Making Software Design Applicable

John McPhee

Christchurch Polytechnic Institute of Technology
Christchurch, New Zealand

mcpheej@cpit.ac.nz

Proceedings of the 15th Annual NACCQ, Hamilton New Zealand July, 2002 www.naccq.ac.nz

ABSTRACT

Entry-level programming courses typically use simple application exercises as a practical teaching tool. Applying a design methodology may increase complexity thus is ignored. The dilemma though is that leaving out any explicit design methodology will also reinforce the perception that design is unnecessary.

This investigation looked at how teaching and strongly encouraging the use of a simple and relevant design methodology, the Coats-Mellon Operational Specification, impacted upon the student's attitude towards design.

1. INTRODUCTION

It is generally accepted amongst Software Engineers that using some form of design process is essential. Modelling the system helps reduce complexity, identify design flaws, and document the decisions made. A model acts as a communication medium between all stakeholders helping to identify and minimise major risks early in the process. This paper investigates a method of introducing a design methodology in an introductory programming course.

There is some research to indicate that Industry also values software design skills. (Rahman *et al.* 1999) found that the Business Community rated software design skills as more important than technical skills for entry-level programmers.

1.1 DESIGN REQUIRED IN LEARNING PROGRAMMING

A considerable amount of research has been carried out on the issue of how students learn programming skills.

A common finding is that experts form abstract, conceptual representations of problems, but novices form representations that tend to retain the surface elements of the problem. One study goes on to conclude that representing problems in terms of several elements describing what is to be done, allows the programmer to change an element in isolation without exceeding the limits of working memory (Adelson,1984).

Another study concluded that students are not given enough instruction on how to put the pieces together and that they should be made aware of concepts as goals and plans, and compositional strategies (Spohrer and Soloway ,1986).

Solutions have been proposed by the teaching of conceptual models (Bayman and Mayer ,1988) where this was found to enhance problem-solving

performance. Also it has been shown that students benefit from explicit instruction in complex design skills (Linn and Clancy, 1992).

Further study on these proposed solutions was carried out where students were given a planning strategy for algorithm development. The experiment confirmed the hypothesis that this would increase their strategic/conditional knowledge (McGill and Volet,1997).

All this research indicates that the difference between teaching a student programming and teaching a student to be a good programmer, lies in the parallel teaching of design methods. Learning to program is significantly about learning to solve problems, and will greatly benefit from explicit instruction in problem-solving strategies and techniques.

1.2 SIMPLE APPLICATIONS DEFER DESIGN METHODS.

A difficulty exists in that at the beginner level, only simple exercises can be used to allow learning of algorithm development and the syntactical features of the language. There has been some argument as to whether Visual Basic is an appropriate language for this introductory level as Visual Basic contains a wide number of components that must be understood if students are to form accurate mental models. There has been concern that BASIC may be a better learning language than Visual Basic by removing the complexity imposed by the event-driven environment. Overall, the literature indicates that students gain a positive response from learning with Visual Basic, finding it interesting, positive, and providing a chance for creativity (Bishop-Clark,1998).

However, most design methods have very little relevance to simple applications. Usually software design and modelling is applied to larger application domains and has very little, if any, relevance to the single form simple application taught in these introductory programming courses. As a result very little, if any, design is introduced in these courses. After all, a purpose of modelling is to reduce complexity. When complexity is at a barely perceivable level, then modelling may indeed increase complexity.

1.3 RESISTANCE TO DESIGN

Larry Constantine in his book "The Peopleware Papers" comments: " Structured methods, disciplined development, paper-and-pencil model building, and software metrics are all unjustified impositions on the free artistic expression of our brilliant programming cowboys" (2001, p36). He goes on to add: "cowboy coders are simply those oppositional developers who can't abide being fenced in by standards, constraints, or discipline, who resist all effort at being reigned in by supervision or collaboration with others, who put idiosyncratic originality above usability or reliability" (p 49).

These 'cowboy coders' are often very talented programmers producing very clever and skilled code. However, as Constantine suggests, "..most software is produced by groups of people working together..... very little software is developed by individuals working alone" (p47). It is in this collaborative environment that some form of shared plan or design is required. There is disagreement between proponents of different processes as to the extent of this: ranging from the Plan-driven processes such as the Rational Unified Process through to the more 'agile' processes such as Extreme Programming. But they all do concur on the need for design: it is only the amount that is debated.

Constructivist educational theory would indicate that when introductory programming is taught, the absence of a design methodology will allow this 'cowboy coder' mentality to be developed or reinforced. Students, by achieving success in their programming course, without their perception of having gained any assistance from a design methodology, will learn that design is not necessary.

2. THE COATS-MELLON OPERATIONAL SPECIFICATION, AS A POSSIBLE SOLUTION

The Coats-Mellon Operational Specification (CMOS) method has been in use since 1994 (Coats and Mellon, 1995). It is a simple method for initiating and nurturing the discovery process that deals with interface object events.

This method captures system behavior from a user's viewpoint, producing an operational specification that can be translated into most existing system-development methodologies.

The method was designed to be useful in a 'paper and pencil' environment. The authors state their goal as being "..to produce an inexpensive way to capture important operational information quickly and effectively without all the hoopla surrounding CASE. It is intended to be a back-to-basics approach, independent of CASE tools".

The operational specification is a set of diagrams that specify incoming stimuli via actor events, and a system's response to these stimuli. An actor event is an occurrence initiated by an actor at some point in time. The operational specification consists of diagrams that divide behavior into a set of actor events and system responses to those events. CMOS can act as a 'front-end' to other modeling approaches such as UML, so it is most certainly not a 'throw-away' approach (Coats and Mellon, 2001).

CMOS provides a simple way of graphically depicting the dynamic behaviour of a collection of GUI controls on a form. It provides a quick and straight-forward method of displaying interactions between controls. Thus small programming tasks with complex control interactions, can be simplified by using CMOS.

The opportunity then exists for educators to use a small task that will benefit from prior design. Experiencing this benefit should reinforce the value of design to the students. Thus the major objection to using Visual Basic in an introductory programming class, (the complexity of the event interactions), becomes a learning resource. When Visual Basic is used with a design methodology which helps reduce complexity to a manageable level, the negative aspects of an event-driven environment are removed and design is made more relevant. Student learning may then be enhanced in a number of ways: practical experience in using a design methodology; enhanced appreciation of the value of a design methodology; and improved problem-solving ability (Bayman and Mayer,1988).

3. DOES IT WORK ?

3.1 METHODOLOGY

A pre-test post-test design was used with an Attitudinal Survey on a class of Students at the beginning of the PP114, Programming Principles (Visual Basic) course at Christchurch Polytechnic Institute of Technology. At the start of the course, they were asked to indicate their attitudes to design by means of a number of rating scales. They were then given instruction on CMOS design, and requested to use this design process to help with their assessed programming exercises. CMOS appeared to be easily understood, as indicated by their resulting CMOS diagrams.

Was the design process we used during the course of any help to you?

0	3	5	6	0
None	Very little	Some	Quite a bit	A large amount

Most students (78%) indicated that they were helped by the CMOS design method. Six (43%) indicated that they were helped 'Quite a bit'.

Do you think a different design process may have been more helpful to you?

3	7	4	0	0
No	A little	Some	Quite a lot	A large amount

Most (78%) indicated that to some extent, a different design process may have been more helpful

There were no recorded characteristics that distinguished those who responded 'No' from the others.

Do you value design more now than you did at the beginning of this course?

0	0	5	7	2
A lot less	A little less	About the same	A little more	A lot more

Many students (64%) indicated that they valued design more at the end of the course than they did at the beginning.

At the completion of the course they completed a follow-up rating scale on their attitudes to design.

3.2 DISCUSSION OF RESULTS

There were 14 students in this class. Of these 10 were male and 4 female, with an average age of 27, and all having completed a Program Development course in which they learned simple pseudo-code and data flow diagramming. Five of the students had completed analysis and design courses (UML), and 6 had completed programming courses in other languages.Following are displayed the key questions and the frequency of responses.

The only characteristic of note between those who saw design as more value now ('A little more' plus 'a lot more'), and the others, was that there were no females in the 'about the same' group. Also the average age of the group who gained in value of design was higher (t-test, $p<.005$) than those who did not ('about the same').

4. CONCLUSIONS

This was purely an exploratory study. It showed that the CMOS Design method could be taught as part of an Introductory Visual Basic Programming course, and that CMOS was seen by many of the students, as offering some benefit. It took very little class time to teach the CMOS method, and the students appeared to master it quite quickly.

It is unclear as to whether it helped 'convert' any 'coding cowboys'. This is a difficult concept to measure; the numbers in this study were too small, and the measures used not rigorous enough to make any judgements.

As a means of teaching Event-Driven programming environment, the CMOS design method proved to be very effective. The diagrams clearly depict the possible flows of control, helping to simply represent what can be a complex process for the novice programmer.

In many ways it was only a partial solution to the problem of an applicable software design methodology, as its focus is restricted to modelling the user interactions and resulting events. It does not help conceptualise and model the algorithms required, thus did not provide a 'total' solution. However, the results seen in this exploratory study were encouraging.

REFERENCES

Adelson B. (1984). "When Novices Surpass Experts: The Difficulty of a Task May Increase With Expertise", Journal of Experimental Psychology:Learning, Memory, and Cognition., Vol 10(3)

Bayman, P. & Mayer, R (1988). "Using conceptual models to teach BASIC computer programming", Journal of Educational Psychology 80(3), 291 - 298

Bishop-Clark, C (1998). "Comparing Understanding of Programming Design Concepts Using Visual Basic and Traditional Basic", Journal of Educational Computing research, vol 18(1)

Coats, M & Mellon, T. (1995). "Constructing Operational Specifications", Dr Dobbs Journal, June 1995

Coats, M & Mellon, T. (2001). "Integrating CMOS with UML", Dr Dobbs Journal, June 2001

Constantine L (2001). "The Peopleware Papers" Yourdon Press, New Jersey

Linn , M.C. & Clancy, M. J (1992). "The Case for Case Studies of Programming Problems", Communications of the ACM Vol 35(3)

McGill, T.J. & Volet, S.E. (1997). "A Conceptual Framework for Analysing Students' Knowledge of Programming".,Journal of Research on Computing in Education. Vol 29(3)

Rahman, Rahim, & Seyal (1999). "Requisite Skills of Entry-level programmers: An Empirical Study", Journal of Educational Computing Research 21(3)

Spohrer J. & Soloway E (1986). " Novice Mistakes: Are the Folk Wisdoms Correct ?" Communications of the ACM Vol 29(7)