The Development, Use and Evaluation of a Program Design Tool in the Learning and Teaching of Software Development

Stuart Garner
Edith Cowan University, Joondalup, Australia

s.garner@ecu.edu.au

Abstract

The learning of software development is difficult for many students. Given a problem statement, students have to be able to: design a solution to the problem; implement a solution in a programming language; and test the solution. Often students miss out the design step and start writing programming code immediately. And yet instructors aim to encourage their students to develop a design in, for example, pseudocode. This helps students think carefully about their program designs without getting bogged down in the intricacies of a programming language. However students do not like writing pseudocode. Reasons for this include: it is another language to learn; they do not think that they are actually programming; they cannot test their designs as the designs are not executable; there is not a rigid syntax and so students are unsure whether their pseudocode meets an instructor's expectations. This paper concerns the development of a simple tool that helps students create pseudocode. The tool has been used and evaluated in an introductory programming unit of study. The results suggest that the tool was easy for students to use and that it helped support their learning.

Keywords: learning, programming, design, pseudocode, tools.

Introduction

This paper concerns the development, use and evaluation of a tool that helps novice programmers generate pseudocode designs for programming problems. It discusses: the difficulties of learning to program; what is meant by pseudocode; the development of the tool; and the evaluation of the tool with students.

Difficulty of Learning to Program

Learning to write computer programs is not easy (e.g., Scholtz & Wiedenbeck, 1992) and this is reflected in the low levels of achievement experienced by many students in first programming courses. 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 et al, 2000).

Programming is a complex process involving many steps (e.g., Winslow, 1996). The process comprises:

- Studying a given problem statement / set of requirements and producing an algorithm, often in pseudocode, to solve that problem;
- Translating the algorithm into the programming code of a certain programming language; and
- Testing and amending the program until it meets the original set of requirements.

Frequently students combine steps one and two above and attempt to produce algorithms in the programming language that is being utilised in their course of study rather than in a design language such as pseudocode. Many instructors do not encourage the use of pseudocode with students and the majority of introductory programming texts include few, if any, references to pseudocode.

What is Pseudocode?

Pseudocode is:

> *A detailed yet readable description of what a computer program or algorithm must do, expressed in a formally-styled natural language rather than in a programming language. Pseudocode is sometimes used as a detailed step in the process of developing a program. It allows designers or lead programmers to express the design in great detail and provides programmers a detailed template for the next step of writing code in a specific programming language* (TechTarget, 2005).

The following are two examples of pseudocode used to stipulate the algorithm for converting a Celsius temperature to Fahrenheit (Villanova University, 2005).

Example 1

```
prompt user for a celsius temperature to be converted
get input from user and store in a variable called celsiusTemp
calculate: (celsiusTemp * (5/9)) + 32 and store in a variable called
        fahrenheitTemp
display fahrenheitTemp
```

Example 2

```
BASE = 32
CONVERSION_FACTOR = 9/5
prompt user for celsius temperature
input celsiusTemp
fahrenheitTemp = (celsiusTemp * CONVERSION_FACTOR) + BASE
output fahrenheitTemp
```

As seen above, the algorithms differ in the syntax that is used to describe the particular algorithm. Robertson (2003, p.7) points out that:

> *There is no standard pseudocode at present. Authors seem to adopt their own special techniques and sets of rules, which often resemble a particular programming language.*

The fact that there is no standard is one of the reasons that many students bypass using it in their program designs and move directly to coding their solutions. Another reason for students disliking pseudocode is that the code is not executable and direct feedback of a design cannot therefore be obtained.

There are also reasons why some instructors do not use pseudocode. The first reason is that an instructor may have their own "standard" pseudocode syntax that they wish to use and yet that standard is probably different to the format used in the available textbooks. A second reason is that many of pseudocode languages do not reflect the event driven nature of programming that is in use today.

However research suggests that the use of pseudocode in the design of programs helps students in their learning of software development. Ramadhan (2000) synthesised certain empirical results (Gilmore & Green, 1988; Jones, 1984; Norman, 1983) and concluded:

- *The underlying programming constructs can be learned independently of the surface structure or syntax of the programming language; and*
- *When users are introduced to programming using a pseudocode language they perform more accurately during problem solving and their programming knowledge is more transferable to other programming languages, and the languages that involve simple and few programming concepts to be learned reduce the mental load on users and help them in the process of program understanding and debugging.*

Development of a Program Design Tool

A technology supported tool to help students create pseudocode for the design of programs was developed by the author. The main aims of the tool were that:

- It should be easy to use.
- It should include a library of "standard" constructs to reduce the amount of typing required. An example of such a construct is the IF/Then/Else/EndIf construct.
- There should be a facility to change the "standard" constructs so that an instructor or students can utilise a different syntax.
- There should be a facility to gradually add pseudocode constructs to the tool thereby allowing for a reduced cognitive load on students in the earlier stages of their learning.

The inspiration for the pseudocode tool came from a tool that is being used to help students learn scripting languages (Gibbs, 2002). The tool developed by Gibbs makes use of a freeware text editor named NoteTab (2005). NoteTab is much more than a text editor as it includes: a "clipbook" feature that lets a user create and organize textual clips; and macros that can range from text replacement to complete mini-applications that use a simple built-in scripting language.

Such a "clipbook" library was created by the author for use with NoteTab. The library was specifically designed to support the production of pseudocode for introductory programming in an event driven environment, eventual implementation being in Visual BASIC. The interface of NoteTab with the pseudocode library loaded is shown in Figure 1.

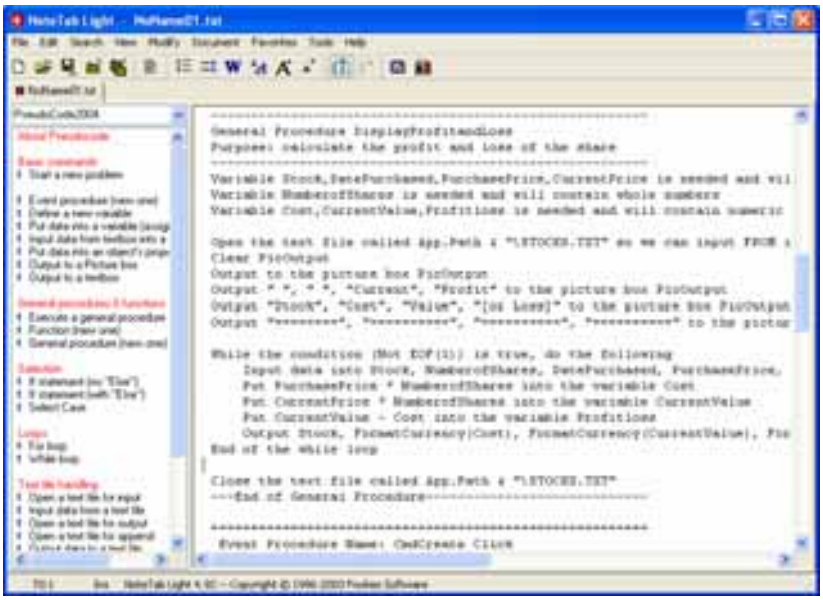


Figure 1: Program Design Tool Interface

The left-hand window contains the pseudocode library and the right-hand window contains an example of the pseudocode that can be "generated". When students use the system, they double click on an item in the library and a macro is executed. The macros prompt the user for relevant information, generates text and then places that text in the right-hand window. For example, double clicking on the library item "**Input data from textbox into a variable**" causes the dialogue box shown in Figure 2 to be displayed. Note that data has been entered into the dialogue box. When the "**OK**" button is clicked, pseudocode is generated as shown in Figure 3.

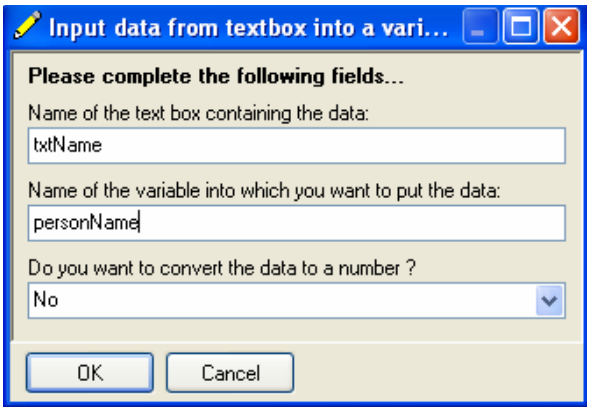


Figure 2: Dialogue Box

Figure 3: Generated Line of Pseudocode

The design tool supports the standard statements that are necessary to create algorithms that will eventually be implemented in a programming language including: input and output statements; assignment statements; the selection and control structures; procedures and functions; etc. A complete example of the pseudocode that was created by a student in an introductory programming class is shown below.

```
=======================================================
 Event Procedure Name: cmdClassify_Click
 Purpose: Classify the yob

 This code is to be executed when the Click event takes place
 on the object called cmdClassify
=======================================================

Variable personName is needed and will contain string data
Variable yob is needed and will contain whole numbers
```

```
Open the text file called YOB.TXT so we can input FROM it
Open the file called SENIORS.TXT so we can output TO it
Open the file called JUNIORS.TXT so we can output TO it
While the condition (it is not the end of file of YOB.TXT) is true, do the following
   Input data into personName, yob from the text file YOB.TXT
   If yob > 1940 Then
      Output the data personName, yob to the text file JUNIORS.TXT
   Else
      Output the data personName, yob to the text file SENIORS.TXT
   End If
End of the while loop
Close the text file called SENIORS.TXT
Close the text file called JUNIORS.TXT
Close the text file called YOB.TXT
```

There is also a library item "**Convert to VB Comments**" that, when double clicked, places a single quote in front of every line of pseudocode. The pseudocode can then be copied and pasted into a Visual BASIC program for self-documentation purposes.

The pseudocode shown above uses a syntax that has been created by the author. The aim of the particular syntax utilised is to help students gain a better understanding of what particular statements in an algorithm "do". A selection of statements, their Visual BASIC equivalent, and the rationale for the syntax chosen is shown in Table 1.

Table 1: Selection of Pseudocode Statements

Pseudocode Statement	Visual BASIC Statement	Rationale
Variable price is needed and will contain numbers with decimal places	Dim price As Double	There is a need for students to understand the types of data to be used in an algorithm
Place the data from the text property of txtPrice into price after converting it to a number	Price = CDbl(txtPrice.Text)	Students have problems understanding assignment statements and the need to change string data to numeric data
Put counter + 1 into the variable counter	counter = counter + 1	Students have difficulty with this type of assignment statement and its mechanism needs to be made explicit as to what is happening
While the condition (X = 9) is true, do the following End of the while loop	Do While X = 9 Loop	Loop mechanisms need to be made more explicit
For counter going from 1 To 5 in steps of 1 Next value of counter	For counter = 1 To 5 Next	
Open the text file called yob.txt so we can input FROM it	Dim sr As IO.StreamReader = IO.File.Opentext("yob.txt")	Some students have difficulties understanding the different modes that files can be opened in

As mentioned earlier in the paper, it should be possible to amend the macros so that the generated syntax meets the requirements of different instructors. The macros are programmed in an interpreted "Clip" language. Examples of the macros for two statements are shown in Table 2.

Table 2: Examples of the "Clip" Programming Language

Library Statement	Macro
Define a new variable	^!SET %variableName%=^?{What is the name of the new variable?=^%variableName%}; %variabletype%=^?{What type (kind) of data will the variable contain=String|Number (with decimal values)|Integer (whole number)} ;Do the testing ^!IF ^%variabletype% = "String" stringPart ^!IF ^%variabletype% = "Number (with decimal values)" numberPart ;Must be integer

	Variable ^%variableName% is needed and will contain whole numbers ^!GoTo end :stringPart Variable ^%variableName% is needed and will contain string data ^!GoTo end :numberPart Variable ^%variableName% is needed and will contain numeric data :end
Input data from textbox into a variable	^!Set %textBoxName%=^?{Name of the text box containing the data=^%textBoxName%}; %variableName%=^?{Name of the variable into which you want to put the data=^%variableName%}; %convertToNumber%=^?{Do you want to convert the data to a number ?=Yes|_No} ^!IF ^%convertToNumber% = "No" NoConvert Place the data from the text property of ^%textBoxName% into ^%variableName% after converting it to a number

The above table suggests that the "Clip" language is not the easiest to use, however it would be expected that a programming instructor should be able to make any necessary changes.

Initial Evaluation of the Program Design Tool

The tool has been used over a number of semesters in an introductory programming unit at an Australian university. The unit is within a major in Information Systems and the programming language used is Visual BASIC. An initial evaluation was conducted via a questionnaire with a group of 21 students and Table 3 contains some initial results together with their interpretation.

Table 3: Questionnaire Results and Interpretation

Question	Result	Comment
How easy was the pseudocode generator to use?	76% of students indicated that the pseudocode generator was straightforward or easy to use.	Any tool, especially one designed for novices, should be easy to use and have a small learning curve.
How often did you create the pseudocode before you created the Visual BASIC code?	67% of students indicated that they usually created the pseudocode before the Visual BASIC code.	It is well documented that a majority of students do not create pseudocode before they create their programming code. In this context, the result of this question is pleasing.
How often did you feel that the creation of pseudocode before the Visual BASIC code helped your understanding?	82% of students indicated that the creation of pseudocode usually helped their understanding.	This confirms the usefulness of good algorithm design that is encouraged by the use of the pseudocode generator.
Did a pseudocode statement such as: "Put 6 into the variable hours" help your understanding of how the assignment statement works?	76% of students indicated that the syntax of this statement helped their understanding of how the statement worked.	One of the aims of the pseudocode generator was to help students gain an understanding of the purpose and mechanism of statements required in computer implementable algorithms and this appears to have been successful.
Did a pseudocode statement such as: "Put the data "Bob" into the property named text of the object txtName" help you understand how assigning data to properties worked?	90% of students indicated that the syntax of this statement helped their understanding of how the statement worked.	
Did a pseudocode statement such as: "Execute the general procedure Add-Numbers passing arguments X, Y" help you understand how general procedures worked?	62% of students indicated that the syntax of this statement helped their understanding of how the statement worked.	
Would you have preferred NOT to use the pseudocode generator?	35% of students replied "Yes" to this question.	It was pleasing that around two thirds of students were comfortable using the generator. It still appears however that a sizeable minority prefer coding algorithms

Question	Result	Comment
		directly in a programming language.

Students were also asked to comment on the use of the pseudocode generator. The positive comments included:

> *It helped me go through all the things needed in the question.*
>
> *The pseudocode in general was very straightforward and easy to follow. The indentation of certain statements was useful as it reminded me to indent the statement when writing the actual Visual BASIC code.*
>
> *It helps you to think about the sequence of the program statements;*
>
> *It forces you to organise your thoughts in a clear logical manner. Without this I find I tend to go back and forth between deciding what I do and do not want in my code. The creation of the pseudocode gave me the necessary preparation time before commencing the programming task.*
>
> *Using the pseudocode helps in understanding what is needed to do in the VB code.*
>
> *It makes me think more carefully about the question and the variables I need for each question.*
>
> *Well... lets me organize a bit in computer instead of the brain. It reduced the typing work required.*

The comments strongly support the use of the pseudocode generator as a tool that helps in the design process of algorithms that can then be implemented in a programming language. It seems to have helped students think about the problems to be solved and organise their solutions and designs in a coherent manner.

There was one negative comment:

> *It is useless!! Because, if we get the pseudocode wrong, we will not know if the program is correct. Using VB directly, we can monitor our mistake.*

This comment concerns the fact that pseudocode is not executable and clearly this student believed that the cycle of coding and testing in a program development environment was more useful in his or her learning. This reflects the fact that some students always prefer to skip the careful thought processes required in program design and use trial and error techniques in their algorithm design.

Conclusions

This paper has discussed the design and use of a tool to support the generation of pseudocode for novice programmers. It has been used and evaluated with students in an introductory programming unit within an Information Systems major at an Australian university. The tool has been well received by students who believe that it is very useful in their learning and that is helps scaffold them in the program design process.

The results of the evaluation suggest that possible changes to the tool and its use might include:

- The actual programming language statements could be generated in addition to the pseudocode statements.
- A help file that includes information on all of the pseudocode statements could be included.

- A facility could be made available that allows students to easily change the wording, or syntax, of statements generated by the macro library. Such a facility should not require students to have to amend scripts that form the basis of the macro library.

References

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.

Gibbs, D. C. (2002). An interactive introductory programming environment using a scripting language. Paper presented at the *Ed-Media 2002*, Denver, Colorado.

Gilmore, D. & Green, T. (1988). Programming plans and programming expertise. *Journal of Experimental Psychology, 40A*(3), 67-82.

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.

Jones, A. (1984). How novices learn to program. Paper presented at the *First IFIP Conference on Human-Computer Interaction*, London.

Norman, D. (1983). Some observations on mental models. In D. Gentner & A. Stevens (Eds.), *Mental models* (pp. 34-39). NJ: Erlbaum.

NoteTab (2005). *NoteTab*. Retrieved November 16, 2005 from http://www.notetab.com

Ramadhan, H. A. (2000). Programming by discovery. *Journal of Computer Assisted Learning*, (16), 83-93.

Robertson, L. (2003). *Simple program design*. Thomson.

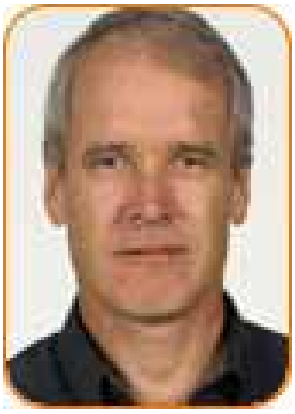
Scholtz, J. & Wiedenbeck, S. (1992). The role of planning in learning a new programming language. *International Journal of Man-Machine Studies, 37*, 191-214.

TechTarget (2005). *TechTarget*. Retrieved November 14, 2005 from http://whatis.techtarget.com/definition/0,,sid9_gci213457,00.html

Villanova University (2005). *Villanova University*. Retrieved November 14, 2005 from http://www.csc.villanova.edu/~map/1051/pseudocode.html

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

Biography



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

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