

The Redesign of the Delivery of an Introductory Programming Unit

Michael Jones

Bournemouth University

Abstract: This study examines the motivation of a cohort of first year computing students and the effects on their engagement as they progress through the revised delivery of an introductory programming unit. The purpose behind the redesign of the delivery of the unit was to maintain student engagement by ensuring that each student was able to assume responsibility for all their work, irrespective of their previous experience. One hoped-for consequence was the avoidance of the bi-modal results profile experienced in previous years. The level of engagement was measured in terms of the numbers of students submitting assignments, given that not all the assignments needed to be attempted to obtain a pass mark for the coursework component of the unit. A survey was conducted at the outset, at which it became clear that a substantial minority of the students were not particularly interested in programming. For these students the motivation was to pass the unit. The number of students expressing either some or considerable interest in programming in the survey almost directly matched the number completing a final, optional assignment. Student surveys and the lack of a bi-modal distribution in the results indicated that the revised delivery programme maintained the general level of motivation and engagement within the group throughout the academic year.

Keywords: curriculum design; introductory programming; student engagement; teaching programming

1. Introduction

In Computing programmes, programming is a problematic subject. At first glance one might assume that every student signing up to a computing programme would be interested and motivated to learn to program. Those delivering such units understand that the reality is much more complex. Within programmes, plagiarism is a major concern and appears widespread (given the number of websites offering to write applications for students). Increasingly, the non-programming components of computing are gaining prominence, including multimedia and networking. The link between 'computing' and 'programming' seems to be becoming weaker, further widening the attitudes and aspirations of those enrolled on an introductory programming unit. The result is a need for the design of introductory programming units to be revisited (Feldgen and Clua, 2003), and to focus on the needs and priorities of the individual (Gill and Holton, 2006).

Jenkins (2001:56) suggests that most students enrolled in a programming unit are extrinsically motivated, as evidenced by high rankings given to the need to progress, and in the hope of a good job. The complicating factor is that the motivational categories (e.g., intrinsic, extrinsic) are not mutually exclusive, and that, if one could measure each of these categories within a given student, not only would the value be continuous, but it would probably change over time. That study focused on student attitudes; this study intended to identify the links between the motivation prior to the unit, and student performance within the unit.

The fundamental nature of programming plays a significant part in the design of a delivery programme. Dijkstra (1989) exhorted teachers to focus on the scientific aspects of computing, avoiding the trivialising of important concepts, in the face of the diversity of students and their aspirations. Knuth used 'the art of programming' as the generic title in his seminal volumes on programming techniques (1997, 1998a, 1998b). A resolution is offered by Mody (1992) using music as an analogy. Computing requires the involvement of the artist in the evolution of the vision of the artefact, combined with the analytical approach of the scientist to minimise faults. This somewhat 'renaissance' view of computing in general, and programming in particular, does nothing to reduce the complexity faced by tutors designing delivery programmes for introductory programmes.

2 Programming in the Context of the Programme in Question

In addition to these concerns, it had been noticed that students on the computing programmes at the author's institution exhibited unexpectedly low levels of motivation regarding programming. The assessment regime for many years had consisted of 100% coursework, of which 20% was formed from two in-class tests, with the remainder being constituted by the completion (in the student's own time) of two specified applications. Over the years, the weighting on the second of these applications was increased to 60%, to ensure that the students had to attempt at least some part of it.

The programme was reviewed, and two major changes were agreed with respect to programming. The first was the re-introduction of an examination (50% of the overall assessment of the unit) to focus on some of the conceptual issues, and the other was the re-introduction of Java to replace C.

The first delivery of the revised scheme produced much the same results in terms of student engagement. The same coursework assessment regime was used, with much the same results. Most of the students had built up a reasonable number of marks from the tests and the first program, which then meant that they only needed to complete part of the final assignment to pass the coursework element of the unit. It was clear from the applications submitted that very few students had fully engaged with the second assignment. The results from the examination mirrored this finding, with a positively skewed set of results with a mode fairly close to the pass mark.

3. The Redesign of the Delivery

The decision was taken to completely redesign the delivery of the unit. The aims and objectives of the unit remained the same, leaving the intended learning outcomes and the unit content unchanged. The pedagogical framework was revisited, with the result that a more balanced relationship with Bloom's taxonomy resulted (Scott, 2003). Rather than mainly focussing on synthesis, greater emphasis would be placed on comprehension, and application, as well as analysis and evaluation. This realignment of the pedagogy, similar to that advocated by Denton *et al.* (2005), was supported by a number of tenets, which were used to inform the redesign:

1. *Programming is a creative activity, not a cypher-based activity.*

If one is automating an activity for which an algorithmic solution already exists (e.g., solving the n-queens problem), then the role of the programmer is to act as a cypher, transcribing the components of the solution into code. A great deal of programming today is creative, where the understanding of the processes involved tends to emerge symbiotically as the application develops. In addition to games, website development and systems integration are domains in which the programmer must play an active and creative role in the development process.

2. *Programming requires continuous application*

Problem solving (including the comprehension of existing solutions) involves complex cognitive processes which require interleaved periods of stimulation and assimilation.

3. *The critical elements of programming are: comprehension, manipulation, and construction*

The presence of large software libraries means that systems integration is a significant (and increasingly large) part of modern programming. It can be argued that being able to understand and tailor code is at least as important as being able to construct solutions from simple components. The meanings of manipulation and construction overlap with a number of Bloom's terms, including application, analysis, synthesis, and evaluation.

4. *Competence in comprehension and manipulation should be sufficient to pass the unit*

Given the time it takes to achieve technician competence (typically 5 years – the length of a traditional apprenticeship), and the growing importance of comprehension and manipulation, the suggestion is that forcing students to produce working solutions increases the likelihood of collusion during the implementation phase, which may be seen as negative by staff and other students.

5. *A single pattern – termed the fundamental programming pattern – forms the basis of all imperative applications*

This pattern is a sequence of three operations: create a collection, populate the collection, and iterate through a collection (to produce secondary data). In the context of the delivery of the unit, 'arrays' were frequently referred to as 'fixed-size collections' with the collections classes often being termed 'variable-sized

collections'. Part of the motivation for this was to emphasise the concept of a collection without descending into the vagaries of Java syntax. Also, there are programming languages (notably those with variable-sized associative arrays) in which the distinction is rather different.

This pattern is both a simplification and an abstraction of the programming patterns identified by Proulx (2000). It was felt that introducing a large number of inter-related patterns may confuse, especially as each can be considered an implementation (or a specialisation) of the fundamental pattern.

6. *Delivery design must take cognisance of 'Einstellung'*

Problem solving involves creating, refining, and rejecting solution strategies. Individuals abstract solution strategies from sample problems they are asked to solve. Einstellung (McCloy *et al.*, 2007) is the name given to the psychological phenomenon whereby individuals are reluctant to reject a solution strategy, even after a more efficient or effective solution presents itself. To minimise Einstellung, one must ensure that the initial examples presented to the students are typical of ones they will find later.

7. *Important concepts should be covered more than once*

This is rather different from the tradition of programming units in which students work through a set of exercises, many of which focus on the current topic. In the revised delivery, the same concept was covered in distinct parts of the delivery, each time from a different perspective – that of a particular application.

8. *'Mathematical' programming should be avoided*

Traditionally, most computer programmers were mathematicians, or individuals fairly comfortable with mathematical notations and concepts. Increasingly, mathematical applications form a smaller subset of all applications. In imperative applications, the data processing model predominates.

9. *Time-based collections must be avoided*

In terms of the redesigned unit, all references to 'time-based collections' were removed, as they use advanced concepts which are not widely applicable. Time-based collections involve the use of a single-valued variable within a block, to represent the 'array'. For a more detailed discussion, see Jones (2007a).

4. How the Tenets Informed the Design

1. *Programming is creative and intensely individual*

Students were able to select which applications they attempted, and select whether to construct new applications, or demonstrate comprehension of existing applications. They also had some freedom to set their own assignments, i.e., using Free Programming Projects (FPPs).

2. *Progress in programming benefits from continuous activity*

The number of programming assignments was doubled to four. Electronic submission and semi-automated marking was used to minimise the time involved in marking and returning work to students. Automated marking was rejected, as that would have contravened the underlying premise that programming is essentially creative.

3. *Comprehension, manipulation, construction*

Tests continued to form 20% of the coursework component of the unit, although the number was doubled to 4. Of these, three were online, and multiple-choice. The number of assignments involving programming was doubled to four, and their nature was changed, from a single, specified application, to a small portfolio of programs. For these 'portfolio' assignments, students were allowed to submit applications they had manipulated, as well as applications they had written. To ensure that the source of the manipulated applications was known, a Java Source Code Obfuscation System (JSCOS) (Jones, 2007b) was constructed. This system dispensed working applications which had a number of transformations applied, none of which introduced syntactic or semantic errors. Each supplied application had an embedded identification string, which was linked to the particular student. The student had to undo the transformations to demonstrate comprehension. The only penalty for students taking this route, was that they needed to supply more programs. The idea was to provide every student with the opportunity to be responsible for all their own work, thus helping to maintain their motivation. The option of submitting these supplied applications applied only to the first two of the four portfolio assignments.

4. *The fundamental programming pattern*

Apart from the very first application (a sequence of 'Hello world' and 'Goodbye world'), all applications encountered by the students included the use of at least one

collection. As each new application was considered, the main questions were: how many collections, what type (fixed or variable), and the type of each item. The idea was to plant in each student's mind the inextricable link between collections, loops and applications.

5. *Managing Einstellung*

The main technique for managing Einstellung is to ensure that no concept needs to be un-learned. For example, even the classic 'Hello, World' application is open to the false abstraction that all the surrounding paraphernalia needs to be repeated for each statement. Including 'Goodbye, World' in the first program assists the students in making the correct abstraction that the paraphernalia of 'public', 'class', and 'main' relate to the application, not the statement. Similarly, the avoidance of time-based collections means that there is a much greater chance that the students will make the correct abstraction that loops and collections are symbiotically associated.

6. *Important concepts should be covered more than once*

Apart from collections, the other programming concepts repeatedly covered were the desire to apply a sequence of instructions to multiple data sources, and the need to organise program components into manageable elements – i.e., delegating operations using methods and aggregating methods into classes. Each of these three central concepts were covered at least three times at different points in the delivery of the unit. To emphasise this, the delivery of the units was divided into four, each of which had two assessment components – a test and a set of programs. All submitted programs were demonstrated to a marking tutor. The demonstrations also included a short question and answer section, and required the student to modify the code in some way.

7. *Mathematical programming should be avoided AND*

8. *Time-based collections should be avoided*

As collections were introduced early on, there was no need to use time-based collections – i.e., loops in which a single-valued variable is recreated on each iteration. The result was that there was always a clear syntactic distinction between single-valued variables and multi-valued (collection) variables.

5. The Delivery Programme

The revised delivery programme consisted of four blocks, each of five weeks. In each week of this 20 credit unit, there was a one hour lecture and a three hour supervised workshop. Each block included one test and one portfolio assignment as forms of assessment. The themes of the blocks were: programming principles, statements and basic graphics, methods and variable-sized collections, and an introduction to object-orientation. Each block covered loops, collections, methods, and classes. As Java arrays are not true classes, this somewhat limited the ability to term the delivery 'objects-first'. A decision had previously been taken by the programme team to adopt Java as the sole programming language for the unit, mainly to enable students to gain specific skills appropriate to a year-long placement they complete during the third year of the programme. In these respects, the programme has similarities with the most common ones identified by Schulte and Bennedsen (2006), although no IDE was used here. The programme for the unit was broadly sympathetic to that outlined in CC2001 (ACM/IEEE, 2001), although there was much less emphasis on mathematical rigour, in line with the 'creative' tenet. The main focus for specification and testing lies with other units.

6. The Cohort

The cohort originally numbered 79 students. They completed a questionnaire at the start of the unit which ascertained their previous programming experience, and their preconceptions of programming. There were a combination of multiple-choice and Lickert-scale questions. Most of the students (63%) indicated they were very or fairly interested in becoming professional programmers, although very few had much more experience than a few small applications in Visual Basic.

7. The Learning Materials

The early introduction of collections, and the merging of the concepts of 'arrays' and (variable-sized) collections is not mirrored in conventional textbooks. This necessitated the development of a number of learning materials, most of which were available online. A number of fully hyperlinked tutorials were produced, which complemented the worksheets, all of which had links to specific tutorials, and the relevant Sun manual pages. Also, students had access to online learning materials for additional support. These proved

useful for reference and explanatory purposes.

8. Tool Support

To support the greater emphasis on comprehension and manipulation, a number of software tools were constructed to support the delivery and assessment of the unit.

1. *Java Source Code Obfuscation System (JSCOS)*

This delivered transformed source code for use both in the learning process, and for submission in assignments.

2. *The Java Method Definition Workshop (JMDW)*

This online system assists the student in comprehending method definition by both presenting sample definitions which the student describes, and by presenting descriptions for which the student needs to supply a definition. For more information, see Jones (2007c).

3. *The Electronic Assignment Submission (EAS) system*

The conventional online submission system was augmented by validation of the incoming (zip) file, to ensure that all the required files were submitted. This simplified the marking and demonstration processes.

4. *The Marking Assistant System*

This system unzips and compiles all the submitted code, producing diagnostic files which are used later in producing feedback. The other components of this system enable the tutor to readily view the submitted code, to produce individualised comments, and to produce a feedback document - a PDF file - for each student from a spreadsheet and from the individual comments. These processes were completed in a few days, enabling the documents to be uploaded within a week of the assignment submission.

5. *The Demonstration Assistant*

This simple system provided the student with access to the submitted code, which could then be used by both the student and the tutor in the demonstrations.

9. Measuring Motivation 1 – the Initial Survey

An anonymous online survey was conducted at the outset of the unit. 62 of the 79 students (78%) responded to the initial anonymous survey, conducted online in the second week of the first term. Four respondents had had a reasonable experience of Java, whilst a further

16 had written at least one application, leaving 42 who had not used the language. Most respondents had some experience of Visual Basic, although 18 had none, and a further 18 had limited experience (i.e., had written no more than 3 applications). The technologies which most respondents had used were web-based, with about half having a website. 9 respondents had no prior programming experience.

39 respondents (63%) indicated that their plan was to become a software developer, 15 were uncertain of their future career path but felt that programming might be fairly useful, whilst 6 indicated that they viewed programming only as part of the degree. (Two respondents skipped this question). This indicates a considerable *ab initio* range of motivation within the cohort.

10. Measuring Motivation 2 – the Student Surveys

Surveys of the reactions of the students to the unit were conducted towards the end of the first term on behalf of the Programme Leader, and immediately prior to the examinations, on behalf of the university. In both cases, the student response was generally excellent, with the continuous nature of the assessment being particularly praised. The university system, the Student Unit Evaluation (SUE) produced an overall score of 4.3 (out of 5). Direct comparisons with previous years are not possible, as a different set of questions had been used. In general terms, no previous cohort had produced a figure above 4.0 for the Programming unit.

11. Measuring Engagement

The main instrument selected to measure student engagement through the unit, was that of the numbers of students submitting each of the 8 assignments (4 tests and 4 sets of programs). The tests were each worth 5% of the coursework mark, and all sets of programs weighted equally. The rubric for the coursework mark was made up from the four test scores plus the best three of the four sets of programs. This rubric was chosen so that, if a student only wanted to pass the coursework, he or she could miss one or all of the tests, and at least one of the portfolio assignments, especially the last.

This quantitative approach was selected for two primary reasons. Firstly, to provide some basis on which future judgements could be made (as the history of the problems of

engagement in the unit provided no such basis). Secondly, to allow maximum time for the creation and testing of the support tools and the learning materials.

66 students sat the examination, a 16.5% attrition rate. There was a steady withdrawal rate through the year, indicated by the declining numbers of submissions for the assignments. These were: 79, 74, 73, 69, 71, 65, 67, and 33. Note that the number submitting the last portfolio assignment (assignment 8) was much lower than for the other assignments. This is in line with what might be expected, given the profile of student motivation identified in the initial survey, and given that students were informed of their overall coursework mark as assignments were completed. The number submitting the final assignment was anecdotally reduced (according to the verbal comments of some students) by the proximity of a deadline for a double assignment affecting two other units, which was a week before that for the final portfolio assignment. Only 2 students missed assignment 6 and did assignment 8 instead. One student who attempted the final assignment transformed his 'fail' grade into a pass. It seems reasonable to assume that intrinsic interest in programming was the main motivation for the other 32 students who submitted the final assignment.

In each of assignments 6 and 8, one of the options available to the students was for them to specify their own project (a Free Programming Project) around a set of meta-requirements. In addition, students were allowed to work in groups in the last assignment, to enable them to tackle more ambitious projects. Ten students selected the FPP option for assignment 6, with thirteen doing so for the last assignment. Nine of the thirteen worked as individuals, the other four worked in two pairs.

The end of unit examination consisted of a compulsory question worth 50% of the total. It was divided into four sections, the first two of which focused on code comprehension. The third section involved a small manipulation of the code supplied. The final section involved writing a snippet of code from scratch. 85% of students passed this question.

12. Conclusions

The consistent pattern of student performance throughout the unit, extending to the examination, implies that the revised delivery scheme achieved the desired goal of

maintaining motivation of all students (by not alienating either those with or those without a passion for programming), and hence avoiding a bi-modal distribution.

It is very clear from the data that there is a significant sub-group within the cohort of students who are motivated by programming only to the extent of successfully completing the unit. Both the numbers completing the final (optional) assignment, and the anonymous questionnaire completed at the outset, indicate that the extent of this less motivated sub-group is approximately one student in three. Given the similarity between the results for the questionnaire and the completion of the final assignment, it is reasonable to assume that the extent to which the delivery regime changed students' initial opinions and attitudes was fairly limited.

The challenge remains to define and refine the appropriate assessment regime. Requiring students who are not enthused by the idea of programming to produce code to a predefined specification, would appear to almost encourage collusion and plagiarism. In turn, this will lead either to an increased support for those students (to reduce the reliance on unacceptable support sources and mechanisms), or more staff effort being expended on plagiarism detection (Joy & Luck, 1999) and the associated costs and disciplinary processes (Maurer *et al.*, 2006). Whether this group could be more engaged by a greater use of Free Programming Projects would appear to be doubtful.

REFERENCES

- ACM/IEEE (2001) Computing Curriculum 2001. ACM/IEEE Computing Committee on Computer Science.
- Bloom, B. S., et al. (1956). Taxonomy of Educational Objectives: The Classification of Educational Goals. Handbook I: The Cognitive Domain. McKay Press.
- DENTON, L. F., MCKINNEY, D. & DORAN, M. V. (2005) A Melding of Educational Strategies to Enhance the Introductory Programming Course. Proceedings of the 35th ASEE/IEEE Frontiers in Education Conference. Indianapolis, IN, ASEE/IEEE.
- DIJKSTRA, E. (1989) On the Cruelty of Really Teaching Computer Science. Communications of the ACM, 32, 1398-1404.
- FELDGEM, M. & CLUA, O. (2003) New Motivations are Required for Freshman Introductory Programming. Proceedings of the 33rd ASEE/IEEE Frontiers in Education Conference. Boulder, CO, ASEE/IEEE.
- GILL, T. G. & HOLTON, C. F. (2006) A Self-Paced Introductory Programming Course. Journal of Information Technology Education, 5, 95-105.
- JENKINS, T. (2001) The motivation of students of programming. Proceedings of the 6th annual conference on Innovation and technology in computer science education. Canterbury, United Kingdom, ACM Press.
- JOY, M. & LUCK, M. (1999) Plagiarism in Programming Assignments. IEEE Transactions in Education, 42, 2, 129-133.
- KNUTH, D.E. (1997a) The Art of Programming Volume 1: Fundamental Algorithms (3rd Edition). Addison-Wesley.
- KNUTH, D.E. (1998a) The Art of Programming Volume 2: Seminumerical Algorithms (3rd Edition). Addison-Wesley.
- KNUTH, D.E. (1998b) The Art of Programming Volume 3: Sorting and Searching (3rd Edition). Addison-Wesley.
- MAURER, H., KAPPE, F. & ZAKA, B. (2006) Plagiarism: A Survey. Journal of Universal Computer Science, 12, 8, 1050-1084.
- MCCLOY, R., BEAMAN, C. P., MORGAN, B. & SPEED, R. (2007) Training conditional and cumulative risk judgements: the role of frequencies, problem-structure and einstellung. Applied Cognitive Psychology, 21, 325-344.
- JONES, M. (2007a) Time-based Collections Considered Harmful in Introductory Programming. Available at: <http://dec.bournemouth.ac.uk/staff/mjones/Reports/2007-TBC.pdf>
- JONES, M. (2007b) The Java Source Code Obfuscation System (JSCOS). Available at:

- <http://dec.bournemouth.ac.uk/staff/mjones/Reports/2007-JSCOS.pdf>
- JONES, M. (2007b) The Java Method Definition Workshop (JMDW). Available at:
<http://dec.bournemouth.ac.uk/staff/mjones/Reports/2007-JMDW.pdf>
- MODY, R. P. (1992) Is programming an art? ACM SIGSOFT Software Engineering Notes, 17, 19-21.
- PROULX, V. K. (2000) Programming patterns and design patterns in the introductory computer science course. IN HALLER, S. (Ed.) Proceedings of the thirty-first SIGCSE technical symposium on Computer Science Education. Austin, Texas, United States, ACM Press.
- SCHULTE, C. & BENNEDSEN, J. (2006) What do teachers teach in introductory programming? Proceedings of the 2006 international workshop on Computing education research. Canterbury, United Kingdom, ACM Press.
- SCOTT, T. (2003) Bloom's taxonomy applied to testing in computer science classes. *Journal of Computing in Small Colleges* 19, 1 (Oct. 2003), 267-274.