam Part A

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	110.646[a]	5	22.129	1.568	.190	.154
Intercept	5538.389	1	5538.389	392.402	.000	.901
GROUP	1.986	1	1.986	.141	.709	.003
ACHIEVEM	108.128	2	54.064	3.830	.029	.151
GROUP * ACHIEVEM	3.753	2	1.876	.133	.876	.006
Error	606.905	43	14.114			
Total	6364.000	49				
Corrected Total	717.551	48				

a. R Squared = .154 (Adjusted R Squared = .056)

The row "ACHIEVEM" reveals that, irrespective of student group, students with higher previous achievement levels scored significantly higher in Exam Part A. The result is:

F(2,43)=3.83, P<0.05

The relevant data concerning the level of interaction between student group and previous achievement level with respect to Exam part A is in the row "GROUP * ACHIEVEM. The result of the two-way ANOVA test

F(2,43) = 0.133, p>0.05

reveals a non-significant interaction between CORT and non-CORT students in Exam Part A. This means that although the figures in Table 7 show that CORT low achievers scored higher than non-CORT low achievers, and non-CORT medium and high achievers scored higher than their CORT counterparts, these differences are not significant.

Interaction between Group and Previous Achievement Level for Exam Part B. The level of interaction between student group and previous achievement level was determined for Exam Part B: the students' ability to generate programming code in response to a problem statement. As in the previous analysis, it was thought that weaker CORT students might perform better than weaker non-CORT students. The descriptive statistics are shown in Table 8.

Table 8: Descriptive Statistics for Group, Previous Achievement and Exam Part B

Dependent Variable: Exam Part B

Group	Prev. Achievement	Mean	Std. Deviation	N
CORT	Low	12.63	3.739	8
	Medium	9.67	3.202	9
	High	13.00	2.944	7
	Total	11.62	3.536	24
Non-CORT	Low	10.86	3.848	7
	Medium	11.00	2.412	12
	High	12.67	4.274	6
	Total	11.36	3.277	25
Total	Low	11.80	3.764	15
	Medium	10.43	2.785	21
	High	12.85	3.460	13
	Total	11.49	3.373	49

The results of the two-way ANOVA test that was undertaken to explore results in Exam Part B across achievement are shown in Table 9.

Table 9: Two-way ANOVA for Group, Previous Achievement and Exam Part B

Dependent Variable: Exam Part B

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	70.179[a]	5	14.036	1.268	.295	.128
Intercept	6315.160	1	6315.160	570.409	.000	.930
GROUP	.764	1	.764	.069	.794	.002
ACHIEVEM	51.635	2	25.817	2.332	.109	.098
GROUP * ACHIEVEM	21.114	2	10.557	.954	.393	.042
Error	476.065	43	11.071			
Total	7015.000	49				
Corrected Total	546.245	48				

a. R Squared = .128 (Adjusted R Squared = .027)

Unlike the previous ANOVA test for Exam Part A, there was no significant result that indicated that students, irrespective of group, with higher previous achievement levels did better in Exam Part B. The result is

F(2,43)=2.332, p>0.05

The results of the interaction between group and previous achievement

F(2,43)=0.954, p>0.05

reveals a non-significant interaction between group and previous achievement level in Exam Part B which is their ability to generate programming code in response to a problem statement.

Discussion Concerning CORT and Students' Previous Achievement Levels. CORT was designed to provide supports and scaffolds for students who are learning to program. Previous research (e.g., Chansilp & Oliver, 2002, 2004) found no advantage in overall programming performance when using a technology enabled system, but significant advantages for low achievers. It was intended that CORT would particularly benefit the less able student and, in the case of this research, such a student was one who had a relatively low previous achievement. However, the ANOVA tests revealed that there was no significant interaction between the student group and student previous achievement levels with respect to student programming achievement tests. In both tests, the descriptive statistics indicated that the CORT low achievers' marks were higher than their non-CORT counterparts. These differences were not large and the non-significant results might be due to the small number of students involved. The student numbers were 8 and 7 for the CORT and Non-CORT low achievers.

Student age

Anecdotal evidence suggests that more mature students often find learning to program more difficult than younger students. They may for example have less basic computer literacy knowledge and also be more anxious about programming. They are more likely to get frustrated when they cannot generate a working program to solve a problem. It was thought that mature students might benefit from CORT's learning supports. Younger students were defined as being 20 and under, whereas mature students were defined as being over 20.

Two ANOVA tests were again undertaken to examine the interaction between student group and student age with respect to achievement. The numbers within the different categories are shown in Table 10. As can be seen, the student numbers were fairly evenly distributed.

Table 10: Descriptive Statistics for Group, Age and Final Exam

Group	Age	N
CORT	20 or under	12
	21 or over	12
	Total	24
Non-CORT	20 or under	9
	21 or over	16
	Total	25
Total	20 or under	21
	21 or over	28
	Total	49

The results of the two, two-way ANOVA tests used to determine the significance of the interaction between student group and student age for student achievement are shown in Table 11.

Table 11: Two-Way ANOVA Tests for Group, Age and Level of Achievement

Achievement Measure	ANOVA Result	Significant?
Exam Part A	F(1,45) = 5.807, p<0.05	✓
Exam Part B	F(1,45)=0.014, p>0.05	

The only significant interaction between student group and age was for Exam Part A, which tested a students' ability to read and understand programming code. Table 12 shows the descriptive statistics for this analysis. It indicates that the mean mark for Exam Part A was 9.00 and 12.22 for the CORT and non-CORT younger students respectively. The spread of marks was greater for the CORT students. However the mean mark was 12.17 and 10.13 respectively for the CORT and non-CORT mature students. Their marks were consistently spread.

Table 12: Descriptive Statistics for Group, Age and Exam Part A

Dependent Variable: Exam Part A

Group	Age	Mean	Std. Deviation	N
CORT	20 or under	9.00	3.766	12
	21 or over	12.17	4.041	12
	Total	10.58	4.149	24
Non-CORT	20 or under	12.22	1.856	9
	21 or over	10.13	4.225	16
	Total	10.88	3.655	25
Total	20 or under	10.38	3.442	21
	21 or over	11.00	4.199	28
	Total	10.73	3.866	49

The ANOVA result is shown graphically in the profile plots of Figure 2.

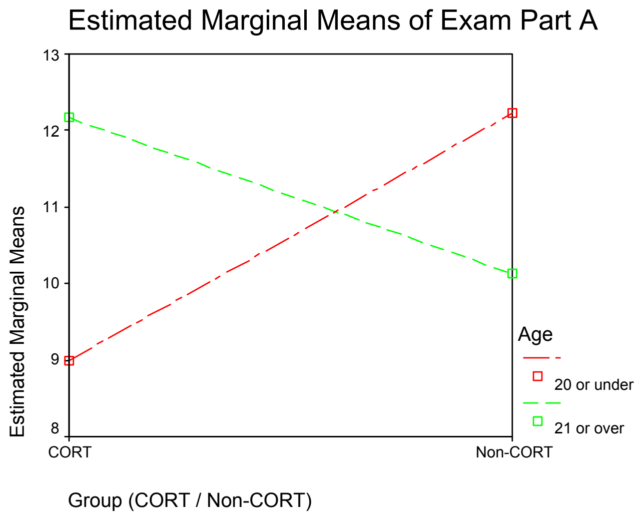


Figure 2: Profile Plots of Estimated Marginal Means of Exam Part A

This significant result shows that mature students who used CORT performed better than those who did not use CORT for Exam Part A. This part of the exam tested a student's ability to carefully read and trace programming code, and those who achieve at higher levels would most probably have a well developed mental model of the way in which programs execute. It is unclear if CORT helped mature students develop such mental models or if they already had appropriate models. In the latter case, it could be suggested that mental models are better developed by not using CORT as students tend to experiment more during the generation of programs "from scratch". This would explain why younger students who did not use CORT achieved better than their CORT counterparts.

The question arises why the other ANOVA test did not show a significant interaction between student group and age. This might have been that Exam Part B only tests the generation of code "from scratch" and CORT does not provide direct support for this.

Computer literacy level

Students who learn programming often have poorly developed mental models of the conceptual machine and also have misconceptions of various language constructs in programming (Bayman & Mayer, 1983). For example, they might have difficulty in: knowing where data comes from when input; how data is stored in memory; and the mechanism of assignment statements. Generally, people with higher levels of computer literacy have better developed mental models of the mechanisms of computers.

CORT provides learning supports and scaffolding to students; however, it is uncertain if it helps develop their mental models. Hence the question to be explored is whether students with lower levels of computer literacy perform better with or without CORT. Again, two ANOVA tests were undertaken to explore the interaction between student group and student computer literacy level for student achievement. The computer literacy level was specified as moderate or extensive. There were no students in the "limited" and "no computer literacy" categories.

The numbers within the different categories are shown in Table 13. The table shows that there were more students with "moderate" literacy than with "extensive" literacy.

Table 13: Descriptive Statistics for Group, Computer Literacy and Final Exam

Group (CORT / Non-CORT)	Computer Literacy Level	N
CORT	Moderate	9
	Extensive	12
	Total	21
Non-CORT	Moderate	18
	Extensive	7
	Total	25
Total	Moderate	27
	Extensive	19
	Total	46

The results of the two-way ANOVA tests used to determine the significance of the interaction between student group and student computer literacy for student achievement are shown in Table 14.

Table 14: Two-Way ANOVA Tests for Group, Computer Literacy and Level of Achievement

Achievement Measure	ANOVA Result	Significant?
Exam Part A	F(1,42)=0.042, p<0.05	✓
Exam Part B	F(1,42)=0.062, p>0.05	

The only significant interaction between student group and Computer Literacy was for Exam Part A, which tested a students' ability to read and understand programming code. Table 15 shows the descriptive statistics for this analysis. It indicates that the mean mark for Exam Part A was 8.44 and 11.56 respectively for the CORT and non-CORT students who had moderate computer literacy. The mean marks were 11.33 and 9.14 respectively for the CORT and non-CORT students who had extensive computer literacy. The spread of marks was relatively even for all four combinations of group and literacy.

Table 15: Descriptive Statistics for Group, Computer Literacy and Exam Part A

Dependent Variable: Exam Part A

Group	Computer Literacy Level	Mean	Std. Deviation	N
CORT	Moderate	8.44	3.972	9
	Extensive	11.33	3.846	12
	Total	10.10	4.073	21
Non-CORT	Moderate	11.56	3.666	18
	Extensive	9.14	3.237	7
	Total	10.88	3.655	25
Total	Moderate	10.52	3.984	27
	Extensive	10.53	3.702	19
	Total	10.52	3.828	46

This significant result suggests that students with moderate levels of computer literacy who did **not** use CORT performed better on Exam Part A than those who did use CORT. The opposite is true for those students who had extensive computer literacy and this is shown graphically in the profile plot of Figure 3.

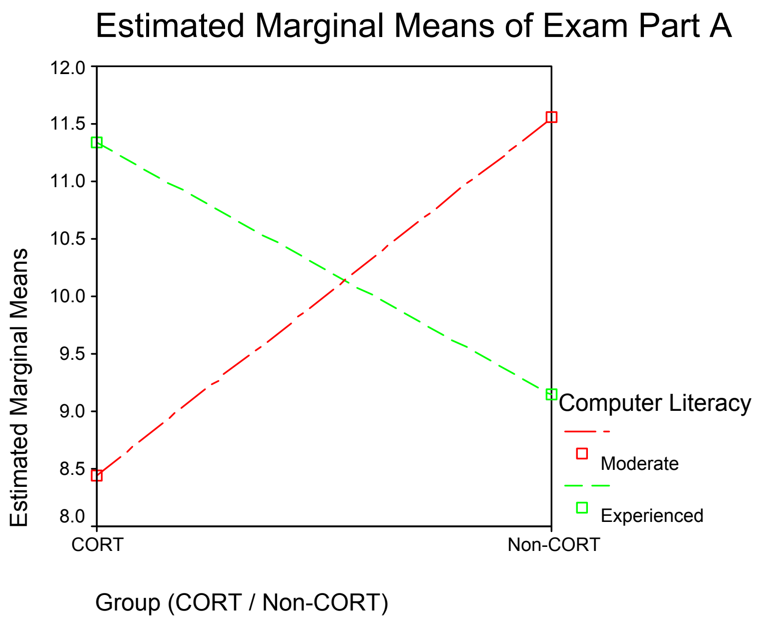


Figure 3: Profile Plots of Estimated Marginal Means of Exam Part A

Exam Part A measures a student's ability to read, trace, and understand programming code, and this requires students to possess a sound mental model of a program's execution process. The results suggest that students with moderate levels of computer literacy gained better mental models by learning in the "conventional" manner without the aid of CORT. A possible reason is that CORT reduces a student's capacity to experiment with code and to make mistakes, such activities perhaps being necessary in mental model construction.

Students with extensive computer literacy performed better with the aid of CORT. This could be because such students already had well developed mental models or were able to create such models relatively quickly. Then, the use of CORT provided them with the necessary learning supports to achieve at a high level.

A conclusion that might be drawn for this result is that students who have lower levels of computer literacy need to construct knowledge and relevant mental models about the conceptual machine before they use CORT to help them learn programming.

Previous programming experience

In an introductory programming unit like that in which this research has been carried out, it would usually be expected that students would have little previous programming experience. However some students may have gained limited familiarity with programming by, for example, the viewing and amendment of simple scripts on the Internet. Similarly to students who have extensive computer literacy, it would be expected that those students who have some previous programming knowledge may have better developed mental models than those with no previous knowledge. Students in the study were classified as having "none" or "limited" previous programming experience. In the original questionnaire that students completed at the beginning of the unit, there had been a category of "moderate" programming experience. However no students indicated that they were in this category.

The numbers within the different categories of programming experience are shown in Table 16, the numbers being distributed fairly evenly.

Table 16: Descriptive Statistics for Group, Previous Programming Experience and Final Exam

Group (CORT / Non-CORT)	Previous Programming Experience	N
CORT	None	11
	Limited	10
	Total	21
Non-CORT	None	14
	Limited	11
	Total	25
Total	None	25
	Limited	21
	Total	46

The results of the two-way ANOVA tests used to determine the significance of the interaction between student group and previous programming experience for student achievement are shown in Table 17.

Table 17: Two-Way ANOVA Tests for Group, Previous Programming Experience and Level of Achievement

Achievement Measure	ANOVA Result	Significant?
Exam Part A	F(1,42)=7.180, p<0.05	✓
Exam Part B	F(1,42)=1.148, p>0.05	

The results indicated that there were significant interactions between student group and previous programming experience for Exam Part A. Table 18 shows the descriptive statistics for this analysis.

Table 18: Descriptive Statistics for Group, Previous Programming Experience and Exam Part A

Dependent Variable: Exam Part A

Group	Prev. Prog. Experience	Mean	Std. Deviation	N
CORT	None	8.55	4.204	11
	Limited	11.80	3.327	10
	Total	10.10	4.073	21
Non-CORT	None	12.00	3.138	14
	Limited	9.45	3.908	11
	Total	10.88	3.655	25
Total	None	10.48	3.970	25
	Limited	10.57	3.749	21
	Total	10.52	3.828	46

This significant result is similar to that obtained for the student group and computer literacy. That is, non-CORT students without any previous programming experience performed better than CORT students without any previous programming experience in Exam Part A. The respective marks were 12.00 and 8.55. CORT students with limited previous programming experience performed better than their non-CORT counterparts, the marks being 11.80 and 9.45 respectively. This is shown graphically in the profile plot of Figure 4.

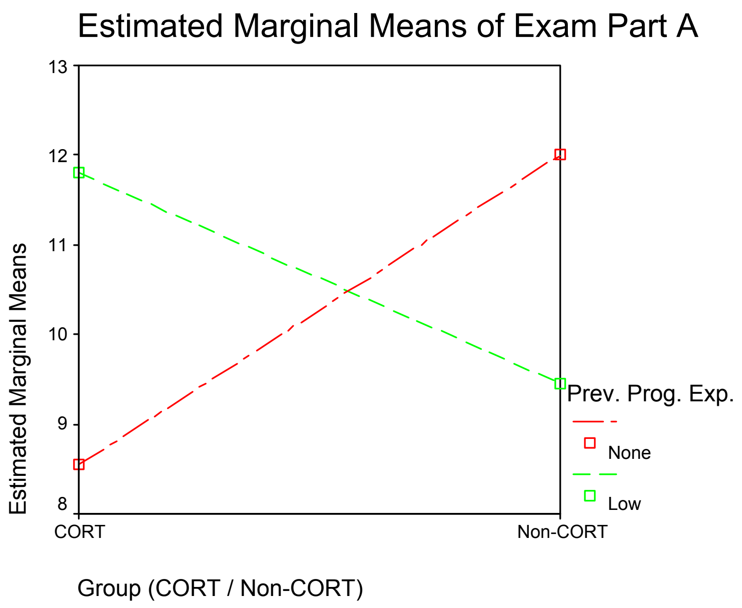


Figure 4: Profile Plots of Estimated Marginal Means of Exam Part A

The results are similar to those in computer literacy and suggest that students with no previous programming experience may need to construct knowledge and relevant mental models about the conceptual machine before they use CORT to help them learn programming. Those students who already have some programming knowledge appeared to achieve better with the aid of CORT as these students probably have better developed mental models and can then use CORT's learning supports to help them build relevant programming plans and schemata.

Gender

Research has shown that female students who have learnt to program using the completion method experienced less anxiety and lower drop-out rates than those learning using the more traditional "generation" method (van Merrienboer, 1990b). This suggests that females may be more comfortable using CORT as it directly supports the completion method and that this may impact on their performance.

As previously ANOVA tests were undertaken to explore the interaction between student group and gender for student achievement. The numbers within the different categories are shown in Table 19.

Table 19: Descriptive Statistics for Group, Gender and Final Exam

Group (CORT / Non-CORT)	Gender	N
CORT	Male	18
	Female	6
	Total	24
Non-CORT	Male	14
	Female	11
	Total	25
Total	Male	32
	Female	17
	Total	49

The results of the two-way ANOVA tests used to determine the significance of the interaction between student group and gender for student achievement are shown in Table 20.

Table 20: Two-Way ANOVA Tests for Group, Gender and Level of Achievement

Achievement Measure	ANOVA Result	Significant?
Exam Part A	F(1,45)=0.012, p>0.05	
Exam Part B	F(1,45)=0.953, p>0.05	

The tests indicated that there was no significant difference between the males and females within the CORT and non-CORT groups. Female students in the CORT group may be more comfortable and less anxious than their non-CORT counterparts; however, this has not been reflected in improved performance. An investigation into how CORT impacts on the affective domain of females could be an area of future investigation.

Time and Help Requirements between Groups

Two other important factors in programming outcomes and learning are the time taken by students to complete their weekly programming problems and the amount of help that students required. A series of tests was used to explore whether the use of CORT revealed significant differences in these factors. It would be a strength of CORT if it could be shown to reduce time and / or help requirements.

Differences in time taken to complete programming problems between CORT and non-CORT students

As part of their learning process, students were asked to estimate and record the time that they took to complete each programming problem. This was done as they attempted the computer laboratories during the semester. The data were recorded in their individual problem questionnaires. It has been suggested (van Merrienboer & De Croock, 1992) that students who have to generate code spend a lot of time searching for relevant worked examples. Because of this, it was thought that there could be a difference between the times taken by students to generate programs and the times taken to complete part-complete programs. The non-CORT group had to generate all their programs whereas the CORT group had to complete part-complete programs. The data collected enabled the testing of this proposition.

The "Average time taken per problem" was the average time that students took to complete each of the programming problems during the semester. Table 21 shows that data were collected for 21

CORT students and 23 non-CORT and that their mean average times were 25.1 and 33.4 minutes respectively. This indicates that the non-CORT group took an average of 8 minutes longer to complete each of their programming problems.

Table 21: Group Statistics for Student Group and Time Taken to Complete Problems

	Group (CORT / Non-CORT)	N	Mean	Std. Deviation	Std. Error Mean
Average time taken per problem	CORT	21	25.14	4.757	1.038
	Non-CORT	23	33.43	9.199	1.918

The data were then further analyzed and Table 22 shows the results of the t-test for the students groups' times to complete programming problems.

Table 22: T-Test: Student Group and Average Time Taken per Problem

		Levene's Test for Equality of Variances		t-test for Equality of Means						
									95% Confidence Interval of the Difference	
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	Lower	Upper
Average time taken per problem	Equal variances assumed	4.393	.042	-3.701	42	.001	-8.29	2.240	-12.813	-3.771
	Equal variances not assumed			-3.802	33.603	.001	-8.29	2.181	-12.726	-3.858

In this case, Levene's F value was significant ($p<0.05$) and equal variances could not be assumed. The result of the t-test

$$t(33.6) = -3.80, p<0.05$$

revealed a significant difference between CORT and non-CORT students with respect to the average time taken per problem. The differences are shown graphically in the box plot of Figure 5. The box plot also gives a visual indication of the larger spread of times for the non-CORT group, the standard deviation being almost twice that of the CORT group. The boxes represent the middle 50% of cases, that is between the 25th and 75th percentiles. The horizontal line inside the box represents the median. The horizontal lines that are not within a box are known as whiskers and represent the expected range of times. The small circle represents an "outlier" or extreme value.

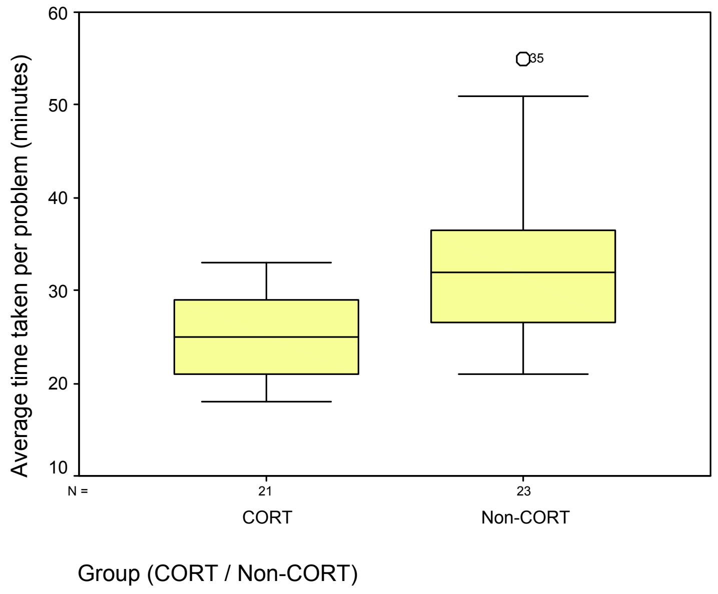


Figure 5: Box plot of "Average time taken per problem" for CORT / Non-CORT Students

The significant difference between the means of the two groups is perceived as very important especially when the previous analyses are taken into account. The previous analyses revealed no significant difference between the end-test performance of the CORT and non-CORT students. However the CORT students had taken only 76% of the time that the non-CORT students had taken. On average, the CORT students had each taken a total of 144 minutes less time to complete all of their problems during the class activities.

The results suggest that there may well have been significant differences in achievement between the two groups if the CORT students had spent as much time practising their programming skills as their non-CORT counterparts. Another conclusion that can be drawn is, that by using CORT, students can reduce the time required to achieve competence in programming.

Differences in help required to complete programming problems among CORT and non-CORT students

Students were asked to estimate the amount of help that they required in the completion of each programming problem. Students could obtain help from other students, the tutor, or the textbook, and they recorded the data concerning this in their individual problem questionnaires.

It has been demonstrated in previous research that students who have to generate code from "scratch" to solve a problem will require more help and, for example, use textbooks and other resources to find programming code that has been used to solve a similar problem to the one that they are attempting (van Merrienboer & De Croock, 1992). The use of CORT could possibly reduce the reliance on help from a tutor and the need to search various resources when attempting to solve programming problems. This is because CORT provides learning supports via its part-complete solution approach.

The "Average help required per problem" was the average help that students required to complete each of the eighteen programming problems during the semester. The data were collected via the individual problem questionnaires that the students completed for each problem that they attempted. The help values were coded in the range 1 to 4. The codes corresponded to: no help; little help; moderate help; and extensive help respectively. Students estimated the help needed for each of the programming problems and these values were then averaged. Table 23 shows that data were collected from 21 CORT students and 23 non-CORT students. The mean average help required was found to be 2.25 and 2.70 respectively on the 1 to 4 scale.

Table 23: Group Statistics for Student Group and Help Required to Complete Problems

	Group (CORT / Non-CORT)	N	Mean	Std. Deviation	Std. Error Mean
Average help required per problem	CORT	21	2.2524	.41427	.09040
	Non-CORT	23	2.7043	.53127	.11078

The data were then further analyzed and Table 24 shows results of the t-test for the student groups' help requirements to complete programming problems.

Table 24: T-Test: Student Group and Average Help Required per Problem

		Levene's Test for Equality of Variances		t-test for Equality of Means						
									95% Confidence Interval of the Difference	
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	Lower	Upper
Average help required per problem	Equal variances assumed	1.131	.294	-3.125	42	.003	-.4520	.14461	-.74381	-.16012
	Equal variances not assumed			-3.161	41.038	.003	-.4520	.14298	-.74072	-.16321

Levene's F value was found to be not significant in this case and equal variances were not assumed. The result of the t-test

$$t(42) = -3.13, p<0.05$$

revealed a significant difference between CORT and non-CORT students with respect to the average amount of help required per problem. The differences are shown graphically in the box plot of Figure 6. This indicates a greater spread of help requirements for the non-CORT group.

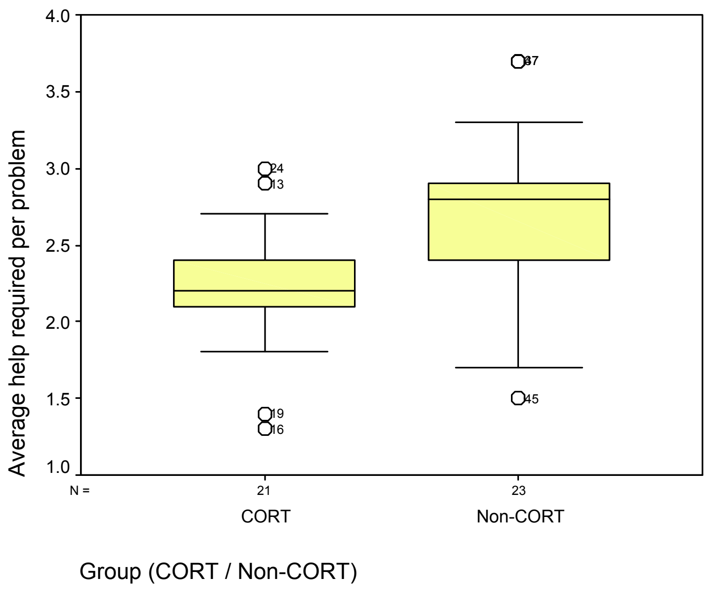


Figure 6: Box plot of "Average help required per problem " for CORT / Non-CORT Students

The ratio scale of 1 to 4 that the students had used to estimate the help that they had required corresponded to none, little, moderate, and extensive. The box plot of Figure 6 reveals that most CORT students needed little help whereas the majority of non-CORT students required moderate help. Students also indicated the type of help that they needed for each problem. They obtained most help from their textbooks; however, a substantial amount was also obtained from their tutor. This finding has implications for the ability of students to learn programming independently. Often students have difficulties learning programming when they reach points, when solving a problem, that they cannot go beyond until they have had some help from a tutor (e.g., Garner, 2002). The fact that CORT students required significantly less help could be attributed to the high level of support that it provides. One implication of this would be that students using CORT would be able to learn programming more independently than non-CORT students and this would be especially beneficial for distance learning students.

The results reveal that CORT students sought less help than non-CORT students in their programming tasks. This suggests CORT could be a useful tool for supporting students needing to study programming independently or remotely.

Conclusions

The results of the data analyses provide mixed outcomes concerning the impact of the PCSM within CORT on student learning. No significant difference was found in the relative achievement of students in the CORT and non-CORT groups in tests of programming achievement, while the results showed significant advantages for the CORT students in terms of time saving and levels of tutor help required. The results suggest that the system can help students to complete programming tasks more quickly and can provide higher levels of support, both factors providing advantage for novice programmers.

Although there was no significant difference between the CORT and non-CORT students in any of the achievement measures, differences did emerge between certain student sub-groups with respect to Exam Part A and these are summarized in Table 25.

Table 25: Significant Achievement Levels amongst Students for Exam Part A

		Higher Achievement		Lower Achievement	
		CORT	Non-CORT	CORT	Non-CORT
Age	Younger		✓	✓	
	Mature	✓			✓
Computer Literacy	Moderate		✓	✓	
	Extensive	✓			✓
Prev. Prog. Experience	None		✓	✓	
	Little	✓			✓

The table entries reveal a similar pattern amongst the categories. That is that the students who used the CORT system who achieved at a lower level were either younger, only moderately computer literate, or without any previous programming experience. Exam Part A was a measure of the students' ability to read and comprehend computer programs. The common factor among the categories may be that such students do not have a satisfactory and well defined mental model of the way in which computers execute programs and they, therefore, have greater difficulty comprehending code.

Research has shown that students with ill-developed mental models can be supported in the learning of programming by helping them visualize the execution process of the programs (e.g., Smith & Webb, 2000). The non-CORT students who learnt programming in a "conventional" manner had to spend a lot of time experimenting as they attempted to generate their programs. This may well have helped them develop appropriate mental models. However, students who used the CORT system were able to complete their programs relatively quickly and with less experimentation. This might not have helped students in their mental model development.

Those students who already had well developed mental models did benefit from using the CORT system. Students who were either mature, with high levels of computer literacy, or with some previous programming experience, performed better than their non-CORT counterparts.

The time that the CORT students required to complete all of the programming tasks was approximately three quarters of that of the non-CORT students. Significant differences in achievement may well have been demonstrated if the CORT students had spent the extra time that they had practising further programming problems.

The CORT students were found to require little help when using the system whereas the non-CORT students required moderate help. Whilst this does not impact directly on achievement levels, it may affect the achievement levels of students who do not have easy access to sources of help. It is well known (e.g., Garner, 2002) that distance learning students of programming have difficulties when they reach a point in solving a programming problem, such that they cannot proceed further until they obtain help from a tutor. If such help is not readily available, or the time taken for a tutor to respond to a query is long, then student achievement may be reduced. The PCSM within CORT provides help and learning supports to students so that the extra help that students need is relatively low. Distance learning students may therefore benefit if they were to use the system in an introductory programming unit.

The fact that the system did not provide learning advantages was not an expected outcome of the study. It is possible that there could have been differences observed in achievement under different conditions. Factors that may have contributed include:

- A lack of sensitivity in the various exams and instruments to the learning supported by the system;
- Insufficient use of the system among the students for the treatment to make a difference; and
- Too small a sample size for differences to emerge.

Recommendations for Future Research

The research conducted in this study confirmed the possibility of utilising a technology supported part-complete solution method, in the form of the CORT system, with students in introductory programming classes. The study investigated only one major aspect of learning outcome, i.e., student achievement in tests. Other outcomes that seem worthy of investigation are:

- The impact of the use of the CORT system on the time needed to learn;
- The type of part-complete CORT problem that is best able to support learning; and
- The impact of the development of students' mental models prior to the use of the CORT system.

Other possible future research might include:

- The use of the CORT system with remote learners;
- The impact of the CORT system on motivation and the affective domains of students.

References

Bayman, P., & Mayer, R. E. (1983). A diagnosis of beginning programmers' misconceptions of basic programming statements. *Communications of the ACM, 26*(9), 677-679.

Bruce, C., Buckingham, L., Hynd, J., McMahon, C., Roggenkamp, M., & Stoodley, I. (2004). Ways of experiencing the act of learning to program: A phenomenographic study of introductory programming students at university. *Journal of Information Technology Education*, 3, 143-160. Retrieved from http://www.jite.org/documents/Vol3/v3p143-160-121.pdf

Bruce, C., & McMahon, C. (2002). *Contemporary developments in teaching and learning introductory programming: Towards a research proposal*. QUT. Retrieved 17 Feb 2005, from http://sky.fit.qut.edu.au/~bruce/phen/learnprog/T&LReportPB3Dec.pdf

Carbone, A., Hurst, J., Mitchell, I., & Gunstone, D. (2000). *Principles for designing programming exercises to minimise poor learning behaviours in students.* Paper presented at the Fourth Australasian Computing Education Conference, Melbourne, Australia.

Chansilp, K., & Oliver, R. (2002). *Using multimedia to develop students' programming concepts*. Paper presented at the EDU-COM 2002, Khon-Kaen, Thailand.

Chansilp, K., & Oliver, R. (2004). *Students' responses to the use of a multimedia tool for learning computer programming*. Paper presented at the Ed-Media 2004, Lugano, Switzerland.

Garner, S. K. (2002). *COLORS for programming: A system to support the learning of programming*. Paper presented at the Informing Science & IT Education (InSITE) Conference 2002, University College Cork, Ireland. Retrieved from http://proceedings.informingscience.org/IS2002Proceedings/papers/Garne069COLOR.pdf

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.

Garner, S. K. (2007). An Exploration of How a Technology-Facilitated Part-Complete Solution Method Supports the Learning of Computer programming. *Journal of Issues in Informing Science and Information Technology, 4*, 491-501. Retrieved from http://proceedings.informingscience.org/InSITE2007/IISITv4p491-501Garn260.pdf

Jenkins, T. (2002). *On the cruelty of really teaching programming.* Paper presented at the 2nd LTSN-ICS one day conference on the teaching of programming, University of Wolverhampton, UK.

Lieberman, H. (1986). An example based environment for beginning programmers. *Instructional Science, 14*(3), 277-292.

Linn, M., & Dalbey, J. (1985). Cognitive consequences of programming instruction. *Educational Psychologist, 20*(4), 191-206.

Morgan, G., & Griego, O. (1998). *Easy use and interpretation of SPSS for Windows*. Mahwah, NJ: Lawrence Erlbaum.

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.

Renkl, A., Atkinson, R., Maier, U., & Staley, R. (2002). From example study to problem solving: Smooth transitions help learning. *Journal of Experimental Education, 70*, 293-315.

Smith, P. A., & Webb, G. I. (2000). The efficacy of a low-level program visualisation tool for teaching programming concepts to novice C programmers. *Journal of Educational Computing Research, 2*(2), 187-215.

Staehr, L., Martin, M. & Byrne, G. (2001). *Computer attitudes and computing career perceptions of first year computing students*. Paper presented at the Informing Science Conference 2001, Krakow University of Economics, Krakow, Poland. Retrieved from http://proceedings.informingscience.org/IS2001Proceedings/pdf/staehrEBKcompu.pdf

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

van Merrienboer, J. J. G. (1990a). Instructional strategies for teaching computer programming: Interactions with the cognitive style reflection-impulsivity. *Journal of Research on Computing in Education, 23*(1), 45-53.

van Merrienboer, J. J. G. (1990b). 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.

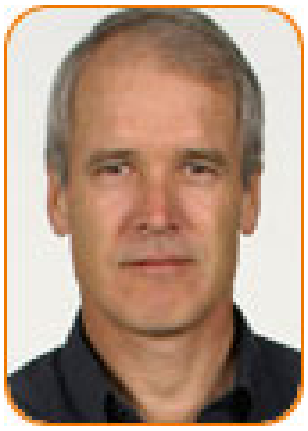
van Merrienboer, J. J. G., & Krammer, H. (1987). Instructional strategies and tactics for the design of introductory computer programming courses in high school. *Instructional Science, 16*(3), 251-285.

van Merrienboer, J. J. G., Krammer, H. P. M., & Maaswinkel, R. M. (1994). Automating the planning and construction of programming assignments for teaching introductory computer programming. In R. D. Tennyson (Ed.), *Automating instructional design, development, and delivery* (pp. 61-77). Springer Verlag.

van Merrienboer, J. J. G,. & Paas, F. (1990). Automation and schema acquisition in learning elementary computer programming. *Computers in Human Behavior, 6*(3), 273-289.

Werner, L., Hanks, B., & McDowell, C. (2004). *Female computer science students who pair program persist*. Retrieved 18 Jan, 2005, from http://www.cse.ucsc.edu/~charlie/pubs/jeric2004.pdf.

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