Full paper

The Teaching of Introductory Programming: Issues of Context

Trevor Nesbit

University of Canterbury

trevor.nesbit@canterbury.ac.nz

Abstract

There are many different contextual aspects to consider when teaching programming to a group of students, who have never studied programming before, or indeed to those who have studied programming before, but not had a particularly positive experience.

This paper explores some of the issues related to the teaching of introductory programming from two different contexts. Firstly issues related to the methodology being followed, and secondly, issues related to the differing contexts of the students.

Issues relating to the methodology are explored from two different standpoints which are firstly the choice of following a procedural or object oriented approach, and secondly whether the analysis and design concepts should be taught before coding or whether the teaching of coding should be integrated with the analysis and design concepts.

Issues relating to the differing contexts of the students enrolled include:

- The subject areas the students are specialising in (networking, multimedia, software engineering, business analysis etc)
- The mathematical ability and background of the students
- The interest of the students including gaming, use of mobile devices

Consideration is also given to the use of micro-worlds to reduce complexity.

The paper concludes with a summary of contexts that should be considered when planning introductory programming courses, and as such would be of some use in planning this aspect of a proposed collaborative computing degree.

Keywords: Teaching, Introductory Programming

1. Introduction

Much has been written about approaches to use in teaching programming to novice programmers. Writers such as Bruce (2004), Kelleher and Pausch (2005), Lahtinen, Ala-Mutka and Jarvien. (2005) and Powers, Ecott and Hirsfield (2004) have highlighted the issues including the use of micro-worlds, approaches to reducing complexity, the extent to which object oriented concepts should be introduced in a first course and the overall purpose of learning programming.

The purpose of this paper is to identity issues that need exploring when planning an introductory programming course to provide a basis for discussion, further study and to provide input into the proposed collaborative degree being developed by the National Advisory Committee on Computing Qualifications (NACCQ).

2. Methodology

A brief literature review was conducted with a view to exploring some of the issues that have been identified by other researchers. Based on this review, a set of questions

This quality assured paper appeared at the *22nd Annual Conference of the National Advisory Committee on Computing Qualifications* (NACCQ 2009), Napier, New Zealand. Samuel Mann and Michael Verhaart (Eds).

were emailed to members of the NACCQ Software Development Special Interest Group, and to a small number of academic staff in other tertiary education institutions. Twelve (12) people responded to the questions in the email, with two of the respondents being from the university sector in New Zealand, and the remaining ten being from institutes of technology and polytechnics (ITPs) in New Zealand.

Due to the small sample size and the manner in which it was selected, it is not possible to use the results to make any generalisations about the views of all academics involved in teaching introductory programming. The results will however give an indication of the breadth of views and may provide a basis for a further study where generalisations may be able to be made. The questions are shown in Table 1.

The responses were analysed and summarised with a view to identifying the range of opinions and views being expressed (as opposed to seeking to make generalisations about the issues being explored, mainly because of the low sample size).

3. Literature Review

A discussion regarding the teaching of introductory programming that took place on the ACM SIGCSE mailing list was analysed and summarised in Bruce (2004). The main points to come from this analysis were related to the choice of language and the extent to which object oriented concepts were covered. Also highlighted was the extent of the issue of teaching introductory programming and the high failure and withdrawal rates that have been experienced by a number of institutions world-wide, and the need for further work to be done in the area as there appear to be no easy answers.

Kelleher and Pausch (2005) identified a number of barriers for novice programmers and explore a number of ways in which these barriers could be lowered. As part of this study they present their taxonomy of languages and environments to make novice programming easier. The high end of this taxonomy separates into two groups with one being “systems that attempt to teach programming for its own sake” and the second being “those that attempt to support the use of programming in pursuit of another goal”. Within this first group exist tools that have been developed for the express purpose of supporting the teaching of programming, for example Alice and Karel amongst many others, and in the second group exist tools that have been created with the purpose of empowering other activities, and within this can exist the creation of

1	Do you think that the methodology for the first exposure to programming should be (a) object oriented (b) procedural (c) it doesn't matter provided it is done well so that the other can be understood later?
2	How do you feel about integrating analysis and design with coding in an introductory course? For example (1) teaching UML in isolation from coding or integrating them together (2) teaching structure logic in isolation from coding or integrating them together?
3	Do you think that we should allow for providers (ITPs/Polytechnics etc) to make their own choice between (a) object oriented and (b) procedural when we consider nationally developed IT qualifications?
4	How much difference would it make to you if you knew that all of the students in an introductory programming class where (a) specialising in multimedia or (b) specialising in networking or (c) wanting to become business analysts? How would this difference manifest itself?
5	Does the mathematical background of the students matter? Do differing mathematical backgrounds require different approaches?
6	What thoughts do you have about teaching intro programming by developing for mobile devices or in gaming environments like Lua for World of Warcraft?
7	What thoughts do you have about developing in micro-worlds (Karel, Alice etc)?

Table 1 – Questions Emailed to Respondents

gaming environments such as the LUA scripting language for Word of War Craft.

In exploring the difficulties experienced by novice programmers Lahtinen, Ala-Mutka and Jarvien (2005) identified that the biggest problem of novice programmers does not seem to be the understanding of basic concepts but rather learning to apply them. In their conclusion they stated that "Programming is not difficult only because of the abstract concepts. Students have also problems in different issues related to program construction. It is important for the learning that the students do programming by themselves. With carefully designed materials and approaches teachers can guide students' knowledge and skill construction.

Powers, Ecott and Hirsfield (2004) analysed and discussed a number of the issues surrounding the use of micro-worlds for the teaching of introductory programming and highlight the important issue of being able to transition to real programming languages such as Java and C++. As part of this piece of work they identified improvement that could be made to micro-worlds to make this transition easier.

4. Results

The following is a summary of the responses that were made to the questions by the twelve respondents. Given the low number of participants, this analysis is not intended to make generalisations about the views held, but more to express the range of views that exist so that they could be used as a basis for a further study.

4.1. Issues Related to Methodology

The responses to the question as to whether the methodology for the first exposure to programming should be (a) object oriented (b) procedural (c) it doesn't matter provided it is done well so that the other can be understood later are shown in Table 2.

While the results in Table 2 appear to show a strong preference for procedural amongst the respondents, 5 of the 8 respondents made comments to support their view,

Methodology	Responses	Percentage
Procedural	8	66.7%
Object Oriented	1	8.3%
Doesn't Really Matter	3	25.0%
Total	12	100.0%

Table 2 – Responses to Question Regarding the Methodology

with these comments being shown in Table 3 and along with the procedural part of the response from one of those who indicated that it doesn't really matter. These comments are suggesting that the reasons for choosing procedural over object oriented are most about the reduction of complexity for new programmers as opposed to a longer term preference of object oriented. There is also support for the concept of some object oriented concepts being introduced reasonably early.

The respondent who expressed a preference for object oriented included the comment "Look at objects of some existing classes - UI classes are great for this. Give a label text etc" which is to an extent connected to the ideas of reducing complexity expressed in some of the comments made by those expressing a preference for procedural. The respondent who indicated that it doesn't really matter made the comment in relation to object oriented that "OO modelling is of the real world and people relate to it better. Procedural programming parts of OO models can be incorporated and explained as objects needing to 'do' certain things and hence we need the procedures". This comment is suggesting that it is important that students can relate to methodology, which is consistent to an extent with the idea of reducing complexity.

4.2. Integrating Analysis and Design with Coding

The breakdown of responses related to the question of whether the teaching of analysis and design should be integrated with coding is shown in Table 4, and indicates a preference to keep them together amongst the respondents.

Comments From Those Preferring Procedural
Since OO modelling also involves doing Procedural programming, perhaps do Procedural first, so when teaching OO there's a lot already understood and taken in
Procedural – can be in an OO environment if they are being taught to write methods – which is procedures.
Procedural first but introducing objects fairly early (use of existing classes), user-written classes only at end
Procedural worked best for me for beginners, as it removes a whole layer of complexity associated with dynamic memory allocation, e.g. object vs. classes. For beginners I use VB.Net which has been "tamed" by Microsoft so that you don't have to use hard-core OO techniques (e.g. FrmMain.Show() works, where FrmMain is a class, shock-horror :-)
Procedural. One step at a time. Let them learn to walk first. Enough concepts being introduced already
Procedural: students need a grounding in the easy stuff first, variables, all the decision structures: loops ifs case. Then they can start passing parameters and calling methods and then OO becomes easier.

Table 3 – Comments from Respondents Preferring Procedural

	Responses	Percentage
Integrate Together	9	75.0%
Keep Separate	3	25.0%
Total	12	100.0%

Table 4 – Responses to Question Regarding the Integration of Analysis and Design

Of particular interest was that many of the comments of those preferring to integrate them together indicated a preference to not go overboard with the amount of analysis and design initially, due to the complexity of methodologies such as the Unified Modelling Language (UML). One respondent quoted that their preference was to "Integrate, but after this year I think teach coding, then analysis and design", with this idea not having been presented as one of the possible responses to the question.

One of the respondents who indicated that they would keep analysis and design separate responded with the following quote: "I don't do formal UML in intro programming. Do lots of this at a higher level tho[ugh]. What structure etc I do I integrate with the programming as necessary"

The comments made in response to this question tended to show that the decision about which approach to take was more related to what would help the students learn the best as opposed to some methodological stand point.

4.3. First Language Methodology

The responses to the question whether institutions should have the choice of which methodology they use in nationally developed qualifications, noting that two of the respondents were from the university sector and that this question was not included in the email sent to them.

	Responses	Percentage
Yes	7	70.0%
No	2	20.0%
National Directive Needed	1	10.0%
Total	10	100.0%

Table 5 – Should the First Language Methodology Should an Institution by Institution Decision

Two the seven respondents who indicated that the institutions should be able to choose which their own approach also commented that this was due to the difficulty in reaching a consensus on this aspect.

4.4. Students Specialising in Different Subject Areas

The responses to the question regarding the differences the students' area of specialisation makes are summarised in Table 6.

The two respondents indicating that it would make a significant difference were both referring to contextual issues in terms of the nature of the approach taken, the examples used and the choice of programming language.

Responses	Number	Percentage
Significant difference	2	16.7%
OK to provide relevant examples, but the basics must be covered	3	25.0%
Not particularly relevant as basics are the same	7	58.3%
Total	12	100.0%

Table 6 – What Difference Does The Students' Specialisation Make?

All of the other respondents, whether indicating that the background had no relevance, or that it might have some impact on the examples used, indicated the importance of the basic concepts and structures being covered well due to their relevance across all aspects of programming.

Those who indicated the relevance of the examples also pointed towards the increase in student interest if the context is more tailored, with one respondent asking the quite valid question as to whether students really know what they want to specialise in until they have had a go at the different subject areas. This last respondent also mentioned that the concept of teaching multimedia scripting to art and design students had some appeal.

4.5. Mathematical Background of the Students

The responses to the question regarding whether the mathematical background of the student really matters on whole, point to the importance of logical thinking and the basic mathematical aptitude as being the background required for first year programming. Amongst the respondents there was some suggestion that it depends on the type of programming that the students end up getting involved with. As with the students specialising in different subject areas, it can have an impact on the examples used with this being mentioned in the three of the twelve responses with one going on to quote that "Mathematical background only matters if the tutor is a maths nerd and treats programming as giant calculator stuff".

4.6. Use of Mobile Devices and/or Gaming Environments

There was a quite varied range of responses to the question about the use of mobile devices and/or gaming environments from some quite positive responses to some quite negative responses, particularly when it comes to the teaching of introductory programming. The complete range of responses is shown in Table 7. In a similar vein to the responses regarding students specialising in different subject areas and the mathematical background, a theme to emerge from the responses is the importance of students grasping the basic concepts well, with some of the more positive responses having that as an overall rider or condition for the adoption of this type of approach.

There is however some degree of support amongst the respondents for making programming more "sexy" or "exciting" as a way of attracting those who might not typically be interested in programming to give it a try. This needs to be balanced with the comments of other respondents regarding the need to not "dumb-down" what programming is all about and the potential for some environments to be demeaning for some students.

4.7. Use of Micro-worlds

When it came to the use of Micro-worlds such as Alice and Karel, 7 of the 12 respondents indicated that they would not use them, 4 of the 12 respondents responded positively, with the 12th respondent being unaware of their use.

In the negative comments that were made, there was some evidence of not wanting to over simplify programming, as well as small amount of use perhaps being OK. One respondent mentioned how the improved Integrated Development Environments (IDEs) now available reduced the need for micro-worlds, with other respondents highlighting the potential difficulty of transitioning to other "real" programming languages.

Those who responded positively also mentioned the need for ensuring that the basics were covered (a similar trend

to responses to earlier questions) as well as not using micro-worlds too much.

5. Analysis and Discussion

A number of themes emerge from the analysis of the respondents' comments and views. These include the importance of students being in an environment where they can grasp the concepts of programming before moving on to areas of greater complexity, and that this needs to be balanced with not providing students with a distorted view of the levels of complexity involved.

This applies to the use of micro-worlds and to the use of mobile devices and gaming environments where there is also the need to balance between attracting people into programming who might not be attracted otherwise.

A further theme to emerge was the degree to which the responses were focussed on the learning needs of the students, and the openness of many of the respondents to consider new options with a view to enhancing the learning of their students.

Given the range of views expressed in response to some of the questions, particularly in choice of methodology and integration of the teaching of design and coding, the development of introductory programming courses for the collaborative degree proposed by NACCQ would require either significant discussion to reach a consensus or sufficient provider flexibility to allow different approaches, some of which could be influenced by the different economies of scale contexts of smaller and larger institutions.

6. Conclusions

There is a need for a number of different contexts to be held in balance when planning introductory programming courses, with these being:

- The use of micro-worlds and other environments to reduce complexity, but not hide it so much to present a distorted view of the complexity that exists.
- The choice of methodology between object oriented and procedural programming when object oriented is the dominant paradigm in industry, but in the view of some of the respondents, starting with procedural will enable students to learn faster.
- The issue of the integration of the teaching of design and coding needs to be considered at the same time, particularly with many of the respondents being keen to reduce the initial complexity of UML.

The varied responses relating to the above issues suggest there may not be one single right way of teaching introductory programming for all institutions, all students and all teaching staff, and that as a consequence of this, and collaboratively developed introductory programming course may need to allow for individual institutions to be able to choose which approach to take in the best interests of their academic staff being able to provide the best first programming experience for their students.

References

Bruce, K.B. (2004). Controversy on How to Teach CS 1: A Discussion on the SIGCSE-members Mailing List. *ACM SIGCSE Bulletin, Vol 36(4)*, 29-35

Kelleher, C. & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys, 37(2)*, 83-137.

Lahtinen, E., Ala-Mutka, K. & Jarvien, H. (2005). A Study of the Difficulties of Novice Programmers. *ACM SIGCSE Bulletin, 37(3),* 14-18

Powers, K., Ecott, S. & Hirsfield, L. (2004). Through the Looking Glass: Teaching CS0 with Alice. *ACM SIGCSE Bulletin, 39(1)*, 213-217

Responses Regarding The Use of Mobile Devices and/or Gaming Environments
Makes it more attractive and levers more on existing concepts
If it can still teach the basics while adding to their interest then that is good.
Introductory programmers shouldn't be able to program properly at the end. Prefer to teach in something that looks relevant to their course. WOW is demeaning.
Not all students have the necessary type of devices required. maybe a course on a higher or separate level – not intro programming
Not for first year programming. At lease not until the fundamentals have been covered first.
I have mixed feelings about that approach. although it is fun for the students to do, you need to prepare an environment in which they can reach that goal. that environment consists helper files and prefab methods/procedures. then it becomes less clear to the students what is 'pure' in the programming language, and what has been prepared by the lecturer (or taken from the internet)
My impression is that we've been going steadily backwards in our teaching of programming for the last 30 years, trying to make it "easier" by providing special environments for teaching that move the students further and further from the basic machine they're supposed to be programming. I think these specialised environments provide additional distraction (extra competing cognitive demands, if you want to make it sound fancy). Of course, it's still fundamental that examples have to be motivating and gaming environments can help in that respect, but not if the examples are built on more and more layers of complexity that the students don't understand.
I think that would be really sexy and I'm all for it, but isn't that too difficult at intro-level? As for mobiles there are gnarly practical problems for tutors, like do you program for an emulator only or do you provide a class set of mobiles for students to use? Are they supposed to use their own mobiles? What about all the different environments and standards with new mobiles coming out every month? With the time constraints and ever increasing class sizes I have put this idea into the too-hard basket. I have no concept of WoW, except that it is a major distraction in the lives of some of our students. Programming for it could either move students from a game consumer habit to creative programming activities, but likewise it could be seen as promoting gaming to a bigger role than we would normally give it. Also, some students may never have seen WoW, so they would need familiarisation time with it, and possibly be at a distinct disadvantage compared to real gamers, an aspect I personally don't like conveying.
No experience. Sounds like fun so long as the basics still get through
This depends on the learner, some prefer an environment they are already familiar with and can relate to that, others might prefer a strict discipline of "non-gaming"environments. I have personally experienced this.
I like to use a stripped back language to begin with, get the basics sorted with no forms or objects. Start by running from the console then more on to a themed approach.

Table 7 – Complete Range of Responses Regarding Use of Mobile Devices and Gaming Environments