Issues and Difficulties in Teaching Novice Computer Programming

I.T. Chan Mow
National University of Samoa
PO Box 1622
Apia
Samoa

Abstract: **The main argument promoted in this paper is that computer programming is a cognitively challenging subject and hence good instructional strategies are important in providing the student with optimal learner support. This paper examines some of the major issues in the instruction of computer programming as based on du Boulay's 1989 framework. For each area of difficulty, some potential solutions are proposed. Solutions are categorized as pedagogical, technological or content based. An attempt is then made to combine these potential solutions and best practices as found in the literature into a learning environment CABLE.**

I INTRODUCTION

Computer programming is a difficult and challenging subject area which places a heavy cognitive load on students ([1],[2]). After two years of learning programming, most novice programmers are still struggling to be proficient ([3],[4]). From an examination of current research in this field, it can be postulated, that one reason computer programming instruction seldom results in the successful transfer of problem solving skills lies in a lack of understanding about good instructional approaches in this direction ([5],[6],[7]). Hence, effective instructional strategies and optimal learner support mechanisms should be developed to provide learners of computer programming with the optimal learning environment that they need. The following section outlines the issues and difficulties in novice programming and the recommended solutions to these.

II ISSUES AND DIFFICULTIES IN NOVICE PROGRAMMING

The discussion on the nature and types of problems encountered in learning programming is based on areas of difficulty as identified by [8], but with some additional concerns as identified by the author such as the issue of resources. These areas are: (a) the cognitive requirements of programming; (b) syntax and semantics; (c) orientation; (d) the auxiliary skills needed for programming; and (e) resource constraints. It must be noted that these areas represent overlapping domains. Table 1 presents a summary of these five main areas of difficulty in learning programming.

A. Cognitive Requirements of Programming

The first area of difficulty in teaching programming relates to the cognitive requirements of learning programming. Learning to program is a cognitively demanding task ([9],[10],[11],[12]).

Table 1
Summary of the five main areas of difficulty in learning programming
Note: This categorization is based on du Boulay's framework, 1989.

Area of Difficulty	Dimensions/Aspects
Cognitive requirements of programming	Cognitive demand (Mental models)
	Precision
	Transition from novice to expert
Syntax and semantics	Closeness of mapping
	Consistency
	Error proneness
	Role expressiveness
	Secondary notation
	Hard mental operations
Orientation	
Auxiliary skills (Pragmatics)	
Resource constraints	

B. Cognitive Demand

Programming requires a programmer to hold a wide range of information in working memory. These include: (a) the details of syntax and semantics specific to the programming language used, (b) some mental model of how to solve the problem, and (c) the ability to differentiate between solving the problem and specifying the solution ([13],[14]).

C. Mental models.

Programming involves the construction of several mental models ([15]). Firstly, computer programs are written as a solution to some problem, and an understanding or mental model of this problem domain needs to be established before any attempt to write an appropriate program ([12]. Secondly, the programmer needs to construct a mental model of the "notional machine". "*The notional machine is an idealized conceptual computer whose properties are implied by the constructs in the programming language employed"* ([8]). Thirdly, the programmer must also develop a mental model of the program itself and how it will be executed ([11] [12]).

An effective instructional model should facilitate the building of effective mental models by strategies such as good modeling, coaching and scaffolding by an expert, the use of modeling tools and visualization techniques. Furthermore, multiple examples should be used to illustrate the behavior of programming models, so that students could use the examples to construct their own viable mental models ([15]). Constant repetition of examples illustrating a concept is needed for the construction of viable mental models. Another effective strategy, as an aid to the organization of knowledge, is the use of advanced organizers.

D. Cognitive load.

Programming requires the student to be proficient in the use of the development environment, mastering the use of the compiler, and interpreting responses such as result codes, error and warning messages. The need to hold this wide range of information in working memory, imposes a demanding cognitive load on the student ([16] [2]). Reference [2] proposes the use of instructional support materials such as worked examples, and partially complete solutions (implemented in part by a code restructuring tool) as strategies for alleviating mental load. The use of visualization techniques will also alleviate cognitive load. Examples include visual programming interfaces and visual debugging tools.

E. Psychological effects.

Learning to program is more difficult, complex and intimidating than simply using one [16]. Psychological factors which may affect student interaction with computers include: (a) unpleasant experiences with computers such as computer breakdown and data losses; (b) intimidating computer jargon or vocabulary; and (c) continually encountering problems that one is unable to solve, or even comprehend, sometimes leading to a state of "learned helplessness".

F. The Need for Precision

Another factor which contributes to the difficulty in learning programming is precision [17]. A high level of precision and specification must be maintained at all times, requiring a level of attention to detail that does not come naturally to human beings. At the same time, the programmer must deal with the limitations of language, machine, and compiler, all of which frequently require the inclusion of details that are not directly related to the problem, such as inclusion of libraries, and memory management routines.

An effective pedagogical model would emphasize the need for precision by extensive modeling, coaching and scaffolding with the use of multiple examples, and also the use of a debugger to identify program errors.

G. Differences between Novices and Experts

The learning process can be viewed as a process in where the learner progresses from a novice state to that of an expert. An investigation of the differences between novices and experts yields important information on how to teach programming to novices or beginners.

Firstly, novices have difficulty in recognizing incorrect grammar, and hence struggle with syntactic knowledge while experts readily recognize grammatical errors [18]. Secondly, in terms of semantic knowledge, experts have effective mental models of the virtual or notional machine, while novices have yet to build these models. Thirdly, in the area of schematic knowledge (knowledge of the structure of a program) experts use deep structure to categorize programs based on the type of routines required. In contrast, novices use superficial features for categorization.

Programming is described as the composition of plans or schemas [4]. Novices are inclined to use low-level plans, and are unskilled at problem decomposition. In contrast, experts keep an overall view of the problem in mind, while decomposing problems into small, manageable sub-problems. Experts also consider many more alternative solutions than novices, and are more adept at comparing different solutions and considering the merits of each [18].

H.. Notation: Difficulties with Syntax and Semantics

The second area of difficulty in teaching programming is the notation for representation of a program. Notation refers to the symbols of a programming language and the syntactic rules for combining them into a program [4]. Some of the dimensions or aspects of notation which affect programming ability are discussed below. These include: (a) closeness of mapping; (b) consistency; (c) error proneness; (d) role expressiveness; (e) secondary notation; and (f) hard mental operations. These will now be discussed in turn.

I. Closeness of Mapping

"Closeness of mapping" is a description of the closeness of the programming language to knowledge which students already possess, to the types of problems students should solve, and to the concepts students are attempting to acquire. A good instructional model would have techniques which would facilitate a closeness of mapping or a match between a student's existing knowledge, the problem which needs to be solved and the entities in a program. Useful techniques include the teaching of only a subset of the language, and the gradual expansion to include the full language as the novice programmer gains expertise in the use of the language [19], and the use of contextualized examples of programs in real world or applied situations.

J. Consistency

There are many forms of consistency, and programming languages require several of these forms in order to be maximally usable. A programming language needs to be consistent with: (a) what programmers already know, (b) within the languages own constructs, and (c) with domain knowledge and terminology [20].

A common strategy used in teaching programming is the use of analogies. However, novices frequently encounter difficulty with the limits of the analogy and mistakes arise out of attempting to extract too much from it. For example, with the command readln (*odd, even*) and the input values "2 3", they expect the computer to read the values 3 into *odd,* and 2 into *even* [21].

A common analogy used in the teaching of programming, is where a variable is described as a box for storage. Hence, users sometimes think that a sequence such as "*a*=2" followed by "*b=a*" means that *a* no longer holds the value 2.

Such inconsistencies can be addressed through teacher mediation in making students aware of the limits of the analogy, and by highlighting and correcting such misconceptions during the course of programming instruction.

K. Proneness to Errors

A common limitation of most programming languages is its proneness to errors. Examples of some error prone programming constructs and concepts include starting array indices at 0 rather than at 1, and in Java, the use of " =" to test for equality of reference rather than equality of value.

To address this common limitation, an effective pedagogical model should emphasize and highlight these common sources of errors so that students develop an awareness of potential errors. Again, the use of worked and multiple examples would be beneficial.

L. Role Expressiveness

"Role-expressiveness" is the degree to which information about the function of a construct can be determined from its structure or form. For example, the Java keywords *println* and *readline* used for printing output and reading in input, are highly role-expressive as their functions are obviously apparent.

However, Java also has some problems with role-expressiveness. For example, the meaning of the keywords *static* and *final*, for example, are not intuitively obvious especially to novice programmers [10].

An effective model of instruction should be able to support role expressiveness by explicit instruction of the meaning of each of the syntactic terms in a language. An example of this would be the use of articulation where an expert articulates the meaning of the terms in a program, and the novices learn by imitating the expert.

M. Secondary Notation

Secondary notation is information within the program text which is not part of the syntactic structure of the program but is useful for the readability and meaningfulness of the program [22]. Examples of secondary notation include comments, white space, indentation of code, and highlighting of key words. Furthermore, training of programmers is necessary in order for them to utilize secondary [22]. An effective learning environment should provide instructional training of novice programmers in the efficient use of secondary notation in order to enhance the readability and subsequent comprehension of programs.

N. Hard Mental Operations

The complexity of programming is a contributing factor to its challenging nature and "hard mental operations" refers to concepts which are complex and cognitively challenging [4] Examples of hard mental operations for novice programmers include iterations, recursion and optimization problems. Formulation of effective instructional strategies such as scaffolding, coaching, and modeling [25]. is needed to help students master such complex and mentally challenging operations.

O. Orientation

Orientation refers to the difficulties students have in recognizing and identifying what the term programming actually means, what processes programming actually entail, and what it is useful for. In recent times, the term programming has increasingly been used to refer to a range of activities which do not fall within the realm of computer science definitions of programming. Examples include data manipulation on spreadsheets, programming in HTML and even "programming" a microwave or a VCR. Such developments have compounded the problem leading to further confusion on what programming actually is [9].

Secondly, because of the way in which computer programming has been taught, and where the focus has only been on learning the syntax and semantics, students are not orientated in the context within which programming occurs and can be applied in real life.

An effective instructional model should address this by incorporating a good introductory section on what programming is, and the different "types" of programming which abound. Furthermore, there should be an attempt to relate programming to its application in software development and somehow avoid a common defect in the teaching of programming where programming is taught in isolation and not related to its real life applications.

P. Auxiliary Skills

The fourth area of difficulty in learning programming is the demand for additional or auxiliary skills. Auxiliary skills necessary for programming include proficiency in dealing with the development environment such as the operating system interface of the computer, and also other technical skills such as editing, compiling, and debugging a program [23].

Hence, instruction in computer programming must also include training or instruction in such skills as file management, editing, compiling, and debugging of program code.

Q. Resources

Other issues in teaching and learning novice programming include the lack of quality instructional resources and the lack of teachers with sufficient background in teaching object-oriented programming [24]. This is certainly a common situation in most tertiary institutions worldwide, and is certainly the case within our programming courses, at the National University of Samoa.

There is also debate over how the object-oriented paradigm should be introduced, with such issues as "objects early", "objects late" being contested in the literature. However, studies seem to indicate that the pathway of teaching object-oriented languages by firstly teaching procedural programming is ineffective and hinders student uptake of object-oriented constructs ([4]; [23]). This then makes the case for teaching object-oriented programming at the introductory level, and also the suitability of the use of pure object-oriented languages such as Java

III ESTABLISHMENT OF A LEARNING ENVIRONMENT

The main argument that this discussion is based on is that an effective learning environment is crucial for improving student performance in computer programming. The four dimensions which comprises any learning environment are content, method, sequence and social context [26].The discussion so far has proposed solutions focusing on methods or pedagogies. However another important dimension of a learning environment is content.

A. Content

Content refers to the different or distinct types of knowledge required for expertise. Cognitive research distinguishes between four types of content knowledge: (a) domain knowledge, (b) heuristic strategies, (c) control strategies, and (d) learning strategies.

Domain knowledge refers to the concepts, facts and procedures in a particular subject area, usually found in textbooks, lectures and demonstrations. For example, for this research, this will be concepts facts and procedures associated with Java programming. When domain knowledge is taught outside its problem solving context, this knowledge remains inert and transfer of knowledge does not eventuate. Hence for this research the course reader was rewritten so that domain knowledge was taught within context of application. E.g., programming taught within the context of SDLC.

Heuristic knowledge or strategies are "tricks of the trade" or "rules of thumb" which are generally effective in solving problems and for accomplishing tasks. Examples to be used in the proposed environment are rules for instantiating objects in a Java program, trouble shooting procedures in debugging Java programs.

Control strategies is knowledge used when there is a need to decide between a range of possible strategies and to decide when to change strategies. Examples for the proposed environment include the use of plans, comparisons of different implementations, the use of trace tables for checking the execution and outcome of a program, and the use of a debugger to identify program errors.

Learning strategies refer to strategies used to acquire new concepts, facts, and procedures in their own or another field. Learning strategies can either be general or domain specific[26]. General learning strategies used in the proposed environment are collaborative learning, engaged learning, and situated learning.

Domain specific strategies include the use of "think alouds" for articulating the steps in instantiating objects, UML modeling, stepping through the program using the debugger, and using the visual representation of the variable stack within the debugger, to trace the values of variables, as the program code executes.

B. Technological solutions

E-learning is defined as a learning approach implemented by the use of a range of technologies such as pcs, CD-ROMs, mobile devices, Internet, and computer-mediated communication (CMC) techniques (e-mail, discussion forums, chat rooms and collaborative software).

The three main advantages to online learning are that students (a) can choose their own pace of study, (b) are able to organize their own schedule and not be tied to time and place, and (c) can choose a pace of study independent of others[28].

Advantages of computer mediated communication (CMC) in e-learning are as follows [29]. Firstly with CMC learners acquire knowledge within the context in which it is used. Secondly the use of CMC eliminates the problem of transfer of knowledge from the context of learning to the context of practice – situated learning. The third advantage is the motivation provided by the presence of a diverse real audience instead of just the teacher as in a conventional classroom[29]. Yet another advantage of CMC is the provision for a more diverse range of people to interact.

The main argument promoted in this paper is that an effective learning environment is crucial for improving student performance in computer programming. It is appropriate that content, and pedagogies recommended in the literature be combined with online technologies to produce an effective instructional model which could enhance the teaching of computer programming. Table 2 summarises the issues in teaching programming and potential instructional solutions.

C. Proposed Learning environment Model

The proposed learning environment is a combination of variety of contemporary best practices which makes use of cognitive apprenticeship, collaborative learning, metacognition, and technologies via the use of tele-apprenticeship and online or computer-mediated communication (CMC). The main driving assumption of this proposed learning environment is cognitive apprenticeship hence the name Cognitive Apprenticeship Based Learning Environment or CABLE.

Cognitive apprenticeship is directed at teaching processes that experts use to handle complex tasks, and is characterized by a number of teaching methods[4]. These include modeling, scaffolding, coaching, articulation, reflection and exploration [27].

Table 2
Table of programming issues and possible instructional solutions

Issues in teaching programming	Potential instructional solutions
Cognitive requirements of programming	
Cognitive demand	Visualization (debugger) UML modeling Coaching Modeling Scaffolding Reflection Articulation
Precision	Scaffolding Coaching Modeling Debugger
Transition from novice to expert	Scaffolding Coaching Modeling Reflection Articulation
Syntax and semantics (Notation)	
Closeness of mapping	Situated learning
Consistency	Scaffolding Multiple examples Coaching Modeling
Error proneness	Scaffolding Coaching Worked and multiple examples
Role expressiveness	Articulation
Secondary notation	Coaching in the use of secondary notation
Hard mental operations	Scaffolding Modeling Coaching Reflection
Orientation	Situated learning Good introductory section on basic programming concepts
Auxiliary skills	Coaching in file management, compiling, debugging
Resource constraints	Construction of a reader Online notes

Another instructional strategy that is gaining prominence as an effective teaching method and is integrated into the learning environment being trialed is collaborative learning.

The starting assumption is that the computer can be used as a tool assisting in both cognitive apprenticeship and collaborative learning. An important element of the CABLE learning environment is its use of computer mediated communication techniques for implementing aspects of cognitive apprenticeship such as scaffolding, coaching, feedback and modeling [28]. Computer mediated communication techniques to be utilized in implementing tele-apprenticeship in the proposed learning environment include email, online notes, interactive tests and a bulletin board.

Hence CABLE is a hybrid model which uses both face-to-face and online mode of delivery. The components of CABLE are as follows:

- Cognitive apprenticeship
  - Modeling
  - Coaching
  - Scaffolding
  - Articulation
  - Reflection
- Computer-mediated communication
  - E-mail
  - Bulletin board
  - Online resources
- Collaborative learning

The transition from theoretical model to practical approach is made with the definition of the implementation details of the CABLE approach. The actual trialing of the CABLE environment is the subject of other papers ([30],[31]). In summary the present paper examines issues of teaching and learning novice computer programming, proposes instructional, content and technology based solutions and then combines these into the proposed learning environment CABLE with its key components of cognitive apprenticeship, collaborative learning implemented both in face to face and online mode.

REFERENCES

[1] O.Astrachan, & T. Selby ,J. Unger , An Object-Oriented, Apprenticeship Approach to Data Structures using Simulation , paper in proceedings of FIE ‘96, Frontiers in Education, 2006

[2] S.Garner, Cognitive Load Reduction in Problem Solving Domains, Edith Cowan University, 2000.

[3] AECT 2001. The handbook of Research for Educational Communications and Technology. In URL: http://www.aect.org/intranet/publications/edtech/24/24-05.html [accessed June 12th 2003]

[4] D.G.Moursound, Increasing your expertise as a problem solver: Some roles of computers. Eugene, OR: ISTE. Copyright (C) David Moursund 2002.

[5] W.Au, Logo programming: Problem solving and instructional methods. Unpublished doctoral dissertation. Palmerston North, New Zealand: Massey University, 1999.

[6] A. Blair. & T. Hume.T.An Exploration of the Application of Constructive learning Techniques to Software development using Object orientation as a Vehicle. Paper, presented at CTI Annual Conference, 1994. In URL : http://www.ulst.ac.uk/cticomp/therhume.html, retrieved March 12th , 2003

[7] J.Tholander., K. Karlgren, R.Ramberg, Cognitive Apprenticeship in Training for Conceptual Modeling, 1998 in URL

[http://www.dsv.su.se/~klas/Publications/webnet98.pdf] accessed July 15th 2003

[8] J.B.H. du Boulay, Some difficulties of learning to program. In E. Soloway and J.C. Spohrer, (Eds.), Studying the Novice Programmer (pp. 431-436) Hillsdale: Lawrence Erlbaum Associates, 1989.

[9] A. Blackwell, What is programming? In J. Kuljis, L. Baldwin & R. Scoble (Eds.), *Proceedings of the 14th Workshop of the Psychology of Programming Interest Group,* Brunel University, June 2002. Retrieved April 4th, 2005 from http:// www.ppig.org.

[10]. M. Kolling, M.. The problem of teaching object-oriented programming. *Part II: Environments*, 1999. Retrieved November 20, 2004, from http://www.mip.ou.dk/~mik/papers/oo-environments.pdf.

[11]. J. F. Pane, & B.A.,Myers, *Usability Issues in the Design of Novice Programming system,* Carnegie Mellon University, School of Computer Science Technical Report CMU-CS-96-132, Pittsburgh, PA, August 1996.

[12] A. Robins, J, Rountree, & N. Rountree, N.. Learning and teaching programming: A review and discussion. *Computer Science Education*, *13*(2),137–172., 2003

[13] H. Balzert, . E-Learning Platform, W3L, *Wirtschaftsinformatik 46, 128/129*, 20-31, 2004.

[14] Rist, R. S. (1995). Program Structure and Design. *Cognitive Science, 19,* 507-562.

[15] Lui, A. K., & Kwan, R., Poon, M., & Cheung, Y. H. Y. (2004). Saving Weak Programming Students: Applying Constructivism in a First Programming Course. *SIGCE Bulletin, 36*(2).

[16]Diethelm, I., Geiger, L., & Zundorf, A. (2005). *Teaching Modeling with Objects First.* Retrieved July 26th, 2005 from[http://www.se.eecs.uni-kassel.de/se/publications/DGZ05.pdf].

[17]Blackwell, A. (2001). *First Steps in Programming*: *A Rationale for Attention Investment Models.* Presented at IEEE Symposia on Human-Centric Computing Languages and Environments. Arlington, VA, 2-10.

[18]Lahtinen, E., Ala-Mutka, K. & Jarvinen, H. M. (2005). *A Study of the Difficulties of Novice Programmers:* Proceedings of ITiCSE, 2005 (pp. 14 – 18). NewYork NY: ACM.

[19] Brusilovsky, P., Kouchnirenko A., Miller P., & Tomek, I. (1994). *Teaching programming to novices: A review of approaches and tools.* In T.Ottman, I. Tomek (Eds.) Proceedings of ED-MEDIA'94 – World Conference on Educational Multimedia and Hypermedia (pp 103-110). Vancouver, Canada.

[20] L.McIver, *Syntactic & Semantic Issues in Introductory Programming education*. Unpublished doctoral dissertation, Monash University: Clayton, Victoria, 2001.

[21] D. Sleeman,, R.T. Putman, J. Baxter, & L. Kuspa. An introductory Pascal class: A case study of students' errors. In R. E. Mayer (Ed.) Teaching and learning computer programming (pp 207-235). Hillsdale, NJ: Lawrence Erlbaum Associates, 1988.

[22] T. R. G . Green, M. Petre, & R.K.E. Bellamy, *Comprehensibility of Visual and Textual Programs: A Test of Superlativism Against the 'Match-Mismatch' Conjecture.* Empirical Studies of Programming: Fourth Workshop. (pp. 121-146).Norwood, NJ: Ablex,1991.

[23] M. Pedroni, *Teaching Introductory Programming with the Inverted Curriculum Approach,* Diploma thesis, Department Computer Science, ETH Zurich, 2003. [Electronic Version].

[24] R.Bruhn, & P. Burton, Teaching programming in the OOP Era. *The SIGCSE Bulletin, 35*(2*),* 111-114, 2003.

[25] J.C. Winnips, *Scaffolding by design. A model for www learner support.* Unpublished PHD dissertation*,* University of Twente, 2001.

[26] A.Collins, J.Brown, & A.Holum, *Cognitive apprenticeship: Making Thinking Visible, 1991.* Retrieved March 20, 2004, from http://www.sapio.org/demo/cognitive_apprenticeship.htm

[27] S. Jarvela, The Cognitive Apprenticeship Model in a Technologically Rich Learning Environment: Interpreting the Learning Interaction. *Learning and Instruction 5*(3), 237-259, 1995.

[28] J. Levin, & M.Waugh,Teaching teleapprenticeships: Electronic network-based educational frameworks for improving teacher education. *Journal of Interactive Learning Environments, 6*(1-2), 39-58, 1998.

[29] J.Levin, *A 2020 Vision: Education in the next two decades,* 2002. Retrieved January 18, 2004, from http://w3.ed.uiuiuc.edu/faculty/J-Levin/2020-vision.html.

[30] I.T. Chan Mow, W.K. Au,and G.Yates, "The impact of the CABLE approach in teaching computer programming" , 2006.

[31] I.T. Chan Mow, W.K. Au,and G.Yates, "The impact of the CABLE approach in teaching computer programming" , 2004