

# Development of an introductory course in Java programming for a ‘supported distance learning’ environment

Kevin Boone  
Department of Computing Science  
Middlesex University, London  
k.boone@mdx.ac.uk

November 1999

## Abstract

This article describes the development of a course in Java programming, and its delivery in a ‘supported’ distance learning environment. By ‘supported’ is meant that, although students on the course may be geographically distant from the centre of delivery, they did have some personal contact with tutors and other students. However, the bulk of the didactic material was delivered using a combination of CD-ROM and Internet methods. To evaluate the appropriateness and success of this method of teaching, a group of ‘local’ students studied the new course, along with the distance-learning students. Comparison of the local students’ comments and success rates with those of students studying the same course the ‘traditional’ way allows some tentative suggestions to be made about the effectiveness of this style of teaching; suggestions will be offered to improve the delivery of this kind of course.

## 1 Introduction

### 1.1 Background

Computer programming is the production of computer software that will be useful to some group of computer users. Most programming is a commercial activity. Programming is carried out in a ‘programming language’: a set of symbols that instruct the computer how to proceed. There are many programming languages; most have at least some recognizable English words as part of their symbols sets.

‘Java’ is a computer programming language which has existed in its present form for only a few years, although it has similarities with older programming languages (notably C++). Despite its relatively short pedigree, it has become widely used, especially in the commercial and banking sectors of computing. Reasons why this might be the case are interesting, but beyond the scope of this article.

Java is one the increasing number of programming languages that are founded on ‘object-oriented’ concepts. This means that the fundamental unit of organization of the program is an entity that contains both instructions and information. This is in sharp contrast with earlier languages that focussed on organizing programs in terms of task decomposition or data structures. Object-orientation causes some problems for *experienced* programmers who are used to earlier methods of working but ought not, in principle, to present a problem to students who have no experience of any kind of programming.

The great demand for skilled (or even semi-skilled) Java programmers makes learning this language very attractive to students. However, despite many claims to the contrary [e.g., Deitel and Dietel, 1996], Java is not a particularly ‘teachable’ language. Unlike languages like Pascal, Java was not designed with teaching in mind. Its main design goals were flexibility, and ease of learning by experienced programmers, not novices.

The MSc programme ‘Business Information Technology’ at Middlesex University is a ‘conversion’ programme designed for graduates of subjects other than computing science. Almost all students that are studying this programme are doing so to effect a career change into the computing industry (only two of this year’s intake have expressed different reasons for studying the programme). These students have a strong motivation to learn skills that will increase their employability. There is no general agreement in the University whether these students should be taught computer programming at all; but students that express a preference all want to learn Java programming, as opposed to a language that will be easier to learn.

## **1.2 Problems with teaching programming**

Students studying this course are unlikely to have any programming experience, and don’t have any idea what programming is like. Many students with whom I have discussed the matter said that they were very surprised at how difficult programming turned out to be.

The fact is that programming *is* difficult for most people. Worse, it takes a long time to become good enough at it that it is actually rewarding. Clearly in a ten week course we cannot set our expectations too high. However, students have to learn enough to be able to apply for, and stand a reasonable chance of getting, positions as junior computer programmers if they so wish. This sets a limit on the minimum learning outcomes.

My prior experience has shown that, in any group of programming students with no prior experience, the group tends to polarize rapidly into ‘can-do’ and ‘can’t-do’ students. ‘Can-do’ students seem to find a natural accommodation with the subject, usually after two-three weeks, and then make rapid progress. ‘Can’t-do’ students never seem to make the conceptual leap necessary to start to make progress. Although they may do reasonably well in the end, these students struggle until the end of the course and are relieved when it is over. Of course, some students do not fit neatly into either group, but most do. I have never found a reliable

indicator of which people are likely to end up in which group. While it might be thought that mathematicians and engineers are more likely to take naturally to programming, my experience has not confirmed this. There is no obvious racial or gender influence that I have been able to determine, at least to a statistically significant level.

The implication is that one is always teaching a group of students in which there are widely different levels of knowledge and ability *even when the students are selected to start out on an equal footing*. A good course must make allowance for this.

The difficulties faced by students on their first exposure to programming are worsened by the fact that modern industrial practice is to hide from the users of software the internal operation of the computer. This is demonstrated in the way in which the computer user has been presented with an increasingly non-technical working metaphor over the last twenty years or so. Consider, for example, the way in which terminology has changed for the process of ‘using a piece of computer software to manipulate data’:

---

---

1980	Programs manipulate data which is stored on a disk
1985	Programs manipulate files which are stored in directories
1990	Programs manipulate documents which are stored in folders
1995	Users open documents that reside in folders

---

---

In modern computing, the role of the computer software is almost invisible to the end user. The user sees ‘documents’ and ‘folders’ and knows certain gestures that will carry out actions on these documents. This means that many of our students are prodigiously efficient in the use of the computer for all manner of tasks, but have no concept of how it works. This is fine until it comes to programming. Programming does not work at the level of documents, tasks and gestures; it works at the level of data and instructions.

Learning to program is therefore different in kind, rather than in degree, from learning most of the other subjects that these students will study. The difference is the same as that between knowing the physics of the internal combustion engine and being able to drive a car. No body of knowledge, however impressive, will enable one to write effective computer programs; theoretical knowledge is a necessary, but not sufficient, requirement.

### **1.3 Course learning outcomes**

Considering the points raised above, the overall learning outcomes are expressed to students as follows (taken from the course handbook):

“After successfully completing this course you will be able to write computer programs in Java, making use of on-line and printed reference sources. You will be able to create and deploy both applets and

independent applications. The scale of the programming tasks you will be able to tackle is approximately compatible with those carried out by entry-grade programmers in the computing industry. You will be able to use the theoretical concepts of object orientation to manage, structure and document your work.”

## **1.4 Constraints**

### **1.4.1 Student body**

The course of which this article is the subject is taught to students following a ‘conversion’ MSc programme in information technology. As yet we have little experience of the backgrounds and capabilities of the students who are studying the programme in distance learning mode, but we know a great deal about the students who have studied it locally. My own records of, and experience with, local students suggests the following.

- They tend to be older than most University students. 72% of students are over 30 and 10% over 40 years of age.
- Consequently nearly all students are in full-time employment when they begin the programme, and many remain in employment throughout.
- Only 10% of students have first degrees in computing or related subjects. This is as we expect, as this is a conversion programme, and we don’t normally accept students with a technical background. About the same proportion of students have some work experience in the computing industry.
- Nearly all students own their own personal computer; in fact this is almost a formal requirement for studying this course
- Nearly all students are following the MSc programme with a clear aim in mind: they want to obtain good first jobs in the computing industry. They make no secret of this and constantly ask about employment opportunities and the relationship between their studies and the requirements of the ‘real world’.
- The students are mostly highly motivated, and are prepared to work for long periods of time, but...
- Students, not surprisingly, spend most time on work that carries some form of summative assessment.
- Most students have no programming experience at all. They usually don’t have the first idea what programming is about, and are disagreeably surprised when they find out.

- Of all the courses that constitute the MSc programme, students report that they find the programming component the most difficult and least personally rewarding. However, last year over 80% rated the programming component's 'usefulness' as '4' or '5' on a scale of 0 to 5. This suggests that the students understand the usefulness of the subject, but don't find it very agreeable. I can understand this, as I feel exactly the same about mathematics and plumbing.
- A surprising number of students are prepared to indulge in academic dishonesty of one sort or another. Last year as an experiment I deliberately set an exercise in such a way that it was very easy for students to copy one another's work; such plagiarism was easy to detect but impossible to prove. The fact that it was impossible to prove was made very clear to students. In these circumstances I estimate that more than 20% of students submitted work with which they had little personal involvement. Students whom I questioned about this were clearly embarrassed about it, but expressed little remorse. The issue of plagiarism in distance learning is a complex one and is discussed in more detail below.
- About half the students are male and half female; about half describe their ethnic background as 'black African', about 40% as 'white' and the rest as an even distribution of 'black Caribbean' and 'Asian'. Of course the distance-learning students will not have the same ethnic composition.

#### **1.4.2 Cultural issues**

The fact that this course is to be studied in distance learning mode in several different countries (it is currently running in Egypt and Hong Kong as well as London) means that we have to be even more careful than usual to avoid racial and cultural offences. Of course we always want to do this, but face-to-face contact allows us the opportunity to show that any accidental lapses are not intentional insults. With distance learning we do not have the opportunity to apologise for oversights. One of the objectives of the proof-reading and review process we employ is to spot possible inadvertent discriminatory language and examples.

#### **1.4.3 Study time**

School policy on student study time limited the amount of time I could *expect* students to study this course to 90 hours, over a total course duration of 10 weeks. Of course, some students spent more time than this, and some less.

### **1.5 Objective**

The objective of this work was to design and evaluate a first course in Java programming that could be taught in a 'supported distance learning' environment, to

students whose backgrounds are as described above.

## **2 Theory**

### **2.1 Course design philosophy**

The philosophy behind this course is very similar to that described by [Wester et al., 1997], with the exception that I did not want to introduce students to a sophisticated visual programming environment. The main characteristics of this course, like Wester's, are as follows.

- It emphasizes the construction of complete applications, with commercial-quality user interfaces
- It introduces the principles of object orientation from the very start

Understanding object orientation is the key to producing good quality (as opposed to merely functional) programs in Java. There is no doubt that, when object orientation is fully mastered, it becomes much easier to write Java programs. Because our students have no prior exposure to programming, I felt that dealing with object orientation would not present any additional problem for them – it is mostly a problem for more experienced programmers who have to adapt to a new way of working. We hoped that the use of object orientation would ultimately make the subject easier, as well as instilling good working practices. Various authors have reported the use of specific teaching aids for object-orientation [e.g., Dershem and Vanderhyde, 1998, Sangwan and Korsh, 1998], but such techniques were felt to be unnecessary in this teaching context.

An advantage of Java over other languages is that it is very easy to construct very pleasant-looking user interfaces (provided that the concepts of object orientation are mastered; Java expresses its user interfaces in terms of objects). We hoped that this very visible sign of mastery would be a motivating factor for students.

### **2.2 Model of interaction**

The model of student interaction adopted by the course was a simple one, and based on the following, well understood, didactic principles.

- Effective learning requires reflection [Bruner, 1966, Ausubel, 1968]. Students should think about their learning experiences and relate them to existing knowledge.
- Effective learning requires application of skills in a realistic context [Honebin et al., 1993]. In the context of this course I take this to mean that learning to write computer programs requires the writing of computer programs

With these principles – which are hardly controversial or novel – in mind, the model of interaction shown in figure 1 was adopted.

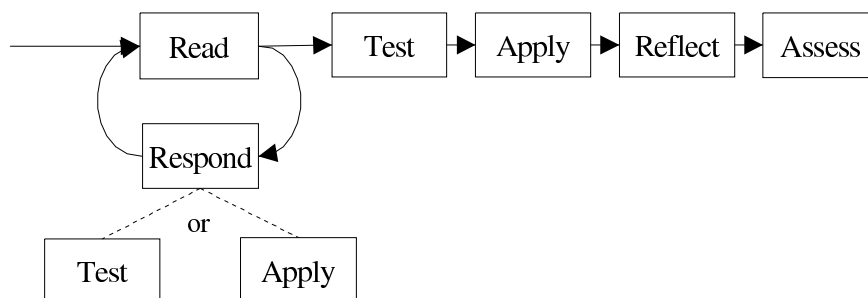


Figure 1: The student interaction model

### 2.3 Course structure

The course was divided into ten ‘units’, with the intention that students would study one unit each week for ten weeks. The units were not self contained; on the contrary each unit was intended to build upon the knowledge and skills learned in the previous one. Each unit had the structure shown in the figure. The learning material was divided into material to read, and opportunity to respond. These responses were either a ‘test’ of some kind (that is, an exercise designed to allow the student an opportunity to assess his or her factual and conceptual knowledge), or an ‘application’. In the context of this course, and following the recommendations of [Honebin et al., 1993], ‘application’ was taken to mean either writing a small computer program in Java, or analysing a program written by myself. These ‘test’ and ‘apply’ exercises were intended to take ten to fifteen minutes each, and there were usually eight to ten in each unit. The reading material was either presented on a CD-ROM or referred to a specific chapter of the recommended textbook.

After these read/response cycles there are some longer ‘test’ exercises (usually analysis of a computer program), some longer ‘apply’ exercises (always involving some actual programming), and a ‘reflect’ exercise. This was usually a series of questions based on the course material, but requiring deeper conceptual analysis. Some of the ‘reflect’ exercises were intended to be carried out in discussion with other students, using Internet bulletin boards or ordinary face-to-face discussion where possible. Finally, there was a short self-assessment exercise. For administrative reasons – it can be marked by computer – this was a multiple-choice test.

At the start of each unit is a chart showing the proportion of the total time that we expected the student to spend on each type of study (reading, exercises, etc).

The overall structure of the course material corresponds loosely to David Kolb’s classical ‘cycle of learning’; Kolb’s ‘active conceptualization’ and ‘active experimentation’ phases are represented in the programming exercises, while the reading, and analysis of programs corresponds to Kolb’s ‘concrete experience’ phases. Kolb’s ‘reflective observation’ has its counterpart in the ‘reflect’ and discussion parts of the course.

Each unit’s content was determined by formal learning outcomes, rather than

by syllabus content. In fact, there is no formal ‘syllabus’ at all for this course; it’s content is determined by its learning outcomes. For example, the learning outcomes for unit five (whose title is “Using Strings and arrays”) are:

“After successfully completing this unit you will be able to:

- explain the differences and similarities between strings and arrays of characters
- declare an array, define, interrogate and change individual elements of an array
- declare and work with instances of the String class
- implement Java programs illustrating the manipulation of arrays and strings”

Note that all these outcomes are testable, and the ‘reflect’ exercise and final self-assessment exercise are intended to test them.

## **3 Methods**

### **3.1 Implementation of course material**

The course materials were developed in the form of a World-Wide Web site, but were distributed to students on a CD-ROM. The CD-ROM also contained the Java compiler software that students would need to be able to write their own programs. It was intended that students should install this software on their own computers. Three different authors contributed material for the CD-ROM, all of whom were experienced teachers. The material was then edited and proof-read before being committed to CD-ROM. As this was the first time I had developed this kind of material, I expected that there would be some errors. My colleagues and I therefore maintained a Web site on a live server for the benefit of students who wanted to see the most up-to-date version of the text. The intention was to update the server to reflect improvements and corrections in the material. In practice, however, the great majority of students preferred to use the CD-ROM, even with its errors, than spend a lot of time on-line (a sample of the on-line course material can be seen at [ericgill.mdx.ac.uk/class\\_modelling/](http://ericgill.mdx.ac.uk/class_modelling/))

Students also had access to a bulletin board (for message posting), e-mail, references to extensive on-line reference and tutorial materials.

#### **3.1.1 Student assessment procedure**

In the distance learning environment we have, by definition, little or no personal contact with students. This has implications for the methods of assessment that are used. MSc students have already shown themselves to be willing to indulge in plagiarism if they get the chance, and a fair scheme of assessment has to make this very difficult.

When teaching the course in ‘traditional’ mode last year, I used the following forms of assessment (all of these were used during a single run of the course).

- Timed computer programming exercises carried out in a computer laboratory under close supervision and in examination conditions
- A number of short (30 minute) tests of theory carried out in examination conditions
- A set of computer programmes to write, of graded difficulty, and for which marks were awarded for the number of exercises completed over the complete course. All students did the same exercises.
- A larger programming exercise carried out without supervision, where each student worked on a different programme that was agreed with myself in advance.
- A formal, unseen examination

In order to eliminate the possibility of plagiarism in at least part of the assessment, some of the exercises were carried out in examination conditions, although they were not ‘exams’ in the traditional sense. Apart from imposing these conditions, it is impossible to be absolutely certain that the work was really done by the student; after all a number of people make a living by providing coursework for students, and these ‘services’ are well publicised. With the assessment scheme described above I felt that plagiarism was quite limited, because it would have been more trouble than it was worth. Although students could have ‘commissioned’ someone to provide at least part of their coursework, or copied work from their fellow students, they were well aware that in the end they would be demonstrating their work face-to-face to a person that they knew to be quite aware of their capabilities. In these circumstances I suspect that for most students outright plagiarism would not have been very attractive. Nevertheless, I don’t doubt that some students obtained assistance from outside the course.

However, it is ultimately impossible –I believe – for coursework to be designed that makes plagiarism more trouble than it is worth in the distance learning environment. By definition we have little or no personal contact with students, and this removes the main disincentive for plagiarism. This view was held so strongly by all staff involved in the MSc programme that a policy was made that required at least 70% of the student’s final grade to be contributed by a single unseen examination. Other kinds of examination that may have been more effective, such as seen and open-book exams, were ruled out on the grounds that we have no control of the examination procedures in the our supporting overseas institutions, and these would be difficult to regulate.

All this presents something of a problem for assessing a programming course, and this problem has not been solved satisfactorily. It is very easy to set an examination that tests a student’s factual recall, and not very hard to set one that

tests the student's superficial understanding of theory. It is extremely hard to set an examination that tests deep comprehension of difficult concepts. It is this deep comprehension that makes it possible to develop useful computer programmes, and without it nothing of value can be achieved in this subject.

Moreover, because the 'coursework' component of the course now carries a relatively small contribution to the final grade, it is difficult to set an exercise that is extensive enough to be interesting.

A further problem in the distance learning environment is the varying cultural interpretation of terms like 'plagiarism'. Even in the West, there is a surprising difference of opinion between students and academics, and among different students, about what constitutes 'plagiarism' [Ashworth et al., 1997]. To reduce this problem, the course material includes a very detailed statement about what constitutes reasonable practice *in the context of this particular course*.

### **3.2 Evaluation**

The new course was taught to 66 local students as part of an MSc programme. There is no reason to think that the student body was significantly different in background or capabilities from the students who studied the course last year in 'traditional' mode, so this group is used for comparison.

At the time of writing we do not have information about students' examination performance in the new course, so evaluation had to be on the basis of student feedback, and completion rates. I obtained feedback from students in three ways. First, by informal discussion during and after classes. I tried as far as possible to obtain comments from the complete (local) student body. Second, at a formal meeting of student representatives that was chaired and minuted. Unusually, the students' comments and concerns were similar in the these feedback exercises. Third, by e-mail.

## **4 Results**

### **4.1 Student feedback**

I received, and replied to, over 600 e-mail messages during the ten-week run of the course. If divided evenly among the 66 students this would amount to about nine messages per student. However, the division was far from even, in terms of students or of time.

Less than half the students contacted me by e-mail at all. Less than ten percent contacted me more than once. However, a few students contacted me more than twenty times each.

It was notable that less than 10% of these e-mail communications were about matters that I would consider to be 'academic'. Over 90% were administrative, and concerned difficulties in installing software, inability to find things on the CD-ROM, and inability to get access to computer resources. The following is an exam-

ple of such an e-mail; clearly this student is trying to get to the root of the problem, but lacks the knowledge even to follow the step-by-step instructions provided.

Kevin

I am trying to install JDK on my computer at home using the instructions on the CD ROM. I have completed step 1 but am having problems with step 2.

I click start and control panel. I can find system but there is no environment within this, so I can't set the PATH.

My computer is an NT machine.

Am I missing something?

Thanks

Fred

I did not anticipate this level of helplessness and dependency among MSc students, and clearly this is something I will have to think very carefully about for the future.

Feedback from students in person tended to emphasize the following points, some of which will be discussed in more detail later.

- More than half the students said that it was quite difficult to follow course materials displayed on a computer screen, and printed materials would have been preferred.
- Only about 20% of students said that there were able to follow the course at the pace originally intended. About 60% of students finished the course in twelve weeks (which is the maximum that was allowed), rather than the ten weeks that was originally planned. About 20% of students never completed the course, in the sense that they were still studying material designed for week nine or earlier when time ran out.
- Almost all students claimed to be spending at least eight hours per week on the course, and about 10% of students claimed twice this much. Many students expressed difficulty in balancing the demands of this course, other courses, and their work and domestic commitments.
- Almost all students said that most of the time they spent studying this course was spending reading, rather than in carrying out programming exercises.

- Most students were positive about the new style of teaching, although some were cynical about the University's motivation for adopting it. A number of students were of the opinion that we were engaged in a cost-saving exercise (nothing could be further from the truth; this course was shockingly expensive to develop). Many students were appreciative of the opportunity to work at their own pace, although in practice there was not that much flexibility in this respect.
- Most students, regardless of their overall performance, were of the opinion that our expectations of their prior experience and knowledge were unrealistically high. About 20% of students, who were making less progress than their colleagues, claimed that unrealistic expectations were the cause of this situation. When asked to account for the fact that some of their colleagues were making better progress, most of these students claimed that their classmates had more prior programming experience. In fact this is manifestly untrue; only one student out of 66 had *any* prior experience.

## 4.2 Completion rates

Because this course was designed for distance learning, a number of students decided that they preferred to study entirely away from the University. About 40 of the 66 students attended every tutorial class. Three students withdrew from the course completely (this is similar to the withdrawal rate – four out of 115 – with traditional teaching last year).

# 5 Discussion

## 5.1 The use of interactive materials

From the outset it was envisaged that students would interact with the on-line material in a well-defined way. We chose the order of the reading, exercises, reflection and tests very carefully in an attempt to lead the students through the material in a rational way. In practice, however, students tended to focus on the reading, to the exclusion of everything else. This was despite the fact that each week's material started with a statement of the amount of time we expected students to spend on each component. Clearly the students did not follow these guidelines very closely. Because the students tended to see study as primarily an exercise in reading, they regarded the CD-ROM as an information resource, like a textbook. However, it is not very easy to read long sections of text on a computer screen, and I did not envisage students doing this. Most students expressed a wish to have the CD-ROM material in a paper form, which would have completely defeated the interactivity.

Of course some reading is essential, but there are a great many excellent textbooks on this subject. In future we may get better results by *reducing* the amount of reading material on the CD-ROM, and just giving more references to textbook

pages. Another approach may be to prevent the students getting access to more reading material until they have completed all the exercises in a given unit.

The fact that students tend to view study as being dominated by reading is a symptom of the broader problem of adapting to a new learning paradigm, as discussed below.

## **5.2 Expectations of students' prior experience and knowledge**

Problems in this respect became apparent almost from the first day of the course. For example, to be able to execute Java programs, students must be able to *compile* them. This means that they must be able to install the compiler software. I mistakenly considered this to be a trivial task, especially as the students had access to detailed step-by-step instructions written by myself, and even more detailed instructions on the supplier's Web site. In fact, not only could about 20% of students not install this software unaided, six students rendered their computers inoperable in the attempt. The reason for this latter problem was rapidly discovered, but not easy to communicate via the bulletin board and e-mail. In any event, the affected students had access to neither of these facilities, as their computers were inoperable. The solution turned out to be very difficult to describe over the telephone as well. When I questioned students about this, a number claimed to have *never* installed software on a computer; many said that this was a task that normally caused them some difficulty.

Although this course is intended for students with no programming experience, it has been quite difficult to deal with the concomitant lack of general technical experience. For example, the course material uses terms like 'megabyte', 'applet', and 'file', assuming that students will know what these terms. Very often they have a superficial understanding. For example, a student may well be able to state that a megabyte is a million bytes (or thereabouts), but have no idea how much storage this represents compared to the total memory available to a computer program. This should not really have been a surprise; it is by learning programming that people gain an in-depth understand of many technical aspects of computing.

## **5.3 Adapting to a new learning paradigm**

Moving from a learning environment which is 'teacher-centred' to one which is 'student-centred' is likely to be a problem for students who have been exposed only to the former. The use of distance learning materials is very much a 'student-centred' environment, in that students must take greater responsibility for planning their study, seeking information and, most importantly, articulating their problems.

In this course, students were less adaptable to a new mode of study than we anticipated. Most students did not take much advantage of facilities like e-mail, bulletin boards, or on-line reference sources. It is possible that our 'supported distance learning' scheme does not offer the best of the teacher-centred and student-centred philosophies as was intended; perhaps the existence of tutorial sessions prevents

students from making the transition to self-motivated study by allowing students to ‘save up’ their problems for the next tutorial session, rather than using the on-line communications facilities.

However, teaching this particular subject to the (non-technical) students on the course *without* explicit face-to-face communication between teacher and student is almost inconceivable. Students are often unable to articulate their difficulties, and need a skilled tutor to get to the root of the problem. Very often the e-mail messages sent by students display what appears to be a staggering lack of comprehension of the subject, but on closer examination it turns out that the student lacks only the ability to express a technical problem in clear prose.

That students were unable to make full, effective use of on-line resources is broadly compatible with the findings of [Soloway and Wallace, 1997], and [Lawhead et al., 1997], who reported a number of Internet-based teaching experiments that had been less successful than expected. This is clearly an area that needs further development.

#### **5.4 Problems with exercises**

Although students were able to carry out most of the exercises set in the interactive material, in many cases their notion of ‘satisfactory completion’ was very different to mine. In some cases, students completed exercises without grasping the principles they were intended to demonstrate. To an extent I exacerbated the problem by including ‘hints’ on carrying out the exercises. It is difficult to write ‘hints’ on programming exercises that do not, to a certain extent, give the game away. This encouraged students to construct an answer from the hints, rather than grappling with difficult problems.

In addition it is hard, with a large group of students, to ensure that all students are getting the full benefit of the exercises that they do complete. Some scheme needs to be developed that allows students to verify that they have extracted from each exercise the key knowledge and skills.

### **6 Recommendations**

1. Interactive material needs to be prepared in such a way that the value of the interactivity is emphasized to students. In this course, I could usefully reduce the amount of reading material, and perhaps prohibit access to later units of material until a satisfactory number of exercises from the last unit has been completed. This prohibition needs to be enforced by software, rather than by a tutor.
2. In distance learning, it is much harder for students to find the meaning of an unfamiliar term or concept. In a first programming course, we need to be aware not only of the students’ lack of programming knowledge, but of their lack of knowledge of technical computing in general. Providing a technical

glossary will go some way towards helping students cope with these difficulties, but ultimately I need to revise the way in which some parts of the course are written.

3. If students are expecting to get some face-to-face teaching, then the amount of on-line assistance on the interactive exercises should not be too large.
4. If students are expected to communicate with tutors and each other by e-mail, for example, then they need to be encouraged to start doing this from the outset of the course. Merely providing the facilities is not sufficient; use of on-line resources needs to be an integral part of the course, perhaps linked to assessment. This may be even more important where a mixture of on-line and face-to-face contact is provided, as students will tend to wait for a face-to-face meeting to air their problems.

## References

- P Ashworth, P Bannister, and P Thorne. Guilty in whose eyes? university students' perceptions of cheating and plagiarism in academic work and assessment. *Studies in higher education*, 22(2), 1997.
- DP Ausubel. *Educational psychology: a cognitive approach*. Holt, Rinehart and Winston, New York, 1968.
- JS Bruner. *Towards a theory of instruction*. Norton, New York, 1966.
- HM Deitel and PJ Dietel. *Java: how to program*. Prentice-Hall, 1996.
- HL Dershem and J Vanderhyde. Java class visualization for teaching object-oriented concepts. In *Proceedings of the ACM technical symposium on computer science education*, pages 53–57, Atlanta, 1998. ISBN 0-89791-994-7.
- PC Honebin, TM Duffy, and BJ Fishman. Constructivism and the design of learning environments: context and authentic activities for learning. In TM Duffy, J Lowyck, and DH Jonassen, editors, *Designing environments for constructive learning*. Springer-Verlag, Berlin, 1993.
- PB Lawhead, E Alpert, G Bland, L Carswell, D Cizmar, J DeWitt, M Dumitru, ER Fahraeus, and K Scott. The web and distance learning: what is appropriate and what is not. In *Proceedings of the ACM conference on integrating technology into computer science education*, pages 27–37, Atlanta, 1997. ISBN 1-58113-012-0.
- RS Sangwan and JF Korsh. A system for program visualization in the classroom. In *Proceedings of the ACM technical symposium on computer science education*, pages 272–276, Atlanta, 1998. ISBN 0-89791-994-7.

E Soloway and R Wallace. Does the internet support student inquiry? don't ask. *Communications of the ACM*, 40(5):11–16, 1997.

F Wester, M Sint, and P Kluit. Visual programming with java: an alternative approach to introductory programming. In *Proceedings of the ACM conference on integrating technology into computer science education*, pages 57–58, Uppsala, Sweden, 1997. ISBN 0-89791-923-8.