Introducing B.Com Information Systems Students to Basic Programming: Overcoming Some of the Difficulties

Lyrice Cohen

The objective of equipping B.Com Information Systems students with basic programming skills is not without its hurdles. Difficulties include large class sizes, varying levels of prior exposure and low motivation for technical learning. The problem is compounded by the standard (and not minor) challenges of teaching programming such as the challenges of teaching cognitive thinking and the need for individual interaction with the computer. Lecturers who are both technically adept at the current programming tools and genuinely concerned with enabling students to "see the light" are not so easy to come by.

This paper discusses and demonstrates an approach to dealing with some of these problems. This approach has been chosen because of its applicability to our specific context, where we have limited laboratory resources and are dealing with students from many diverse educational and technological backgrounds. Elements of the approach would, however, be relevant in all contexts.

The approach includes a programming concepts course that is run during lecture time, complemented by a series of detailed on-line tutorial sessions. A Visual Programming Tool is used for these tutorial sessions, exposing students to ideas of event-driven programming and the mechanics of using the tool. Through the visual environment, students also have practice with the programming logic they have learned in the programming concepts course. The tutorials are consistent with the methods used by John Barrow et al [1]. Techniques are employed to encourage and motivate the students as they work through the tutorials, giving them a sense of empowerment with the technology. The tutorials have been prepared in detail, the longer term goal being for them to be re-used in future years, regardless of which lecturer is actually giving the course. A survey of student reaction to these tutorials indicates a positive response.

This paper consists of four sections. These discuss the scope of the course and the choice of the tool to be used, low student motivation for learning basic programming, student evaluation and the future.

Scope of the Course and the Choice of Tool to Be Used

It is important to point out at the outset that this approach has been developed specifically for First Year B.Com Information Systems students. The emphasis of the Information Systems course lies not on the understanding of intricate advanced programming concepts, but rather on the deployment of Information Technology in a way that will further the objectives of the business. The Information Systems student, therefore, need only attain an understanding of the fundamental principles of programming and a working knowledge of a visual tool that is sufficient for him / her to develop the Second Year project, which is a simple Information Systems application. This understanding and working knowledge should provide the student with a level of confidence about programming that will stand him / her in good stead in whatever commercial environment he /she chooses to go into. It is not considered to be necessary for the First Year Information Systems student to acquire in-depth advanced programming skills.

Bearing this in mind, the choice of the particular Visual Tool to be used for the tutorials was not considered to be of prime importance. With a view to using a product that is extensively used in the industry and therefore constitutes a marketable skill, a Java option was considered and then discarded due to problems experienced[1]. Visual Basic was then turned to as another marketable option.

The final decision was to use Visual Basic for Applications in the Microsoft Access environment (see, for example [2]). This choice proved to be advantageous for a number of reasons. Firstly, the programming tutorials are done just as the students have finished learning Microsoft Access. They are now familiar with the Microsoft Access environment and it is a natural progression for them to learn Visual Basic in that environment. Secondly, Visual Basic for Applications is a natural precursor to the Visual Basic.Net course they will be doing in the following year of study. Thirdly, Microsoft Access is a stable product that has been installed in our computer laboratories for a number of years. It does not have a high memory requirement and gives no installation problems. This makes laboratory sessions using limited facilities for large classes much easier to manage. A further advantage in this regard is the fact that most students who have computers at home also have Microsoft Access installed on those computers. They can therefore work through the tutorials at home – freeing up the computer laboratories for those who do not have computers at home, a common problem [3].

Once the tool was decided upon attention could be turned to the mode of presentation of the actual tutorials. After careful consideration it was realized that the mode of presentation, if carefully constructed, could make a major contribution to addressing one of the difficulties discussed earlier, i.e. low motivation of B.Com students for technical learning.

[1] The intention was to use a Java tool called "Sun-One" that was downloaded from the Internet. This was, however, found to be an advanced IDE rather than a fully developed Visual Development environment. In addition serious problems with product support were encountered, causing it to not be a viable option.

Low Student Motivation for Learning Basic Programming

As mentioned previously, one of the difficulties that have been encountered in past years of teaching programming to B.Com students is the low motivation levels of these students when it comes to learning how to program. The fact that programming is not considered to be of prime importance in the Information Systems curriculum means that insufficient weight is given to it to make the acquisition of the programming skill an essential part of attaining the credit. This enables many students to "prioritise it out of the picture" – to pay minimal attention to it and get through their degrees without passing the programming part of the course. For this reason it was considered important to establish the possible causes for these low motivation levels and to work on ways of combating these.

Four main areas were identified here: manipulating student priorities, fear / phobia / resistance to programming, maintaining student interest and dealing with complexity.

Manipulating Student Priorities

Perhaps the most obvious cause of students paying insufficient attention to the programming course is the fact that it requires a lot of time and effort to learn how to program. It is not a skill that can be attained quickly and easily or by doing some light reading. Students have many conflicting priorities to achieve in a limited time period, and they will tend to select those things for which their "investment" in time and effort will have the greatest perceived "return". The major currency of this "return" in our institution is marks, and we need to use this as a way of manipulating student priorities, i.e. of getting them to do the programming tutorials.

Last year, a total of 6 of these tutorial sessions were run. At the end of each tutorial session there was an assignment to be handed in. Only 2 of these assignments were marked, but the students did not know which ones they would be, so they had to do them all. In addition, students lost 1/6 of the achieved mark for each assignment that they neglected to hand in. This technique has been used previously for several years for another course, and has shown positive results.

Fear / Phobia / Resistance to Programming

A phenomenon that appears to be prevalent amongst B.Com students that also contributes to low levels of motivation for technical learning is an almost tangible fear / phobia / resistance to programming. Evidence for this can be heard in some common comments coming from Third Year Information Systems students, like "Programming isn't my thing – I cope OK with the other stuff." or "I'm not good at the programming part – I leave that up to the other members of the project team." Last year, introducing this practical programming course to the first year students and explaining that it would involve a large amount of individual interaction with the computer, I asked the class how they felt about embarking on this experience. One of the ladies in the front row replied, wide-eyed, with one word – "terrified!". It is my opinion that her reply echoed the sentiments of many of the other students in the lecture hall at that time.

It would appear that the source of this fear / phobia / resistance to programming is the fear of the individual that they will not be able to sufficiently master the programming skill. This perceived inability leads them to have a very low self-image / self-esteem as a programmer. As always with low self-esteem – we don't want to do what we are not good at – the result is a low motivation for learning how to program. Expecting the students to do programming exercises that are out of their reach or that they have not been adequately prepared for serves to further exacerbate the problem.

Positive self-esteem / self-image as a programmer can only be encouraged with positive achievements, however small. A meaningful objective, therefore, would be to give the students ample opportunities to perform small, achievable tasks with visible results. These tasks would, of course, have to have at least some substance – there would have to be something, however small, in each one that would attempt to "stretch" the student, and make them work out something they had not previously known or apply something they have just learned to a new situation. It is contended here that each of these tasks would provide for a small triumph that would give the student a sense of achievement, perhaps even of excitement, and that this in turn would lead to an improved self-image / self-esteem as a programmer, which would in turn motivate and encourage the student in their learning effort.

Maintaining Student Interest

Many a student is put off from the idea of programming by the perception that it is a boring activity. The common stereotype of a studious "nerd" with glasses who sits in front of a computer all day prevails. We need to break this stereotype and show that programming can be fun,. This can be done in part by careful choice of exercises – games, for example – and by introducing humour. The tone of the text of the tutorials should be informal, friendly and encouraging and it is preferable for the text to be presented in an informal font that is pleasant for the students to work with. Starting each tutorial with a short synopsis that lists the tutorial's objectives also helps to maintain interest.

Care should be taken not to lose the students' attention during the course of the tutorials by indulging in too much detail. It is not necessary, as we uncover each successive cognitive or programming language concept, to be exhaustive and go through every option available. Students do not need to be immediately aware of every option available, they need to be able to understand and apply the concept. An exhaustive discussion of every variable type, for example, can be quite boring and quite meaningless too. It is far more meaningful to discuss a different variable type in each different tutorial and experiment with its application and explore other related aspects of programming before progressing onto the next variable type.

Dealing with Complexity

Programming is a complex activity, and teaching programming involves developing within the student the capability to master this complexity. We are effectively teaching cognitive thinking and we need to look at ways of doing this most effectively.

The human mind is capable of learning only one new thing at a time. It is essential, therefore, that we break up the complexity into manageable chunks and approach the material in a step by step manner. Each "chunk" or step must be fully understood before moving onto the next step, and the only way to do this is to expect the student to apply what they have just learned and to require them to be "stretched" a little. Too many texts that portend to teach computer-based tools fall into the trap of merely instructing the students, step by step, mouse-click by mouse-click , how to achieve the task, never discussing the concepts involved and never expecting an independent application of what has been learned, and leaving the students not too much the wiser than when they started out. It is absolutely vital, in whatever way possible, to encourage the students to think, internalize what they have learned and to be curious about the workings of the program and the programming environment.

Concepts or "steps" previously learned also need to be reinforced with repetition. Using previously learned programming constructs in later exercises is a natural way of doing this.

At all times we should be attempting to move from what is "known" to the student to what is "unknown" to them, so that they start off with a secure basis for understanding. One technique of doing this is to first demonstrate the effect of a program by executing it, and then afterwards to examine the code and explain how it works. Then, of course, the student must be asked to apply what he has just learned. Barrow et al [1] use this way of teaching extensively, referring to it as the "experiential" model.

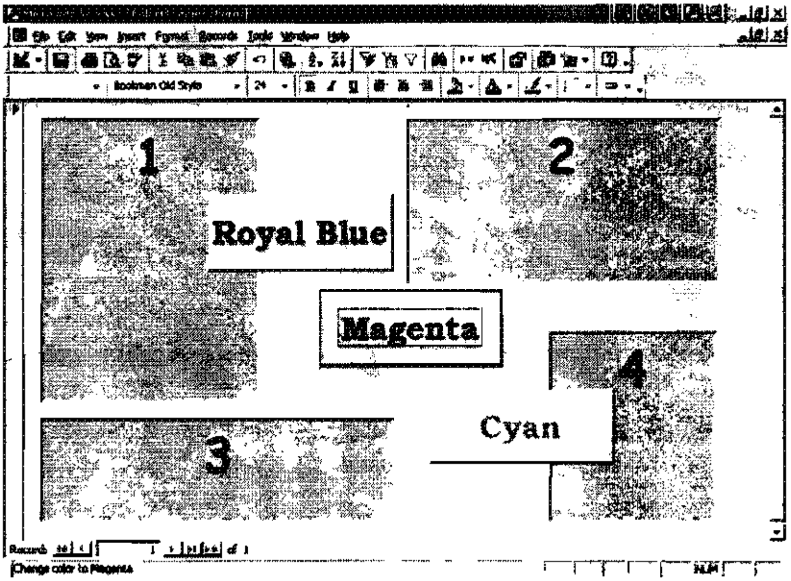
Unnecessary complexity should be removed wherever possible, so that students are able to focus on one particular concept at a time. In a practical programming environment it is often necessary to complete a whole lot of prerequisites before getting a programming construct, say an "If" statement, to work. These prerequisites require the time and attention of the student and detract from the main issue, i.e. understanding the programming construct.

An attempt has been made in these tutorials to find ways of doing these prerequisites for the student wherever possible, so that they can concentrate on understanding the next important concept, say, the "If" statement. For example, the students are provided with ready-formatted forms to work from. They are never asked to develop their own forms. Formatting forms is a time-consuming activity and not a particularly difficult one, and one that they should already have mastered in their Microsoft Access course. Precious computer laboratory time is better spent on the programming itself. Certain subprograms or small pieces of code that enable them to achieve things that they are not yet able to do on their own have also been provided to them. Several constants have also been predefined for them to make the programming easier to understand. For example, constants have been used to associate the names of colours with the lengthy numbers that Microsoft Access uses to represent those colours.

To further remove distraction and allow them to focus on absorbing the concepts students are reassured that they are not expected to understand everything they see on the screen initially and the things they can safely ignore for now are continually pointed out to them. Particular care has been taken to avoid difficult vocabulary and the tutorials are printed in a large font with wide spacing, all of which are believed to facilitate understanding.

Perhaps the most useful part of this approach in terms of conquering complexity is the fact that a Visual Tool is being used as the medium of teaching. Working in the Visual environment does increase the complexity and the amount that has to be learned, as now the student has to master both the programming concepts and the Visual tool. The visual aspect and the immediacy of the environment, however, can be used to enhance understanding.

As an example, let us consider the first of the tutorials. The students are asked to display the following form:



As you can see, this is a simple form with four text boxes (numbered 1 to 4) and 3 command buttons. The students are asked to click in different places all over the form and observe what happens. They soon discover that nothing happens when they click anywhere other than on the command buttons, and that when they click on a particular command button the colour of all the text boxes changes to that button's colour. It is pointed out to them that this happens not before they click, not after they click, but as they click. That click is the *event* that made the colour change happen. Actually somewhere there is a little program called an *event handler* that made the colour change.

It can be seen here that the students have just used the visual environment to demonstrate very quickly to themselves the concept of event driven programming. This kind of live "self-demonstration" can be used in many ways to enhance student understanding.

The extent of the visual impact of what is happening on the form is also thought to contribute to the understanding process. This is why the form displayed was designed to take up the whole screen, and the text boxes that change colour are so large.

Student Evaluation

An evaluation questionnaire was handed out to students to determine their reaction to the course. The response was quite positive, with 80% of students indicating that they now feel more enthusiastic about programming, 72% are feeling less afraid of programming, 65% feel they learned a lot and 60% said they actually enjoyed the course. Surprisingly, though they are feeling more enthusiastic and less afraid, only 42% of them are feeling more confident about learning how to program. This could possibly be due to the fact that this was the first time the course was run, and since there were only six sessions, insufficient ground was covered in order for them to really gain confidence.

The Future

Though the approach adopted appears to have been successful, at least to some degree, it is still considered to be a work in progress. It will be continued with during this year and every effort will be made to improve upon it as much as possible, taking student evaluations and lessons learned from experience into account. This year, the course will be starting earlier, and tutorial content will be adjusted to be more appropriate for the time available. An additional 3 computer lab sessions have also been made available, bringing the total to 9. Tutorial content will also be manipulated and worked upon to even better implement the principles and ideas discussed in this paper.

In closing, this approach appears to have been a step in a positive direction towards resolving many of the difficulties that we have experienced in teaching programming to B.Com Information Systems students. With further effort and development, we will hopefully finally realize our objective of enabling First Year Information Systems students to be motivated to learn programming and empowered with a level of confidence about programming in a visual environment.

References

Barrow, J, H Gelderblom & L Miller 2004. *Introducing Delphi: Theory Through Practice.* Oxford: Oxford University Press.

Zak, D 2001. *Microsoft Visual Basic for Applications*. Thomson Learning.

Winchiers, H 1999. Reflections on Teaching Computer Science to Previously Disadvantaged Students. *SACLA '99*

Author's Contact Details

Lyrice Cohen (lyricec@isys.wits.ac.za)
School of Economic and Business Sciences

University of the Witwatersrand, Johannesburg, South Africa