COLORS for Programming: A System to Support the Learning of Programming

Stuart Garner
Edith Cowan University, Perth, Australia

s.garner@ecu.edu.au

Abstract

Learning introductory software development is a difficult task and students often perceive programming subjects as requiring significantly more work than others. This paper describes a learning model for programming that has its basis in cognitive load theory. This theory suggests that there are three types of cognitive load that learners experience: intrinsic which is determined by the mental demands of the domain of knowledge; extraneous which is generated by the instructional format used in the teaching and learning process; and germane which can be utilised by learners to engage in conscious processing.

The learning model is used as a basis, together with a particular instructional design framework, for the development of "COLORS (Cognitive Load Reduction System) for Programming". COLORS is described together with a software tool, CORT (Code Restructuring Tool), that has been developed by the author to support various aspects of COLORS.

Keywords: cognitive load theory; programming; instructional design; code restructuring.

Introduction

Learning introductory software development is a difficult task and students often perceive programming subjects as requiring significantly more work than others (Green, 1998). Students need to understand concepts such as structuring code, reusability, ease of maintenance, meaningful naming, and user interface design for all but the simplest programs. Fowler & Fowler (1993) suggest that the challenge of learning programming in introductory courses lies in simultaneously learning: general problem solving skills; algorithm design; program design; a programming language in which to implement algorithms as programs; and a software tool (the programming environment) that supports design and implementation. This leads to many students feeling overwhelmed.

This paper describes a learning model for programming that takes cognitive load theory into account. The model is used as a basis, together with a particular instructional design framework (Oliver, 1999), for the development of "COLORS (Cognitive Load Reduction System) for Programming". COLORS is then described together with a software tool, CORT (Code Restructuring Tool), that has been developed by the author to support various aspects of COLORS.

Cognitive Load Theory

Cognitive load theory is built upon the idea that working memory is limited to around seven chunks of material (Miller, 1956) and that people can only deal with two or three elements simultaneously.

The degree of interactivity between the elements also affects the capacity of working memory.

Chess playing can be considered a problem solving domain and research (Chase & Simon, 1973) showed that the main difference between novices and experts was the fact that the latter had thousands of board configurations, as many as 100000 (Simon & Gilmartin, 1973), stored in long-term memory within schemata. The consequence is that, unlike less-skilled players, experts do not have to spend as much time searching for good chess moves using their limited working memory. Similarly, research into problem solving (Carroll, 1994) confirmed that, compared to novices, experts have knowledge of an enormous number of problem states and their associated moves. Such states are within long-term memory and such research indicates that human problem solving comes from stored knowledge and not from complex reasoning within working memory. It is suggested that humans are poor at complex reasoning unless most of the elements with which we reason are already in long-term memory, working memory being incapable of highly complex interactions using novel elements (Sweller, van Merrienboer, & Paas, 1998). This means that novices who are attempting a problem must engage in complex chains of reasoning using their working memory and in doing so it is likely that working memory will be overburdened. In other words the cognitive load on novices is too great.

Ways in which cognitive load can be reduced for novice problem solvers are therefore very important. In the schema theory of model representation, a schema can be anything that can be treated as a single entity or element such as a mathematical formula or a particular programming algorithm. Schemata have the function of storing knowledge and reducing the burden on working memory.

Experts can process information relevant to their domain automatically, novices however having to process information consciously (Schneider & Shiffrin, 1977; Tindall-Ford, Chandler, & Sweller, 1997). An example of such automatic processing is that of the expert driver who can drive their car without apparently thinking, whereas a learner driver has to consciously think of several things at the same time such as depressing the clutch and shifting to a new gear, observing the road ahead, moving the steering wheel etc. Any instructional design for a domain has to therefore not only encourage the construction of sophisticated schemata but also encourage the automatic processing of those schemata. This is important because of the limited capacity of working memory that can only deal with a few schemata at the same time. The ease with which information can be processed in working memory is the main thrust of cognitive load theory.

Working memory may be affected by intrinsic, extraneous and germane cognitive load (Sweller et al., 1998) and an understanding of these three categories is helpful for instructional designers.

Intrinsic Cognitive Load

Intrinsic cognitive load is determined by the mental demands of the task (Chandler & Sweller, 1996). Some tasks such as the learning of the basic vocabulary of a foreign language have a very low intrinsic cognitive load. Each element or schema is independent from the others with no interactivity and subsequently the required mental processing, or intrinsic cognitive load, is low. Tasks that have low element interactivity can be learnt serially rather than simultaneously. Tasks with a high degree of element interactivity have a heavy intrinsic cognitive load and an example is the learning of the grammar of a foreign language as all the words in phrases need to be considered and processed at once.

Computer programming is considered to be a domain with a high intrinsic cognitive load and this needs to be recognised in any instructional design. The intrinsic cognitive load cannot be reduced, however something can be done about the extraneous cognitive load.

Extraneous Cognitive Load

Extraneous cognitive load is generated by the instructional format used in the teaching and learning process and poor design leads to a high extraneous cognitive load. If a high extraneous cognitive load is combined with a high intrinsic cognitive load then this can lead to working memory overload. This is often what happens with novice programmers when the instructional design is poor.

The important point is that when the intrinsic cognitive load of the material is high, then it is incumbent on the instructional designer to think very carefully and ensure that the extraneous cognitive load is as low as possible. A lot of research has been done in looking at ways of reducing extraneous cognitive load, for example (Chandler & Sweller, 1991; Kalyuga, Chandler, & Sweller, 1998; Tindall-Ford et al., 1997). These include: integrating diagrams and text so as to reduce the "split-attention" effect; goal-free problem solving; and the use of worked examples in problem solving.

Germane Cognitive Load

It is thought that if the instructional design is such that the extraneous cognitive load is kept to a minimum, and the intrinsic cognitive load is not too high, then there may be some unused working memory available (Sweller et al., 1998). This could then be used by learners, with appropriate instructional design, to engage in conscious processing that helps in the construction of schemata in the particular domain of interest. This conscious processing is the germane cognitive load. An example is the use of part-complete solutions in the learning of problem solving (Paas, 1992; van Merrienboer, 1990; Van Merrienboer & De Croock, 1992). The studying of complete worked examples by students is seen as one way of reducing the extraneous cognitive load. When students have to complete an incomplete worked example then they have to "mindfully abstract" the schemata from the example in order to understand it. That is, they have to consciously process it and this increases the germane cognitive load.

A Learning Model for Programming

A learning model for programming has been designed that is firmly based on cognitive load theory. The use of worked examples in the learning of programming is sometimes known as the “reading” method and this method reduces the extraneous cognitive load on students. However it does not encourage students to think deeply about the examples being read and to “extract” the necessary schemata. However the “completion” method of learning programming requires students to have to complete part-complete programs thereby encouraging schemata production. The learning model makes use of the completion method and has the following attributes:

1. **Support for student centred learning**. Different learners gain expertise in programming at different rates. It is therefore important that the learning environment supports independence thereby allowing learners to construct knowledge within their own time frames. (eg. Jonassen, 1996). The completion method is a refinement of the reading method of learning programming and this method allows learners to work through appropriate materials independently and in a self-paced manner and also supports active learning with students having to engage with learning materials.
2. **Support for the creation of appropriate schemata and mental models**. The learning environment should support the creation and amendment of appropriate schemata that pertain to programming and also support the mental processing that needs to take place during this process (eg Winn, 1996). The completion method supports such schemata development in the form of stereotypical programming plans. The environment should also support the development of mental models of notional machine processing.

3. **Support for the reduction of extraneous cognitive load**. The learning environment should help reduce the extraneous cognitive load as programming is considered to have a high intrinsic cognitive load (eg Sweller, 1998). This is supported by using the reading method of learning programming.
4. **Support for the increase of germane cognitive load**. To promote programming skills, cognitive load theory suggests that a learning environment should encourage learners to mindfully abstract appropriate programming patterns (eg Paas, 1992). The removal of lines of code from complete programs increases the germane cognitive load on learners. The environment would also require learners to have to answer questions concerning their programming solutions.
5. **Support for the promotion of reflection and higher order thinking**. The development of problem solving skills in a specific domain of knowledge requires support for higher order thinking with learners being encouraged to reflect on their solutions to given problems (eg McLoughlin, 1997). The completion method reduces the amount of lower order thinking that is required and encourages more higher order thinking to take place.

COLORS for programming

COLORS (Cognitive Load Reduction System) for programming is a system that has been designed by the author to help students in their learning of programming at Edith Cowan University in Australia. It has been designed to try and meet the requirements of the learning model above and it has also taken into account a generic learning framework proposed by Oliver (1999). The framework is heavily influenced by his belief that constructivism best describes how learning takes place and it comprises three critical elements, these being: learning resources; learning activities; and learner supports as shown in figure1.

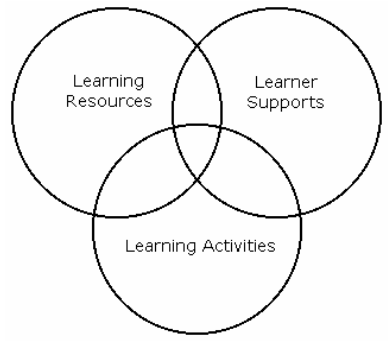


Figure 1: Oliver's Instructional Design Framework

The overall design of COLORS for programming will now be described with reference to this instructional design framework.

Learning Resources

Learning resources provide the content for a course and can be thought of as the materials which are used to help students construct their knowledge and meaning with respect to a domain of knowledge. Traditionally these resources have been available in the form of books and lecture notes and the move to flexible technology based systems has led to a lot of content being made available electronically. Unfortunately it has been estimated that many such systems are too content-oriented with 90% of planning and development being in content creation (Dehoney, 1999).

This emphasis within COLORS is the completion method of programming and so the content is provided by a programming textbook (Schneider, 2000) and the lecture notes delivered by the lecturer. It is recognised that on-line content and resources would be very useful to learners and is something that might be explored in the future. Typical content for programming courses includes descriptions of language syntax; data and control structures; descriptions of algorithms; descriptions of how to solve certain categories of problem; and example programs.

Learning Activities

Learning activities are the second element of the instructional design framework and play a fundamental role in determining learning outcomes (Wild, 1997). The activities determine how learners engage with the various materials and well designed activities can help reduce the extraneous cognitive load and stimulate the germane cognitive load.

The activities that are used within COLORS for programming comprise a set of programming problems and their part-complete solutions that need to be completed by a learner. The completion of a part-complete solution is done by selecting appropriate lines of code from a set of possible lines and placing them in the "correct" locations within the corresponding part-complete solution; and / or keying-in appropriate lines of code.

After "completing" a program, that program is tested in the programming environment of the particular language being used which, in this case, is Visual BASIC (VB).

There are also questions that learners are expected to answer in connection with the program that they have just completed. Such questions are another way of stimulating students to construct knowledge by applying another form of germane cognitive load.

These learning activities have been designed to be directly supported by a software tool, CORT (Code Restructuring Tool), that has been built by the author.

Learning Supports

Learning supports are the third element of the instructional design framework and can be thought of as the supports required to help guide and provide feedback to learners in a way that is responsive and sensitive to learner individual needs (McLoughlin, 1998). In "traditional" settings such supports have been provided by actively involved teachers (Laurillard, 1993) whereas in technology based learning environments, such supports are often known as "scaffolds" to help learners during their knowledge construction process (Roehler, 1996). In programming, an example of such a support is the facility that some programming editors have to help complete lines of programming code for the user as they are keyed-in. It is usually accepted that scaffolding is gradually reduced during learning, this process being known as "fading".

In COLORS for programming, some learning activities act as learning supports and so the boundary between activities and supports is somewhat blurred. There are several learning supports in COLORS, some of which are directly supported by CORT.

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

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

Method 2. All of the lines of code that are missing from the program plus some extra lines of code that are not needed to complete the program. These extra lines act as distracters.

Method 3. Zero or more lines of code that are missing from the program, other missing lines having to be keyed-in by the learner.

The important variable that affects which of the above methods should be used for a given problem is the degree of difficulty of that problem. For example if a problem is relatively simple then method 2 might be used, whereas method 1 might be used with a more difficult problem. Fading is not straight forward as the programming problems in latter the part of a programming course are usually more difficult than earlier ones and it might therefore still be necessary to use method 1 supports for some of the problems. CORT has been designed to provide a mechanism to easily manipulate the missing lines of code from a part-complete solution.

The second support provided by COLORS is a facility to easily move missing lines of code into a part-complete solution and within that solution. Such support has not been available in previous work with respect to the completion method and yet this is seen as important in helping reduce extraneous cognitive load.

The third support provided by COLORS is the provision, for each programming problem, of a screen image of the problem interface. The interface is the output "form" or window that is displayed to a user of a program when it is executed and includes the various objects such as buttons and text boxes. The screen image is also annotated with the internal names of the objects (i.e. the object names that are used within the programming code) thereby reducing the split-attention effect (eg. Chandler, 1991).

The fourth support provided by COLORS is the environment of the programming language itself. Many such modern programming environments, or integrated development environments, provide sophisticated facilities to help programmers debug their programs. These include the tracing, or step by step execution, of code and the ability to display the contents of variables. The language used is Visual BASIC (VB) which has excellent debugging facilities that can be used by novices in their learning of programming.

Other supports that are provided by COLORS include the "conventional" ones such as the provision of a tutor, other students, and a textbook. When campus based students require help in solving a programming problem, they might directly seek such help from their tutor or fellow students. With a flexible, technology based course that support would most likely be provided by email. Learners also look to their conventional textbook which, in addition to providing content, can also be considered to provide support.

Summary of COLORS for programming

The various components of COLORS for programming were developed from the learning model for programming using the completion method and from the instructional design framework proposed by Oliver (1999). The code restructuring tool, CORT, was designed to support certain features of COLORS for programming. Figure 2 summarises the development of COLORS and also those features supported by CORT.

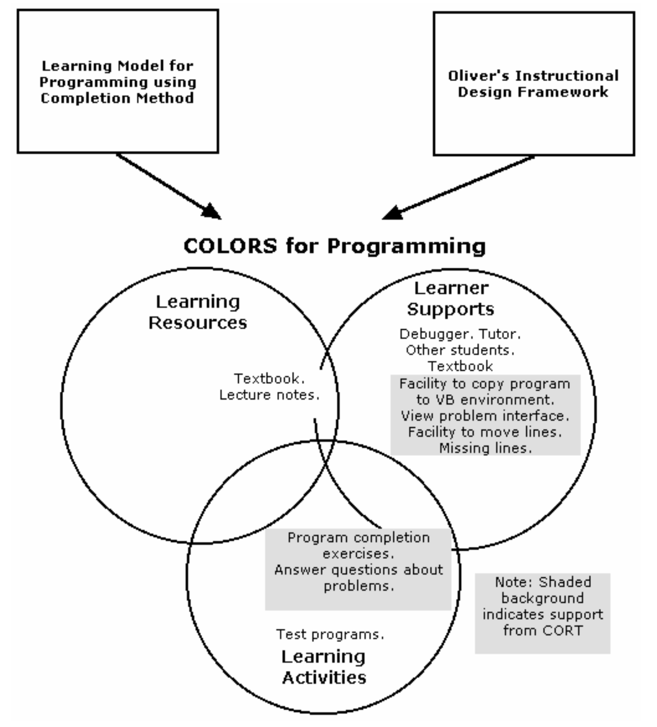

Learning Model for Programming using Completion Method
Oliver's Instructional Design Framework
COLORS for Programming
Learning Resources
Textbook. Lecture notes.
Learner Supports
Debugger. Tutor. Other students. Textbook
Facility to copy program to VB environment. View problem interface. Facility to move lines. Missing lines.
Program completion exercises. Answer questions about problems.
Test programs.
Learning Activities
Note: Shaded background indicates support from CORT

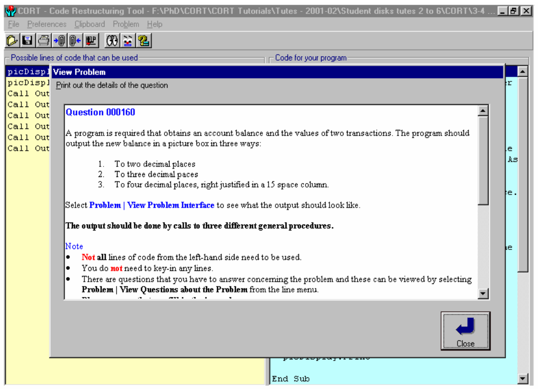
Figure 2: COLORS for Programming

CORT (Code Restructuring Tool)

CORT was designed to provide a basis for learning activities that in turn provide learner supports. CORT will be described from a learner's standpoint.

CORT Description: from a Learner’s Standpoint

1. The learner runs the CORT program and loads a “completion” problem. The problem description is displayed. The only actions that can be taken are to print out the description or to close the window. The problem description may have been given to learners in hard copy format.


CORT - Code Restructuring Tool - F:\PhD\CORT\CORT Tutorials\Tutes - 2001-02\Student disks tutes 2 to 6\CORT\3-4 ...
File Preferences Clipboard Problem Help
Possible lines of code that can be used
Code for your program
View Problem
Print out the details of the question
Close

Question 000160

A program is required that obtains an account balance and the values of two transactions. The program should output the new balance in a picture box in three ways:

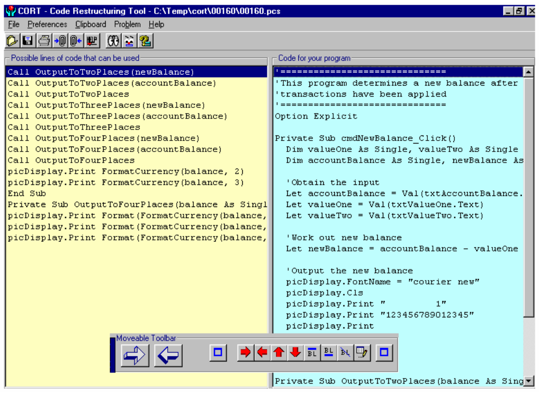
1. To two decimal places
2. To three decimal paces
3. To four decimal places, right justified in a 15 space column.

Select **Problem | View Problem Interface** to see what the output should look like.

The output should be done by calls to three different general procedures.

Note
- **Not all** lines of code from the left-hand side need to be used.
- You do **not** need to key-in any lines.
- There are questions that you have to answer concerning the problem and these can be viewed by selecting **Problem | View Questions about the Problem** from the line menu.

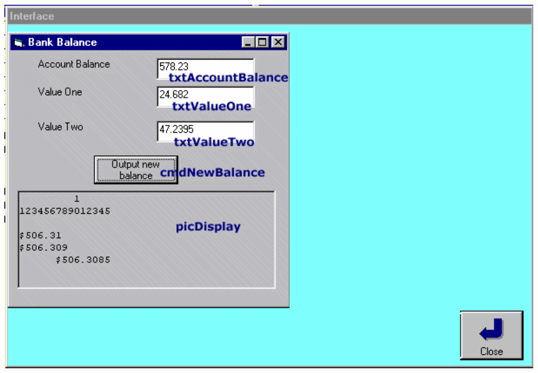
2. After closing the problem description window, two parallel windows can be seen. The right-hand window contains the part-complete solution to the problem and the left-hand window contains lines that can be used to complete the solution. These windows can be expanded and contracted horizontally so as to view the complete lines by clicking the corresponding □ button.


CORT - Code Restructuring Tool - C:\Temp\cort\00160\00160.pcs
File Preferences Clipboard Problem Help
Possible lines of code that can be used
Code for your program
Moveable Toolbar

```
Call OutputToTwoPlaces(newBalance)
Call OutputToTwoPlaces(accountBalance)
Call OutputToTwoPlaces
Call OutputToThreePlaces(newBalance)
Call OutputToThreePlaces(accountBalance)
Call OutputToThreePlaces
Call OutputToFourPlaces(newBalance)
Call OutputToFourPlaces(accountBalance)
Call OutputToFourPlaces
picDisplay.Print FormatCurrency(balance, 2)
picDisplay.Print FormatCurrency(balance, 3)
End Sub
Private Sub OutputToFourPlaces(balance As Singl
picDisplay.Print Format(FormatCurrency(balance,
picDisplay.Print Format(FormatCurrency(balance,
picDisplay.Print Format(FormatCurrency(balance,
```

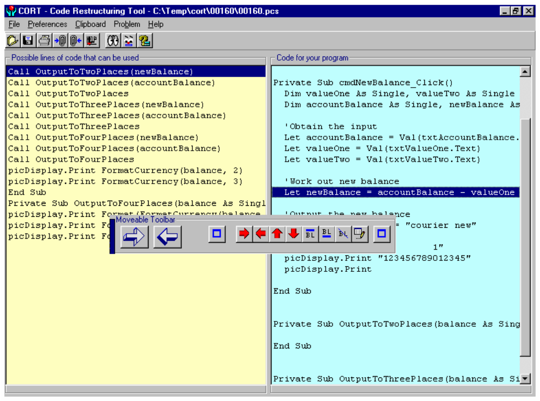
```
'==============================
'This program determines a new balance after
'transactions have been applied
'==============================
Option Explicit

Private Sub cmdNewBalance_Click()
  Dim valueOne As Single, valueTwo As Single
  Dim accountBalance As Single, newBalance As

  'Obtain the input
  Let accountBalance = Val(txtAccountBalance.
  Let valueOne = Val(txtValueOne.Text)
  Let valueTwo = Val(txtValueTwo.Text)

  'Work out new balance
  Let newBalance = accountBalance - valueOne

  'Output the new balance
  picDisplay.FontName = "courier new"
  picDisplay.Cls
  picDisplay.Print "         1"
  picDisplay.Print "123456789012345"
  picDisplay.Print

Private Sub OutputToTwoPlaces(balance As Sing
```

3. A learner can click on to view the problem interface. This is a screen image showing the expected output “form” for the problem that the learner is attempting. This image is annotated with the internal VB names of the objects. This again lowers the extraneous cognitive load by reducing the “split-attention” effect.



```
         1
123456789012345

$506.31
$506.309
     $506.3085
```

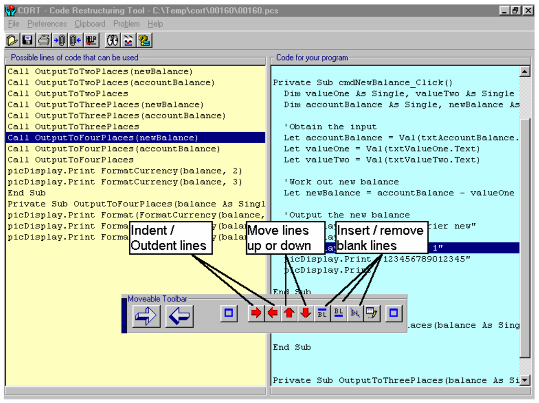
4. A line can be moved from the left to the right by: highlighting the line in the left-hand window; highlighting the line in the right-hand window after which the line from the left is to be placed; and clicking the button. Several lines can be highlighted in the left-hand window and moved in one operation. Lines can also be moved back into the left-hand window.


CORT - Code Restructuring Tool - C:\Temp\cort\00160\00160.pcs
File Preferences Clipboard Problem Help
Possible lines of code that can be used
Code for your program
Moveable Toolbar

```
Call OutputToTwoPlaces(newBalance)
Call OutputToTwoPlaces(accountBalance)
Call OutputToTwoPlaces
Call OutputToThreePlaces(newBalance)
Call OutputToThreePlaces(accountBalance)
Call OutputToThreePlaces
Call OutputToFourPlaces(newBalance)
Call OutputToFourPlaces(accountBalance)
Call OutputToFourPlaces
picDisplay.Print FormatCurrency(balance, 2)
picDisplay.Print FormatCurrency(balance, 3)
End Sub
Private Sub OutputToFourPlaces(balance As Singl
picDisplay.Print Format(FormatCurrency(balance
picDisplay.Print Fo
picDisplay.Print Fo
```

```
Private Sub cmdNewBalance_Click()
  Dim valueOne As Single, valueTwo As Single
  Dim accountBalance As Single, newBalance As

  'Obtain the input
  Let accountBalance = Val(txtAccountBalance.
  Let valueOne = Val(txtValueOne.Text)
  Let valueTwo = Val(txtValueTwo.Text)

  'Work out new balance
  Let newBalance = accountBalance - valueOne

  'Output the new balance
                                 = "courier new"

                                          1"
  picDisplay.Print "123456789012345"
  picDisplay.Print

End Sub


Private Sub OutputToTwoPlaces(balance As Sing

End Sub


Private Sub OutputToThreePlaces(balance As Si
```

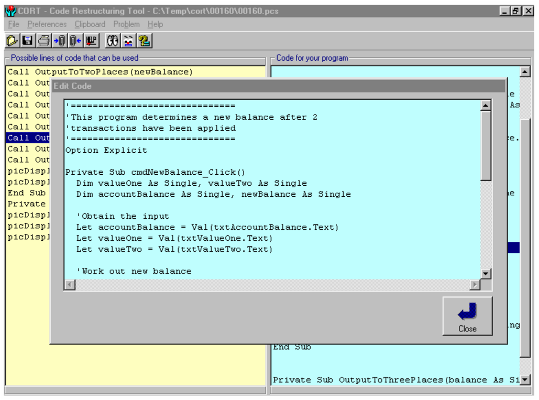
5. Lines can be rearranged in the right-hand window by moving them up or down. Lines can also be indented or outdented. Blank lines can be inserted before or after an existing line and blank lines can be deleted.


CORT - Code Restructuring Tool - C:\Temp\cort\00160\00160.pcs
File Preferences Clipboard Problem Help
Possible lines of code that can be used
Code for your program
Indent / Outdent lines
Move lines up or down
Insert / remove blank lines
Moveable Toolbar

```
Call OutputToTwoPlaces(newBalance)
Call OutputToTwoPlaces(accountBalance)
Call OutputToTwoPlaces
Call OutputToThreePlaces(newBalance)
Call OutputToThreePlaces(accountBalance)
Call OutputToThreePlaces
Call OutputToFourPlaces(newBalance)
Call OutputToFourPlaces(accountBalance)
Call OutputToFourPlaces
picDisplay.Print FormatCurrency(balance, 2)
picDisplay.Print FormatCurrency(balance, 3)
End Sub
Private Sub OutputToFourPlaces(balance As Singl
picDisplay.Print Format(FormatCurrency(balance,
picDisplay.Print Forma                y(balar
picDisplay.Print Forma                y(balar
```

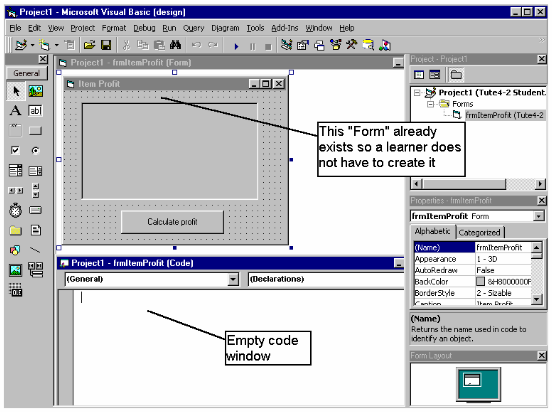
```
Private Sub cmdNewBalance_Click()
  Dim valueOne As Single, valueTwo As Single
  Dim accountBalance As Single, newBalance As

  'Obtain the input
  Let accountBalance = Val(txtAccountBalance.
  Let valueOne = Val(txtValueOne.Text)
  Let valueTwo = Val(txtValueTwo.Text)

  'Work out new balance
  Let newBalance = accountBalance - valueOne

  'Output the new balance
                                  ier new"

                                  1"
   icDisplay.Print "123456789012345"
   icDisplay.Pri

End Sub

                              aces(balance As Sing

End Sub


Private Sub OutputToThreePlaces(balance As Si
```

6. Lines of code can be keyed-in by learners using a simple text editor. This can be invoked by clicking on the button. After editing the program, the editor is closed by clicking the **Return** button and the changes are reflected in the original right-hand window.


CORT - Code Restructuring Tool - C:\Temp\cort\00160\00160.pcs
File Preferences Clipboard Problem Help
Possible lines of code that can be used
Code for your program
Edit Code
Close

```
'==============================
'This program determines a new balance after 2
'transactions have been applied
'==============================
Option Explicit

Private Sub cmdNewBalance_Click()
  Dim valueOne As Single, valueTwo As Single
  Dim accountBalance As Single, newBalance As Single

  'Obtain the input
  Let accountBalance = Val(txtAccountBalance.Text)
  Let valueOne = Val(txtValueOne.Text)
  Let valueTwo = Val(txtValueTwo.Text)

  'Work out new balance
```

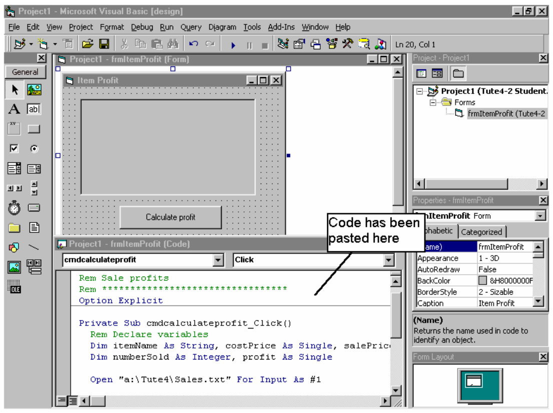
7. When a learner is ready to test their solution, they can click on the button and the code from the right-hand windows is pasted into the Windows Clipboard. They then run VB and open a file that contains the VB output "form" but does not contain any code. This figure shows an example VB "form" with an empty "code" window.


Project1 - Microsoft Visual Basic [design]
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
General
Project1 - frmItemProfit (Form)
Item Profit
Calculate profit
This "Form" already exists so a learner does not have to create it
Project - Project1
Project1 (Tute4-2 Student.
Forms
frmItemProfit (Tute4-2
Properties - frmItemProfit
frmItemProfit Form
Alphabetic
Categorized
Project1 - frmItemProfit (Code)
(General)
(Declarations)
Empty code window
Form Layout

(Name)	frmItemProfit
Appearance	1 - 3D
AutoRedraw	False
BackColor	&H8000000F
BorderStyle	2 - Sizable
Caption	Item Profit

(Name)
Returns the name used in code to identify an object.

8. A learner now pastes the contents of the Windows Clipboard into VB's empty code window by clicking the button. The program can then be run and / or traced in VB. After testing a program in VB, a learner can if necessary switch back to CORT and amend the solution, recopy the code and re-paste it into VB. This is an iterative process that is carried out until the program works to the learner's satisfaction.


Project1 - Microsoft Visual Basic [design]
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
Ln 20, Col 1
General
Project1 - frmItemProfit (Form)
Item Profit
Calculate profit
Project1 - frmItemProfit (Code)
cmdcalculateprofit
Click
Code has been pasted here
Project - Project1
Project1 (Tute4-2 Student.
Forms
frmItemProfit (Tute4-2
Properties - frmItemProfit
Alphabetic
Categorized
Form Layout

```
Rem Sale profits
Rem **********************************
Option Explicit

Private Sub cmdcalculateprofit_Click()
  Rem Declare variables
  Dim itemName As String, costPrice As Single, salePric
  Dim numberSold As Integer, profit As Single

  Open "a:\Tute4\Sales.txt" For Input As #1
```

frmItemProfit Form

(Name)	frmItemProfit
Appearance	1 - 3D
AutoRedraw	False
BackColor	&H8000000F
BorderStyle	2 - Sizable
Caption	Item Profit

(Name)
Returns the name used in code to identify an object.

Conclusions

COLORS for programming and CORT have been used with students at Edith Cowan University during semester 2, 2001, and data has been collected concerning its use as part of a research project. The data is of a qualitative nature, the data collection methods including questionnaires, on-line journals, observation, and interviews. The data has not yet been analysed however preliminary feedback suggests that students enjoy using the system and that cognitive load is reduced.

Some student comments include:

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

The data analysis that has yet to be undertaken will hopefully reveal interesting insights into the usefulness of the COLORS for programming system.

References

Carroll, W. (1994). Using worked examples as an instructional support in the algebra classroom. *Journal of Educational Computing Psychology, 86*, 360-367.

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

Chandler, P., & Sweller, J. (1996). Cognitive Load while Learning to use a computer program. *Applied Cognitive Psychology, 10*, 151-170.

Chase, W. G., & Simon, H. A. (1973). The Mind's Eye in Chess. In W. G. Chase (Ed.), *Visual Information Processing*. New York: Academic.

Dehoney, J., & Reeves, T. (1999). Instructional and social dimensions of class web pages. *Journal of Computing in Higher Education, 10*(2), 19-41.

Fowler, W. A. L., & Fowler, R. H. (1993). A Hypertext Approach to Computer Science Education Unifying programming Principles. *Journal of Multimedia and Hypermedia, 2*(4), 433-441.

Green, R. (1998). *Learning Programming through JavaScript.* Paper presented at the Australian Computers in Education Conference, Adelaide, Australia.

Jonassen, D. H., & Reeves, T. C. (1996). Learning with technology: Using computers as cognitive tools. In D. H. Jonassen (Ed.), *Handbook of research on educational communications and technology* (pp. 693-719). New York: Macmillan.

Kalyuga, S., Chandler, P., & Sweller, J. (1998). Levels of expertise and instructional design. *Human Factors, 40,* 1-17.

Laurillard, D. (1993). *Rethinking University Teaching: A Framework for the Effective use of Educational Technology*.: London Routledge.

McLoughlin, C. (1997). *Investigating conditions for higher order thinking in telematics environments.* Unpublished PhD, Edith Cowan University, Perth.

McLoughlin, C., & Oliver, R. (1998). *Scaffolding Higher Order Thinking In A Telelearning Environment.* Paper presented at the Ed-Media/Ed-Telecom 98 World Conference On Educational Multimedia And Hypermedia & World Conference On Educational Telecommunications, Virginia.

Miller, G. A. (1956). The Magical Number Seven, Plus or Minus Two: Some Limits on our Capacity to Process Information. *Psychological Review*(63), 81-97.

Oliver, R. (1999). Exploring strategies for on-line teaching and learning. *Distance Education, 20*(2), 240-254.

Paas, F. G. W. C. (1992). Training strategies for attaining transfer of problem-solving skill in statistics: A cognitive load approach. *Journal of Educational Psychology, 84,* 429-434.

Roehler, L. R., & Cantlon, D. J. (1996, May 10th 1996). *Scaffolding: A Powerful Tool in Social Constructivist Classrooms,* [HTML Document]. Available: http://www.educ.msu.edu/units/literacy/paperlr2.htm [1998, 3/5/98].

Schneider, D. (2000). *An introduction to programming in Visual BASIC 6.*: Prentice Hall.

Schneider, W., & Shiffrin, R. (1977). Controlled and automatic human information processing: Detection, search and attention. *Psychological Review, 84,* 1-66.

Simon, H., & Gilmartin, K. (1973). A Simulation of memory for Chess Positions. *Cognitive Psychology, 5,* 29-46.

Sweller, J., van Merrienboer, J. J. G., & Paas, F. G. W. C. (1998). Cognitive Architecture and Instructional Design. *Educational psychology review, 10*(3 Sep 01 1998), 251-296.

Tindall-Ford, S., Chandler, P., & Sweller, J. (1997). When two sensory modes are better than one. *Journal of Experimental Psychology: Applied, 3,* 257-287.

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-.

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

Wild, M., & Quinn, C. (1997). Implications of educational theory for the design of instructional multimedia. *British Journal of Educational Technology, 29*(1), 73-82.

Winn, W., & Snyder, D. (1996). Cognitive Perspectives in Psychology. In D. H. Jonassen (Ed.), *Handbook of research on educational communications and technology* (pp. 112-142). New York: Macmillan.

Biography

Stuart Garner is a member of the school of Management Information Systems within the faculty of Business and Public Management at Edith Cowan University in Perth, Western Australia. He teaches in the areas of systems and software development, and Web commerce development.