Teaching Novices Programming with Core Language and Dynamic Visualisation

Minjie Hu
Tairawhiti Polytechnic
PO Box 640, Gisborne, NZ
minjie@tairawhiti.ac.nz

This paper attempts to explore a new method to improve the teaching of computer programming for beginners at tertiary level. It begins with a change of approach to teaching and then to a mix of basic theory and practice. It then finds a core language, which simplified the scope of programme design and implementation. Dynamic visualisation technique was used as a supplementary tool in programme logic design, debugging and testing with desk-check, which helps novices understand their programme. Finally, the author expects programming educators to integrate and use these findings throughout the procedure of teaching programme development, from teaching approach, logic design, desk-check, implementation, to debugging and testing.

KEYWORDS

Core Language, Dynamic Visualisation, Software Visualisation, Programming Visualisation, and Algorithm Animation

1. INTRODUCTION

Teaching computer programming is always thought to be "hard" and a real challenge. Students feel (Jenkins, 2001) scared and try to avoid programming. Even if written in English, programming is believed to be as difficult to learn as a foreign language.

Why does it happen to this subject? How do we teach novices? What do we teach them? When we were students, how could we have been successful in programming? What is the way to improve our teaching?

The following research starts from an analysis of the current teaching approach on the first programming course for novice uses and later allows a mix of theory and practice rather than only syntax-free logical design. The research looks for a suitable first computer language for novices, which leads to the discovery of the minimal number statement of a computer language: a core language to teach novice programming. Next, the research introduces using visualisation techniques to assist programme logic design by the use of a simple flowchart tool. The research also finds debugger can be used as a simple programming visualisation tool to dynamically illustrate the execution of running-time code and data. This helps novices to test their desk-check solutions again running-time dynamic results. Finally, the integrated use of these findings is highly recommended to programming educators. It can be used throughout the whole procedure of teaching programme development from teaching approach, logic design, desk-check, implementation, to debugging and testing.

2. TEACHING APPROACH

Successful computer programming requires knowledge of abstract mathematical thinking and logical intelligence. It also requires problem-solving and critical thinking skills (Crews & Zieglger, 1998) as well as the same skills needed for language, arts and business (Prasad & Fielden, 2002).

Prasad and Fielden reported teaching programming usually starts off with an introduction to the program develop life cycle, then moves on to problem solving and algorithms and desk-check and then the translation of algorithms to a programming language. Finally, it completes with testing and documentation.

The introductory programming course for the Diploma in Information Communication Technology (DipICT) Level 5 at Tairawhiti Polytechnic used to start with module PD500 (Programming Developments, which is purely a syntax-free logic programming design), and then would implement the design

with VB language at module PP400 (Programming Principle).

Fincher (1999) summarised four approaches in teaching programming. The "Syntax-free" approach, like PD500, is to teach a quite separate notion (or pseudocode) from the real computer language. The "Literacy" approach, like module PP400, is based on a real computer language. The "Problem-solving" approach is used in conjunction with analysis and design. It sounds like the next step of previous two approaches, more like module PP510. The "Computation as Interaction" approach is for object-oriented and GUI programming.

Even some educators (Nelson & Rice, 2000) like to teach programming in a language-independent manner, they believe that it is important to let students execute their programme on computer so that they can receive feedback on what they have done. Thus, the introductory course starts in a language-independent manner, and then goes on to implement in real language. Crews (2001) argued pseudocode approach lacks real-time feedback to students and requires teacher's evaluation. Students have to spend more time on design than implementation details.

After several years teaching syntax-free programming, I re-considered the way I taught and the way I learnt two-decade ago. I tried to provide students with some optional practical workshops to implement their logic design with minimal syntax in Visual Basic (VB) language. The debugger is also applied to dynamically illustrate the execution of running-time code and data, which is described late in this paper. Students test their desk-check solution by following the running-time statement flow step-by-step. It dramatically changes the feedback from students. Students used to say it's too hard to understand, too abstract. Now they find practice lets them understand better, especially loop and selection.

A survey was conducted to DipICT students by the end of year 2003. The result shows most students (90%) like to learn theory and practical together when they learn programming. Only few students (10%) prefer to learn logical design first and then practice. One response said, "Students would be able to see the logic behind the paper work when using VB". Another response said, "I learned better by doing practical "hands on" stuff". However, another respondent was worried that the errors made in language might confuse the logic design. Thus, module PP490, a combination of programming theory and practice is introduced as the first programming course for novice this year together with further changes discussed as follows.

3. CORE TEACHING LANGUAGE

After the teaching approach is decided, the next thing is to choose an appropriate computer language for novices as their first computer language, as this could affect their learning of another computer language.

Microsoft Visual Basic is one of most popular computer languages for application development and programming education. But when educators (Nelson & Rice, 2000) used VB in their first course, they found students struggled to divide up their algorithm between various event-driven procedures. Thus, they had to turn back to using C++ again in the introductory course, using VB as a follow-on course. Although VB can be used to implement both simple and large programs, Raymond and Welch (2000) reported that it is difficult to see the flow of execution when designing an even-driven programme. They also found the VB environment to be too complex for novice to use. So, they chose Java for introductory course. However, other educators (Knowlton, 2002, Hu, 2003) also discovered that it is better to teach VB first and then teach C++.

Some educators tried to use a simple language called mini-language (Crews, 2001) for the novices. The mini-language such as Logo, Karel and Karel++ only provide only a few basic commands. But, a novice has to face special symbols and complex rules. Others use simple language called Kernel language (Roy and Haridi, 2002), which is modelled on a wide variety of languages. The kernel language approach focuses on programming concepts and helps students to design abstractions. It is truly language-independent.

To teach programming practice, we have to use a real computer language, which is simple and easy for novices. At Tairawhiti Polytechnic, Visual Basic 5.0 was first introduced to the Certificate in Introductory Computing (CIC), instead of command-based DOS version QBasic used in 1999. This is a free nine-week level-3 course. Most students have

Table_1 Comparison of Syntax between Pseudocode and Computer Language

Items	Pseudocode	VB6.0	VB.NET	C++	C#
Declare variables	No	Optional	Optional	YES	YES
Assignment statement	A = 5	A = 5	A = 5	A = 5;	A = 5;
Output statement	Print "The Answer is", A	Print "The Answer is", A	Label1.Text = "The Answer is " & A	cout<< "The Answer is "<< A <<endl;	Label1.Text = "The Answer is " + A. ToString ( );
Input statement	Input A	A = InputBox ("Enter a number")	A = InputBox ("Enter a number")	cout <<"Enetr a number"<<endl; cin >> A;	A = int.Parse (TextBox1.Text)

Selection statement	If A > 0 Then
	A = A + 1
	Else
	A = A- 1
	End If
		If A > 0 Then
		A = A + 1
		Else
		A = A- 1
		End If
			If A > 0 Then
			A+ = 1
			Else
			A - = 1
			End If
				if (A > 0)
				{A ++ ; }
				else
				{A --; }
					if (A > 0)
					{A ++ ; }
					else
					{A --; }

Pre-test Loops	While A < 10 A = A +1 End While	Do While A < 10 A = A +1 Loop	Do While A < 10 A = A +1 Loop	while (A < 10) { A++;}	while (A < 10) { A++;}
Post-test Loops	Repeat A= A+1 Until A > 10	Do A = A+1 Loop Until A > 10	Do A = A+1 Loop Until A > 10	do { A++; } while (A <=10)	do { A++; } while (A <=10)

a poor academic background and are novices in terms of programming. I tried not to put any control tools on VB form but only display results on the form. Students enjoyed the visual program rather than command-based interface. It was successful taught, even to the students who had no programming background or other training, like PD500. In the following year, the trimmed and pruned light Visual Basic was introduced to PP400 instead of C++ (Hu, 2003), and then moved on to add certain controls on window form.

The successful experience of this low-level class motivated me to consider using a core language for novice in PP490. Table_1 lists the comparison study of syntax between pseudocode and other currently used computer language, such as VB6.0, VB.NET, C++, and C#. A core language selected from VB 6.0 is the best choice amongst the other languages to implement pseudocode. The VB 6.0 core language runs on a window form application, rather than a console application. It is not necessary for novices to put any controls on the form. Therefore, it is used to simulate the traditional procedure programming, rather than event-driven programming, because it is the only one form-loaded event automatically driven when the programme starts.

There are a number of significant benefits for novices in using VB 6.0 core language. It is not necessary to declare variables with different data type. Neither has to convert a variable from one data type to another. The spelling of a variable name is not case sensitive. The beginners don't need to get frustrated when seeing Windows Form Designer generated code by VB.NET and C#. More so, it is easy to debug, which will be discussed later. The selected core language filled the gap between "Syntax-free" and "Literacy" approach. It also lets us focus our teaching on "Problem-solving" and try a little bit of "Interaction" through input box and window Form.

A related study has been conducted by Cilliers and Vogts (2002). They developed a simple programming interactive development environment called Delphi Lite, which was used as non-event-driven style with console applications for the first year course. They also found a significant difference in success rate by using Delphi Lite rather than the Delphi Enterprise version. However, the VB 6.0 core language runs on a window form application rather than a console application at command level user's interface. Since VB 6.0 core language is part of VB 6.0, we do not have to develop a new package like Delphi Lite. The main advantage of Delphi Lite is to use simple programming environment for non-event-driven style programming. In contrast, VB 6.0 core language is used for the purpose of simplifing the programming logic design with minimum syntax, in order to fill the gap between pseudocode and real

language. Novices can also implement a traditional procedure design on the window Form.

4. DYNAMIC VISUALISATION

While teaching programming, the old-fashioned teaching style (Mnuk, 2003) is still prevailing. Some source code fragment is written on the board or copied on a slide with OHP and a couple of pictures are drawn to illustrate how the programme works. They said: "A picture is worth thousands of words". This is because the human mind is very visually oriented (Hils, 1992). Pictures are easily learned, remembered and retrieved. After static visualisation teaching style of code list, flowchart, diagram, and pictures, educators introduced dynamic visualisation fashion in terms of one-way presentations, like movies, PowerPoint Slides, Flash animations with sound explanation and music. They also used two-way interactive tools, like executable flowchart, algorithm animation, programming visualisation and software visualisation. The following paper will discuss how to employ the interactive visualisation tool in programming teaching.

4.1 Visualisation with Flowchart

4.1.1 Iconic Programming Language

Educators (Calloni, *et al.* 1997) tried to use icons and lines to graphically build up programmes instead of traditional text-based statements for their first year programming course. They developed an iconic programming language named BACCII (Ben A. Calloni Coding Iconic Interface). But the icons are not executable until they generate programming code of real computer language such as Pascal and C++. Recently, Cilliers and Vogts (2002) also use another iconic-programming tool called B# for their first year programming course of Delphi. Although B# has a very limited scope of programming functionality, it is more helpful than Delphi Lite when faced with the logical errors.

4.1.2 Tools for Flowchart

Crews and Zieger (1998) reported students struggled with programming syntax but were in favour of a visual algorithm design environment using flowcharts. They found flowchart is easier to be understood than pseudocode and QBASIC. Some researchers (Hyrskykari, 1993) did not find any advantage to the flowchart over programme lists. Several experiments still support that flowcharts outperform with complex algorithms.

Furthermore, Crews and Zieger developed a Flowchart Interpreter (FLINT) System for students to develop flowcharts using an iconic interface. FLINT flowcharts are immediately executable. Students can get immediate feedback on their logical design before implementing with a real computer language. Also, Crews and Murphy (2004) evolved a Visual Logic Flowchart Simulation Tool. It combines the utility of flowcharts and pseudocode with computer simulation. Importantly it can be executed step-by-step in debugging, which provides immediate and accurate feedback to student. It also presents variable values observable to allow students to follow the effects of the execution of each statement. Finally, Visual Logic flowchart generates the latest Visual Basic .NET code. Figure_1 shows using Visual Logic to represent a flowchart. It illustrates the execution of the following chart step-by-step.

4.1.3 Adoption of Flowchart Simulation Tool

The code generated by Visual Logic is only for VB.NET console window application rather than for VB 6.0. The features of this flowchart simulation tool are suitable as a supplement to adapt to the teaching of flowcharts. The dynamic illustration by this executable flowchart assists novice to concentrate on problem solving and to understand programming logic design. However, to demonstrate the execution of real programming code step-by-step, we need to use another tool found in this research.

4.2 Programming Visualisation by Debugger

To teach programming, not only we need to teach how to design programme and write code to process data, but also to let students see, test and understand how the code executes and works with data.

4.2.1 Taxonomy of Visualisations

Programming visualisation is defined as (Price, *et al.* 1993) the use of various techniques to enhance the human understanding of computer programs, while visual programming is the use of "visual" techniques to specify a programme in the first place. Algorithm animation (or visualisation) is the visuali-

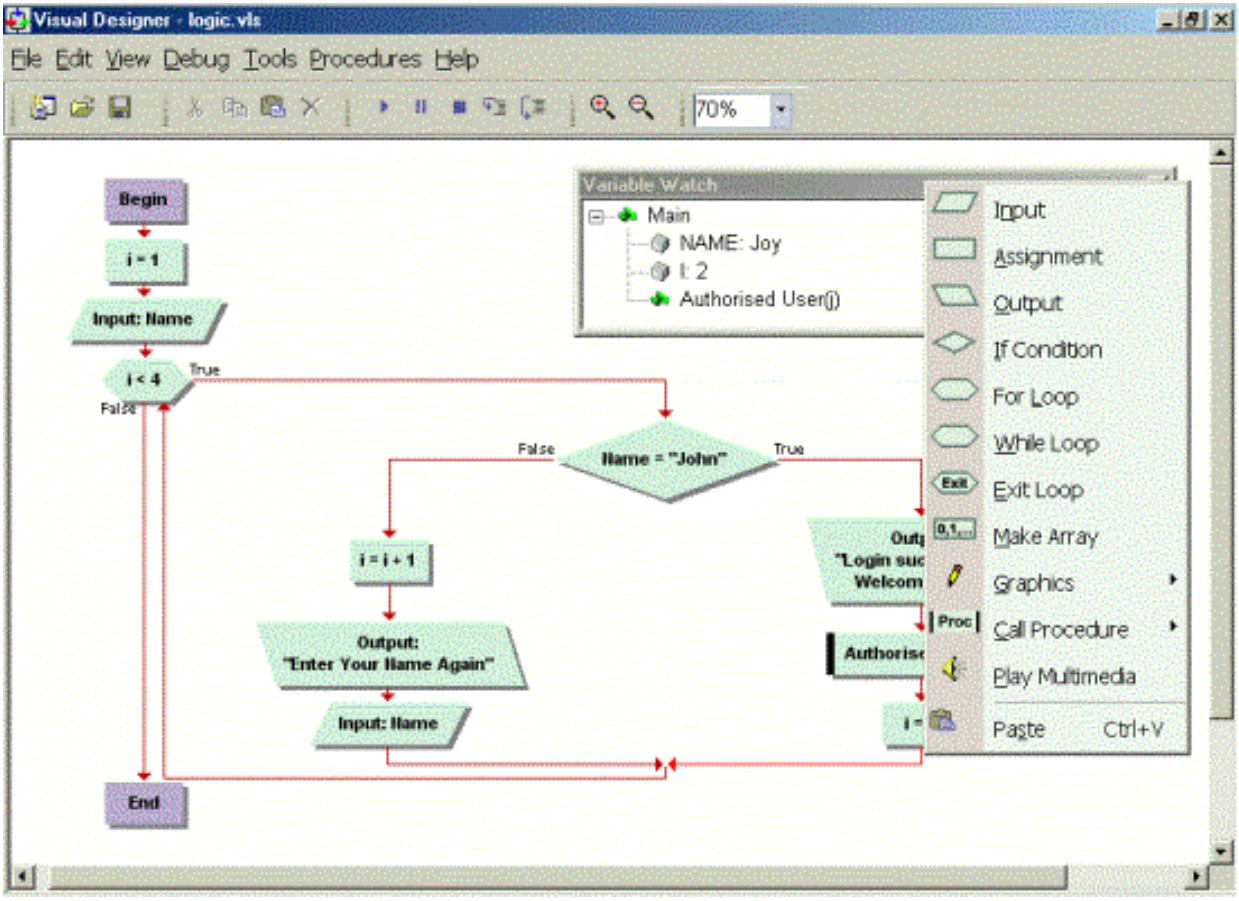

Figure_1 Example of Visual Logic Flowchart

sation of a high-level description of a piece of software while the low-level visualisation of code and data is also defined as a kind of programming visualisation. Software visualisation includes all of these.

Programming visualisation must not be confused with visual programming. Martin *et al.* (2002) argued visual programming means to construct a programme visually. This is similar as today we use controls from the toolbox of Visual Basic to build up user's interface by drag and drop. On the other hand, programming visualisation concerns the presentation of a programme. It is helpful in understanding a programme.

4.2.2 Algorithm Animation

It seems almost obvious (Kehoe *et al.*, 1999) that students could learn faster and more thoroughly with algorithm animation. The dynamic images provide a concrete appearance to the abstract notions, which could be helpful for students to understand algorithm. Unfortunately, the empirical study reported disappointing results. Lawrence et al. (1994) hypothesised that students, who created their own test data sets for the algorithm, could learn better than those who passively watch the animation.

The earlier algorithm animation focused on showing the results by images at high-level. It is hard to tell students how the code and data related to programme execution at a step-by-step low-level. Many available algorithm animation systems, such as Animal, EVEGA, and Polks do not support the visualisation of the source code (Mnuk, 2003). Although some system like Animated Algorithms supports visualisation of user-selected code, it is still difficult in incorporating changes and extensions.

4.2.3 Programming Visualisation

Smith & Webb (2000) conducted an experiment using a low-level program visualisation tool, called Bradman, in teaching novice C language programming. It gave concrete empirical evidence that such a tool is beneficial in assisting novices to learn programming. Sangwan et al. (1998) used a system of programming visualisation for standard C/C++ programs. Ben-Ari (2001) developed a programming visualisation tool, called Jeliot 2000, to assist students understanding Java source code of algorithm. However, most of current programming visualisation systems are designed either for particular computer language, such as C and Java, or only for cer-

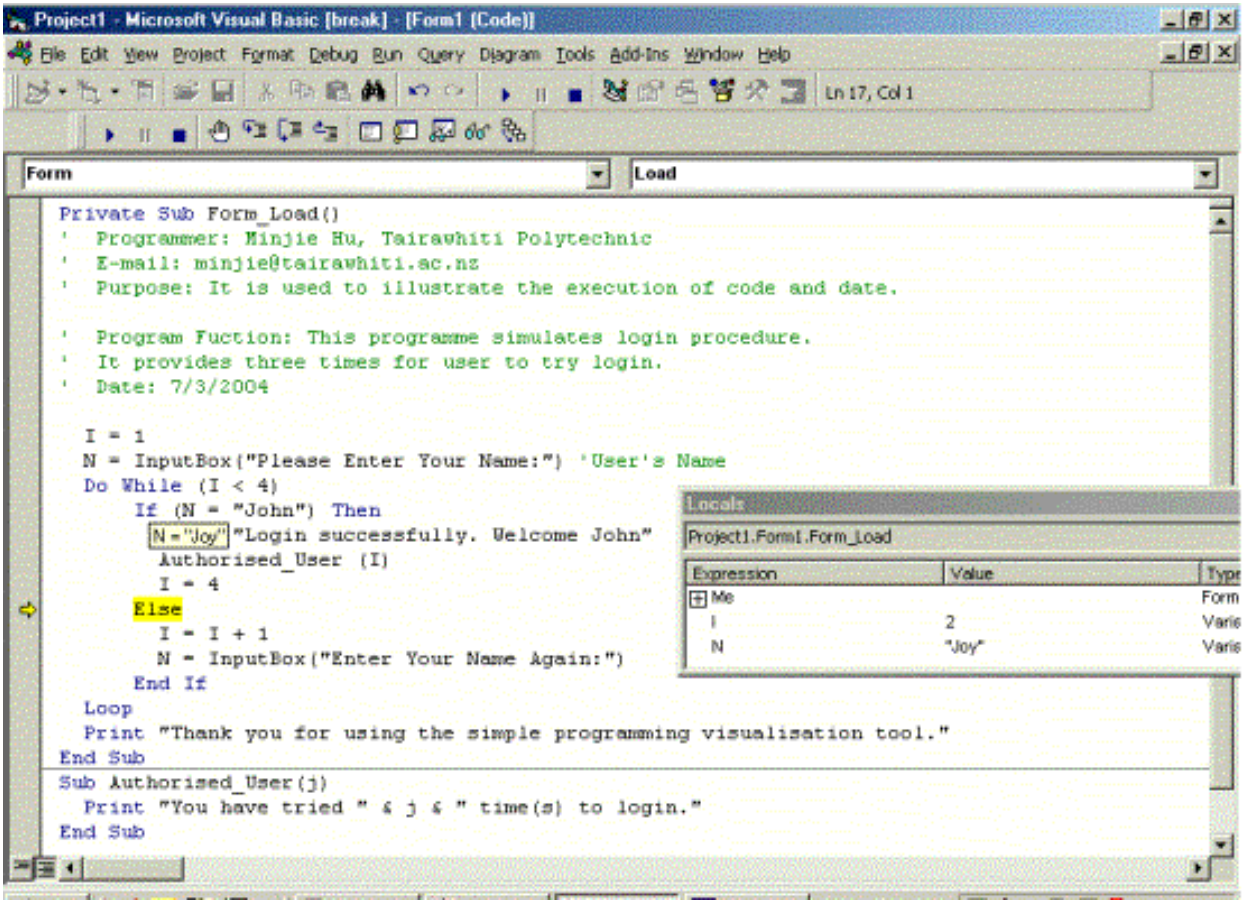

Figure_2 Example of Simple Programming Visualisation by "Step Into" Debug Tool

tain programmes of algorithms. Therefore, Robling and Freisleben (2001) tried to develop a tool, which is useful for generic software visualisation called Animal. It provides a language AnimalScript, which can be easily translated into other language. Unfortunately, AnimalScript only supports a limited number of operations.

Since no single visualisation system (Naps, 2002) is best for all learners, a research group suggested educators weigh carefully whether to adopt visualisation tools. Their survey indicated that more than a quarter of respondents never use visualisation tools for teaching. Most respondents use them once or twice for demonstrations during lectures. Only three respondents use them nearly every week. The survey also addressed the issue of what makes educators reluctant to use visualisation tools. Most respondents were concerned about the time taken to develop or to learn the new tool. They worried about the time needed to find good examples for the course content and the time to adopt visualisation to teaching approach. Meanwhile, this survey also indicated the significant benefits of using visualisations. Most respondents felt that their teaching was more enjoyable and had more fun for the students. Visualisation improved student motivation and leaning.

4.2.4 Find a Debugger for Dynamic Visualisation

I would rather find out some alternative features from existing languages than develop or look for a programming visualisation tool to use at a small polytechnic. The debugging techniques generally provide tools to help users analyse how execution flows from one statement to another, and how variables change as statements are executed. Particularly, the "Step Into" debug function in visual programming languages (such as Microsoft Visual Studio) can be used as a simple programming visualisation tool to execute each line of code step-by-step and to display the current value of variables in another Window. It dynamically presents both statement execution and data value. It is easy to use and perform. It can be used for a programme with any algorithm rather than only for that with particular pre-chosen algorithm. Meanwhile, there is no extra cost of selecting and installation.

With VB6.0 debugger, while each statement is executed in mode of "Step Into", it is highlighted

one-by-one which is controlled to pass to the next one by the user's command. At the same time, the running time status of local variables is displayed in the "Locals Window". Also, the value of each variable will prompt out when the mouse moves over the name of each variable. Figure_2 shows an example of programming visualisation by "Step Into" debug tool.

Using "Step Into" with VB6.0 is not only for the purpose of debugging the programme, but also for programming visualisation. It is using one stone to kill two birds. If we say that the selected core language approach simplifies the programming scope and the executable flowchart assists programme logic design, we could also say the debug tool helps novices to locate the bugs and understand the programme, particularly when novices compare their desk-check prediction solution again the running-time execution results.

However, it is not always so easy to perform programming visualisation by "Step Into" with the other languages. With Visual C++ debugger, "Step Into" jumps to display assembly language step-by-step on Disassembly windows. As well as displaying variables in Debug Variable Window, it also turns to step-by-step display C++ included system file, such as a header file and other C language code in different windows. Apparently, there is no benefit in showing novices low-level computer machine language code with every programme at an introductory course. But novices have to spend their time working through all the code sentence-by-sentence and window-by-window.

It does not seem an advantage for novices to use the current version of Microsoft Visual Studio rather than using the previous version. With VB.NET and C# debugger, although "Step Into" works with selected code, it has to firstly start from the code generated by Windows Form Designer. This may be many times longer than novice's programming code. The more controls on the Form, the more code it generated. On the other hand, with VB6.0 debugger, it gets into novice's code directly. Thus, the debug tool in VB6.0 is the best choice for a simple programming visualisation. This confirms the previous choice of selecting a simplified core language from VB6.0.

There is a related study conducted by Cross, *et al.* (2002) in teaching novice Java object-oriented programming. They use debugger of their jGRASP interactive development environment in their lecture presentation. They found the debugger could also be used to explain a correct programme to students additional to tracking down errors. However, we use debugger of VB 6.0 not only to visualise the code and data, but also to test the desk-check solution stepping into running-time results. It is also easy for novices to step through the simplified core language code by VB 6.0 debugger rather than to see extra lines of code and extra windows added by a certain language.

5. FURTHER TEACHING MODULES

Since the first introductory programming course at Tairawhiti Polytechnic has changed from PD500 to PP490, and now it follows the change from PP400 to PP590. Currently, the easiest and most productive programming languages are to turn to Microsoft Visual Studio.NET (Microsoft Corporation, 2003) for their rapidly building both Window and Web applications. Once students learn the basic programming skills from VB6.0, they will switch to VB.NET by adding controls on window form in the next module (PP590). Procedure and function will be also introduced. In the next module (PP510), array and file will be taught with C#.NET instead of C++. Application with database (MS SQL Server and Access), related files, object-oriented programming and Web programming will be introduced in the second year module (PR610) with VB.NET and/or C#.NET. However, using both Visual Logic flowchart and programming visualisation with debugger will still be helpful as a highlighted thread through the whole programming teaching. Table_2 presents the changes in programming course at Tairawhiti Polytechnic.

6. SUMMARY

It is hard to teach novice programming. Does the software visualisation technique make it easy? It depends on how we use them. A combination of theory and practice in teaching programming is the prelude for our teaching with dynamic visualisation. Selection of a core language makes programming simple and easy. This is the foundation for another teaching approach. An executable visualisation flowchart helps novices to learn programme logic design

Table_2 Changes in programming course at Tairawhiti Polytechnic

Changes	Module	Contents	Language/other
Before	PD500	♦ Principle of logic design ♦ Sequence, selection and loop ♦ Procedure and function	♦ Pseudocode ♦ Flowchart ♦ Desk check
	PP400	♦ Implement ♦ Test and documentation	♦ VB 6.0
	PP510	♦ Array ♦ Flat file	♦ Visual C++
	PR610	♦ Application with database (MS Access) ♦ Related files ♦ Object-oriented programming ♦ Script language	♦ VB 6.0, ♦ VBScript/JScript
After	PP490	♦ Principle of logic design ♦ Sequence, selection and loop ♦ Implement ♦ Test and documentation	♦ Pseudocode ♦ Flowchart ♦ Desk check ♦ VB 6.0
	PP590	♦ Procedure and function ♦ Implement ♦ Test and documentation	♦ VB.NET
	PP510	♦ Array ♦ Flat file	♦ C#.NET
	PR610	♦ Application with database (MS SQL Server and Access) ♦ Related files ♦ Object-oriented programming ♦ Web programming	♦ VB.NET/C#.NET

and gain immediate feedback. A simple programming visualisation by debugger pushes this research in teaching novice programming to a climax, as it shows how each statement flows and affects the value of data, and also how it provides evidence in detail as to whether the final results are the same as the novice's prediction by desk-check. It helps them to understand what they have done in programming. Moreover, these findings are a significant benefit thread for further programming teaching.

In a word, the findings of the research include the teaching approach with a mix of theory and practice: the language usage with a simplified core language and the supplement tool with dynamic visualisation by an adoption of executable flowchart and a debugger. One of them may only change a little of our teaching. Integrated use of these findings will affect the whole programming teaching procedure from logic design, desk-check, implementation to testing. In the other words, an appropriate mix of theory and practical application is maintained throughout teaching rather than "concepts first" (Ziegler & Crews, 1999, Crews & Murphy, 2004), teaching with flowchart for the first half of the semester and then move to programming language. For example, after teaching sequence with a flowchart and VB 6.0, starting to teach Selection with flowchart and VB 6.0 again, and we then teach Loop with flowchart and VB 6.0, etc. The following figures (Figure_3 and Figure_4) show the different teaching order. Finally, we should integrate these findings into every single programme logic from simple to complex step-by-step throughout the procedure of teaching programme development in terms of teaching approach, logic design, desk-check, implementation, debugging and testing.

An empirical study will be conducted for the series changes in teaching novice programming, particularly putting two modules of both PP490 and PP590 together to compare the teaching with the other two modules of both PD500 and PP400. Individual comparison is also needed for the following modules such as PP510 and PR610 to their previous teaching, in order to find out how the changes from the starting module affect the other subsequent programming modules.

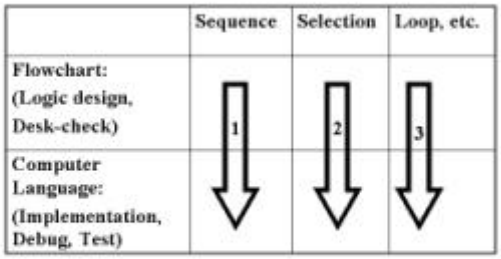

Figure_3 My teaching order

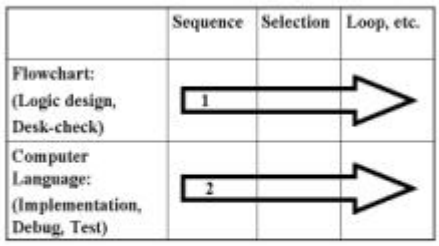

Figure_4 Another teaching order

REFERENCES

Cilliers, C. and Vogts, D. (2002) "Initial experience of delivery methods in a first year programming course", SACLA2002

Crews, T. and Murphy, C. (2004) Programming right from the start with Visual Basic.NET, Prentice Hall

Crews, T. (2001) "Using a flowchart simulator in a introductory programming course", <http://www.cstc.org/data/resources/213/Visual.pdf>

Crews, T. and Zieger, U. (1998) "The flowchart interpreter for introductory programming courses", Proceedings of the FIE'98 Conference, pp307-312

Cross II, J., Hendrix, D., Barowski, L. (2002) "Using the debugger as an integral part of teaching CS1", 32nd ASEE/IEEE Frontiers in Education Conference, 2002 IEEE

Fincher, S. (1999) "What are we doing when we teach programming?", 29th ASEE/IEEE Frontiers in Education Conference, 1999 IEEE

Hils, D. (1992) "A visual programming language for visualization of science data", Ph D thesis, University of Illinois at Urbana-Champaign

Hu, M. (2003) "A case study in teaching adult students computer programming", Proceedings of the NACCQ conference, NZ, pp287-290

Hyrskykari, A. (1993) "Development of program visualisation system", The second Czech British symposium of visual aspects of man-machine system

Jenkins, T. (2001) "Teaching programming: A journey from teacher to motivator", <http://www.ics.ltsn.ac.uk/pub/conf2001/papers/Jenkins.htm>

Kehoe, C., Stasko, J. and Taylor, A. (1999) "Rethinking the evaluation of algorithm animations as learning aids: An observation study", Technical Report GIT-GVU-99-10, Georgia Institute of Technology

Knowlton, T. (2002) "Using Visual Basic to prepare learners for C++", Tech tends - industry articles, Thomson Course Technology, <http://www.course.com/techtrends/VB_041801.cfm>

Lawrence, A., Badre, A., and Stasko, J. (1994) "Empirically evaluating the use of animation to teach algorithm", Technical report GIT-GVU-94-07, Georgia Institute of Technology

Martin, L., Giesl, A., and Martin, J. (2002) "Dynamic component program visualization", Proceedings of the Ninth Working Conference on Reverse Engineering (WCRE'02), IEEE

Microsoft Corporation, (2003) "Product Overview for Visual Basic .NET 2003", <http://msdn.microsoft.com/vbasic/productinfo/overview/default.aspx>

Mnuk, M. (2003) "How helpful are systems for algorithm visualization?", Technical Report TUM-I0303, Technology University Munich (Technische Universität München), German

Prasad, C. & Fielden, K. (2002) "Introducing Programming: Abalanced Approach", Proceedings of the NACCQ conference, NZ, pp101-108

Naps, T., Fleischer, R., McNally, M., RoBling, G., Hundhausen, C., Rodger, S., Almstrum, V., Korhonen, A., Velazquez-Iturbide, J., Dann, W., and Malmi, L., (2002) "Exploring the role of visualisation and engagement in computer science education", Report of the working group on "Improving the educational impact of algorithm visulisation", Proceedings of the Annual Joint Conference Integrating

Technology into Computer Science Education (ITiCSE), Denmark

Nelson, M. & Rice, D. (2000) "Introduction to algorithms and problem solving", 30th ASEE/ IEEE Frontiers in Education Conference, 2000 IEEE

Price, B., Baecker, R., and Small, I. (1993) "A principled taxonomy of software visualisation", Journal of visual language and computing 4(3): pp211-266

Raymond, D. & Welch Jr., D. (2000) "Integrating information technology and programming in a freshmen computer science course", 30th ASEE/IEEE Frontiers in Education Conference, 2000 IEEE

Robling, G. & Freisleben, B. (2001) "Software visualisation generation using AnimalScript", Informatik-Informatique, 2/2001

Roy, P. and Haridi, S. (2002) "Teaching programming broadly and deeply: The kernel language approach", Workshop on Functional and Declarative Programming in Education (FDPE02), 2002, Pittsburgh, USA

Sangwan R., Korsh, J., and LaFollette, P. (1998) "A system for program visualization in the classroom", Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education 1998, pp272-276, ACM Press

Smith, P. & Webb, G. (2000) "The efficacy of a low-level program visualisation tool for teaching programming concepts to novice C programmers", Journal of Educational Computing Research, 22(2): 187-216

Ziegler, U. and Crews, T. (1999) "An integrated program development tool for teaching and learning how to program", ACM SIGCSE Bulletin, 31(1): 276-280