The Effectiveness of Assessment Learning Objects Produced Using Pair Programming

Andrew Adams, Jude Lubega, Sue Walmsley and Shirley Williams
School of Systems Engineering, The University of Reading, UK,
a.a.adams@rdg.ac.uk
j.t.lubega@rdg.ac.uk
sue.walmsley@rdg.ac.uk
shirley.williams@rdg.ac.uk

Abstract: Pair Programming is a technique from the software development method eXtreme Programming (XP) whereby two programmers work closely together to develop a piece of software. A similar approach has been used to develop a set of Assessment Learning Objects (ALO). Three members of academic staff have developed a set of ALOs for a total of three different modules (two with overlapping content). In each case a pair programming approach was taken to the development of the ALO. In addition to demonstrating the efficiency of this approach in terms of staff time spent developing the ALOs, a statistical analysis of the outcomes for students who made use of the ALOs is used to demonstrate the effectiveness of the ALOs produced via this method.

Keywords: Learning Objects, Assessment, Blended Learning, Pair Programming

1. Introduction

Learning objects [LOs] (Wiley 2000a) are a new design concept for "learning content": digital entities suitable for reuse. Instructional Designers build small instructional components that can be re-used in different contexts. An example learning object would be a short introduction to Boolean logic. This object could be incorporated into many subject areas: mathematics; philosophy; computing; engineering. Related objects could include a multiple choice test on simple propositional logic tautologies. Where a LO is designed for assessment it can be called an Assessment Learning Object [ALO].

Many of the different approaches to Instructional Design are based on software development methods. For a range of software development project types, traditional software engineering methods have begun to be regarded as too "heavyweight", given their lack of flexibility. This led to the development of various "lightweight" or "agile" approaches to software design. Combinations of such approaches are often grouped under the title "eXtreme Programming" [XP] (Beck 2000). One of the most commonly used elements of XP is "Pair Programming", which involves two programmers working on a small independent section of a program. One programmer "drives" the development, typing in the actual code, while the other combines the role of "navigator" (deciding which direction the "driver" takes next) with that of "driving instructor" (providing instant feedback and review of the code being produced). The two programmers periodically switch roles and, when necessary, brainstorm solutions to tricky sections.

At The University of Reading a blended learning (Lubega and Williams 2003) approach is taken to the separate modules:

- Introductory Programming
- Programming
- Functional Programming

and the students are experienced in using the *Blackboard* Managed Learning Environment (Blackboard 2003). For academic year 2002/3 the course structure at The University of Reading changed: first year results had been previously available in May, while students were still attending classes, but from 2003 they were only available at the end of June, just as students departed for the summer vacation. Resit examinations were still held in late August, meaning that students who had failed and needed to revise had eight weeks, but all of it away from the academic environment. A number of formative ALOs (i.e. quizzes) were therefore deemed necessary for each module, so that students could guide their revision from the existing module notes and practical tasks on Blackboard.

A learning object approach was decided upon, so that the material could be reused in future courses by a variety of students. A Pair Programming approach was deemed highly suitable for producing this material since it gave the combination of swift production together with high quality levels: revision tests to be taken by students away from the academic environment must be completely correct (Adams et al 2003).

For each of the modules the development process was recorded along with a comparison between prior experience of developing Blackboard content alone and the new pair programming approach. In each case a third party reviewed the quality and correctness of the learning material produced.

The content of the ALOs were all developed as Blackboard pools of questions. From these pools individual questions were selected and placed into short tests (ALOs) that exercised particular learning outcomes for a module.

Student usage of the ALOs was monitored and compared with both the original and resit exam results in order to check the utility of the ALOs for supporting self-directed revision tasks.

2. Learning objects

Wiley (2000b) states: "Learning objects are elements of a new type of computer-based instruction grounded in the object-oriented paradigm of computer science."

Within Computer Science the terms 'object' and 'object-oriented' are widely used in a variety of contexts, including: design methods, programming languages and systems (Sommerville 2001).

In 1996 the Institute of Electrical and Electronic Engineers (IEEE) established the Learning Technology Standards Committee (LTSC) to develop standards, practices and guides for learning technology (IEEE 1996), working formally and informally with other groups from around the world to ensure global standards. One of their working groups (IEEE 2001) provided the following definition:

> *Learning Objects are defined here as any entity, digital or non-digital, which can be used, reused or referenced during technology supported learning. Examples of technology supported learning include computer-based training systems, interactive learning environments, intelligent computer-aided instruction systems, distance learning systems, and collaborative learning environments. Examples of Learning Objects include multimedia content, instructional content, learning objectives, instructional software and software tools, and persons, organizations, or events referenced during technology supported learning.*

The term ALO is used to describe objects specifically associated with assessment (summative or formative) such as a multiple choice test.

3. Pair programming

The concept of pair programming is part of the wider idea of XP (Beck 2000), which has one of those slightly nebulous definitions where sometimes it is not clear whether a particular process is XP or not. XP involves a number of different software production methods. Individual projects will use some or all of the methods to a greater or lesser extent. Pair programming is one of these methods. The central philosophy of XP can be summed up by this quote from the Series preface to the Beck book:

> *Although XP is often presented as a list of practices, XP is not a finish line. You don't get better and better grades at doing XP until you finally get the coveted gold star. XP is a starting line. It asks the question "How little can we do and still build great software?"*

Given the time constraints on academics, the application of analogues of XP in preparation of teaching material seemed obvious to the authors. Pair Programming is one of the most commonly used aspects of XP. The website Object Mentor (2001) defines pair programming as:

> *Two programmers working side-by-side, collaborating on the same design, algorithm, code or test. One programmer, the driver, has control of the keyboard/mouse and actively implements the program. The other programmer, the observer, continuously observes the work of the driver to identify tactical (syntactic, spelling, etc.) defects and also thinks strategically about the direction of the work. On demand, the two programmers can brainstorm any challenging problem. Because the two programmers periodically switch roles, they work together as equals to develop software.*

The aim of pair programming is to ensure productivity of the programmers by avoiding mental blocks and ensuring attention to both detail of the current procedure and to the overall scheme, and to improve code quality by avoiding syntactic and semantic errors (mis-typing a variable name, using the wrong variable or procedure call, using a "while...do" instead of a "repeat...until" loop etc).

Writing a computer program and writing interactive learning objects (and in particular writing formative assessment objects) have many similarities. In particular it is necessary to produce high quality output. In programming terms the program must do what is required and only what is required in an efficient form, with a suitable level of modularity and re-use of code. In the production of ALOs the individual questions and answers must be accurate and a suitable coverage of the topic at hand must be achieved.

For computer scientists familiar with a variety of programming methods, pair programming seemed a suitable method for applying to the problem of quickly and efficiently producing sets of questions for interactive formative ALOs.

4. The modules

In this study three related modules on programming were considered. A similar approach to teaching and learning was used with each.

4.1 Introductory programming

This module is intended for students with little or no experience of programming. The main thrust of this module is to provide the student with a working knowledge of the basic methods of Imperative (traditional) Programming. Topics covered include

- The representation of data, including use of arrays, records, arrays of records, sets, components, basic objects and basic pointers;
- Program constructs such as conditionals, loops, functions and procedures and their use;
- Design, implementation, testing and debugging of a program to solve a problem.

The language Object Pascal, in the Delphi Rapid Application Development environment (Williams and Walmsley 1999), is used for practical work.

4.2 Programming

This module is intended for students with a prior experience of programming, for example at a standard equivalent to the English A-Level or BTEC. The content includes the syllabus for introductory programming as described above, and also

- Use of components from standard, additional and dialogue tabs of Delphi IDE;
- Exception handling;
- Advanced data structures (static and dynamic);
- Object-oriented and modular programming;
- Cross-platform development using Kylix and CLX .

Because the Programming and Introductory Programming modules had considerable common content the question pools were shared between them. Collectively these two modules are referred to as Delphi Programming.

4.3 Functional programming

This module aims to give students a parallel experience of Unix usage and functional programming abilities, to complement the Windows-based Delphi programming modules. It presents programming in Caml Light (Cousineau and Mauny 1998) on the Sun Unix systems. First, the students are given exercises in using the Solaris operating system and Open Windows environment. The Caml Light interpreter and a text editor for maintaining programs are introduced. Elements of functional programming are introduced in appropriate order including: values, expressions, types, records, pattern matching, higher order functions, lists, recursion, polymorphism. The final section of the module concentrates on using the functional language to implement more complicated examples.

4.4 Teaching and learning methods

Each module consists of around 50 contact hours over 8 months. In a typical term time week there are either two lectures per module, or a lecture and an exercise class, or similar. Each student has a weekly hour-long supervised practical per module, and access to a suitable laboratory at other times.

Practical assignments are given out regularly; advice will be available in practical sessions. Exercise classes assist the student to determine the correct use of programming constructs, and answers are distributed at the close of the class.

The on-line Blackboard system is used to support student learning (Lubega and Williams 2003). Material routinely used includes: lecture notes and assignments; discussion groups; questionnaires; general information.

5. Experience of using pair programming for the development of ALOs

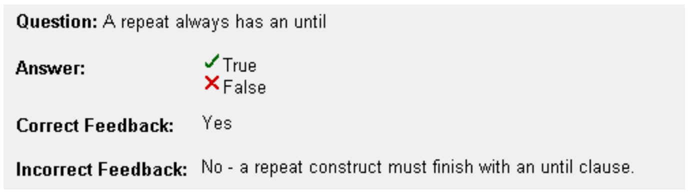
It was found that the pair programming approach allowed much quicker, and more efficient, production of material, even when only one of the pair was an expert on the subject matter, while the error rate was reduced drastically.

5.1 True/false questions

It was decided that the first ALOs to be created would be for the Delphi Programming. The pair of module designers (B and C) working on this were both experienced in the field, and had worked together on three programming text books (Walmsley and Williams 1990 and 2002; Williams and Walmsley 1999) and had taught as part of a team in the past. It was decided that true/false questions could be quickly created and would be a good revision aid. The same approach would then be applied to the Functional Programming. The pair here would be made up of one of the Delphi Programming pair (C) – to allow transfer of experience from the first set, and the lecturer in charge of the module (A).

5.1.1 Delphi programming

In one hour 20 questions and appropriate answers were designed. See for example Figure 1.


Question: A repeat always has an until

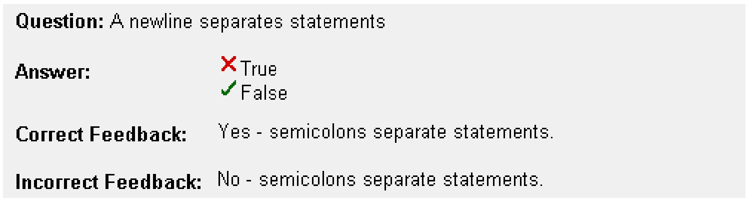
Answer: ✓ True ✗ False

Correct Feedback: Yes

Incorrect Feedback: No - a repeat construct must finish with an until clause.

Figure 1: A true/false question

In this example no explanatory feedback was deemed necessary for the correct answer. In a number of examples the same explanation was provided whether or not the right answer was selected. See for example Figure 2.


Question: A newline separates statements

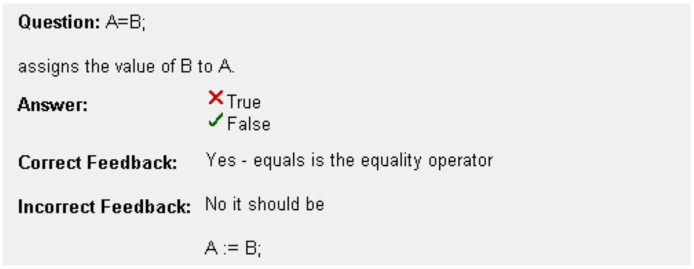
Answer: ✗ True ✓ False

Correct Feedback: Yes - semicolons separate statements.

Incorrect Feedback: No - semicolons separate statements.

Figure 2: Explanatory Feedback

This was thought appropriate for students who had just guessed the answer. With other questions slightly different explanations were appropriate depending on the choice made by the students (see Figure 3).


Question: A=B;

assigns the value of B to A.

Answer: ✗ True ✓ False

Correct Feedback: Yes - equals is the equality operator

Incorrect Feedback: No it should be

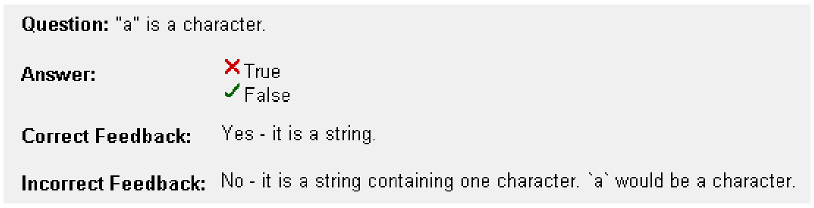
A := B;

Figure 3: Longer explanation

The questions were then reviewed. A few errors were identified, both typographical and logical. The logical errors almost all were due to the fact that the pair had recently been preparing material related to the C programming language.

5.1.2 Functional programming

In the first hour 26 questions and appropriate answers were designed. These were similar in structure to the ones created for the Delphi Programming, see for example Figure 4.



Question: "a" is a character.

Answer: ✗ True ✓ False

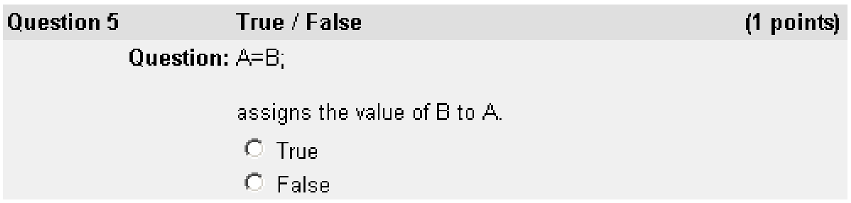
Correct Feedback: Yes - it is a string.

Incorrect Feedback: No - it is a string containing one character. \`a\` would be a character.

Figure 4: Functional programming

5.1.3 *Feedback*

Subsequently a student who had successfully passed both the Delphi and Functional Programming modules trialled some of the assessments and raised issued about the layout of both the questions and the answers. For example in Figure 5 whether it was wise to have the code on the same line as "Question:" and then a blank line. The mix of true/false and yes/no in the feedback was also seen to be confusing (see Figure 6). However because of pressure of time more assessments had been created before this feedback was received. This problem is often also found in rapid development methods used in software. Some methods, such as DSDM (Stapleton 1997) favour including a user in the design team. In this case it would be a student who would use the system. However there are potential difficulties with this approach as these assessments were aimed at students who had already failed the course and such students may not be the most diligent. In addition, since the aim of this was to quickly provide distance-provision of revision material during the students' summer vacation, physical availability of students was almost impossible.



Question 5 True / False (1 points)

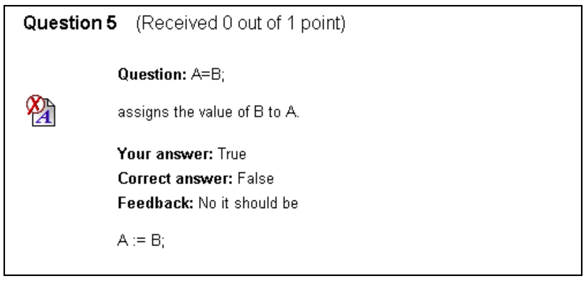
Question: A=B;

assigns the value of B to A.

○ True

○ False

Figure 5: Positioning of the question



Question 5 (Received 0 out of 1 point)

Question: A=B;

assigns the value of B to A.

Your answer: True

Correct answer: False

Feedback: No it should be

A := B;

Figure 6: The student view

5.1.4 *Stylistic issues*

Between the pairs a number of differences were noted. The most appropriate wording for similar questions varied from one assessment to another. Issues of punctuation also varied. With these questions, often based on the syntax of the programming language, care is needed to ensure the use of punctuation in the question does not make the question incomprehensible.

Since rapid production was an important factor the available environment, Blackboard, was used. However there were a number of frustrations in its use, which the choice of an alternative development platform may have avoided. In particular we experienced difficulties with the relatively small boxes for feedback and the inability to quickly see the question as the students would see it.

5.2 Other questions

As well as true/false Blackboard offers a number of different categories of questions:

- Multiple choice questions: allowing the student a multitude of choices, one of which is correct.
- Multiple answers: similar to multiple choice, but the student can choose one or more of answers. Blackboard does not allow questions with no correct choices.

- Fill in the blank: these are evaluated based on a close text match, but the interpretation of the free text input is quite strict.
- Matching: allowing the students to pair questions to answers.
- Ordering: allowing the students to select the correct order the answers are to appear in.
- Short answer essay: allowing students to type and/or cut and paste an answer into a text field.

5.2.1 Multiple choice questions

All the modules already extensively used paper based multiple choice questions for tests and formal examinations. Experience of detractor design made creating these questions relatively straightforward. However the time taken was longer than for true/false: Typically 12 questions were produced in 1 hour.

When working on the Functional Programming, where only one of the pair was an expert in the language, it was found to be much more satisfactory for the less experienced person to be the driver (that is working at the keyboard) whilst the expert navigated. When both of the pair were experts it was best to change roles with every session.

5.2.2 Multiple answers

Early use of multiple answers by the Delphi programming pair had not been successful. The students were more familiar with multiple choice questions and tended to select only one answer. However it was determined that if the question finished with the message:

(Select all that are correct.)

the students realised that there may be more than one answer.

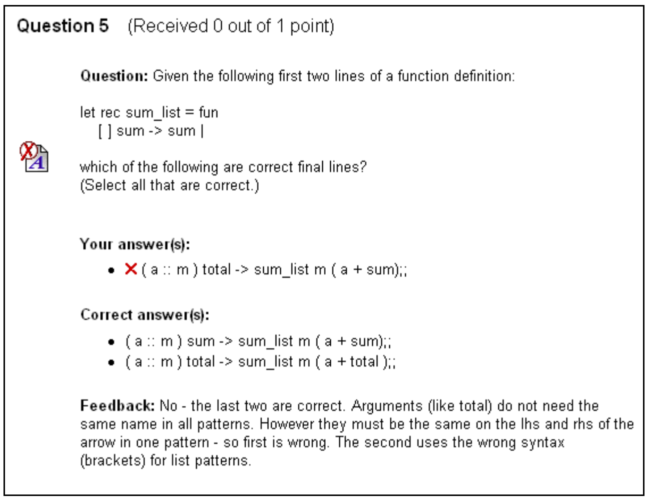
It was later discovered it was impossible to have assessment where none of the multiple answers were correct. It was found necessary to use a final answer that said:

None of the above are correct.

The explanations for multiple answers became more complicated depending on a number of factors. An initial mistake was to explain which of the answers were correct and which not in this manner:

No - the last two are correct. Arguments (like total) do not need the same name in all patterns. However they must be the same on the lhs and rhs of the arrow in one pattern - so first is wrong. The second uses the wrong syntax (brackets) for list patterns.

However when the feedback was displayed (Figure 7) the possible answers were not displayed as they appear when the question is presented, and so the explanation was nonsensical.



Question 5 (Received 0 out of 1 point)

Question: Given the following first two lines of a function definition:

```
let rec sum_list = fun
   [ ] sum -> sum |
```

which of the following are correct final lines?
(Select all that are correct.)

Your answer(s):

- ✗ (a :: m) total -> sum_list m (a + sum);;

Correct answer(s):

- (a :: m) sum -> sum_list m (a + sum);;
- (a :: m) total -> sum_list m (a + total);;

Feedback: No - the last two are correct. Arguments (like total) do not need the same name in all patterns. However they must be the same on the lhs and rhs of the arrow in one pattern - so first is wrong. The second uses the wrong syntax (brackets) for list patterns.

Figure 7: Feedback does not include all the options

5.2.3 Fill in the blank

The aim of both modules was for students to learn how to program. In this context, while true/false and multiple choice questions are useful for students to assess their progress, they are primarily summative in nature, rather than formative. A larger aspect of formative assessment would be useful and initially the

"Fill in the Blank" style of question was regarded as having potential in this area. For instance, giving students a partial function/procedure definition and asking them to provide an appropriate section of code to complete it. It was felt that given the precision required for writing computer programs, that the mechanical nature of checking the submitted answer should not be a problem, as it might be for many other subjects. However, even in this case it was found to be unsatisfactory for general use. The problem is that the "fuzzy matching" algorithm of the Blackboard assessment system is not documented in any easy to find location. One of the programming languages (Object Pascal in Delphi) is not case-sensitive while the other (Caml Light) is. Whitespace (spaces, returns, tabs) are sometimes necessary, sometimes ignored, and sometimes prohibited, depending on the language and context. All of these contribute to a difficulty in providing an exhaustive set of possible correct submissions by students. Apart from a very few simple "fill in the blank" questions for Caml Light, it was felt that even where possible, the use of this question type did not provide sufficient student value for the authors' time spent producing them.

5.2.4 Matching and ordering

Experience of working with special needs students (including those with dyslexia) indicated that the format of these types of questions created accessibility issues and it was decided not to use them.

5.2.5 Short essay

Short essays need to be tutor marked and since the objective of creating these assessments was to allow students to revise without tutor help they were not appropriate.

5.3 Code tracing

Early feedback from the student tester suggested that "code tracing" examples should be included. This tallied with the intent of all the authors who agreed that this was an important skill. "Code tracing" involves providing students with function/procedure definitions and an actual concrete example of applying the function or procedure and asks for either an identification of the final result (Black Box tracing) or the identification of which branches in the code will be executed in the example (White Box tracing). In addition to being an important skill for the students to learn for the direct purposes of debugging their own code (the corollary of the skill being an improvement in their ability to produce code), this is also a necessary skill underpinning the important topic of program testing, which follows on from programming modules in most Computer Science degrees. In particular, white box tracing is particularly necessary for recursive programs, for which Multiple Answer style questions were particularly useful, as a variety of correct (and incorrect) answers could be offered at the same time, encouraging students who are properly engaged in the assessment process to think clearly about the topic.

5.4 General observations

All authors felt that the pair programming approach improved their rate of output. The pair approach tended to enforce a discipline of actually implementing questions on the system. The combination of interactive creativity ensured that new questions and detractors could be easily and quickly produced. The immediate review of the peer, combined with the knowledge of short term testing by an experienced student, provided confidence in the accuracy of the material being published to students. Typographical mistakes were almost always spotted by the navigator and corrected before the question was inserted. This freed the driver to produce the question in its entirety and then return to correct slips of the keyboard, but prevented more serious mistakes wasting valuable author time following up blind alleys leading to unsuitable questions. In particular, very few questions were abandoned once input into the system, a definite difference from author experience of setting such questions for exams or class tests previously. Despite a number of irritations (mostly due to poor interface design) with the Blackboard question setting environment, the experience was a positive one for all the authors, and has produced demonstrably useful results: students who are due to resit the exam are using the tests to aid in their revision.

6. Analysis of student usage

The efficiency of the pair production approach was shown above in terms of the number of ALOs produced in a given period. Even though informal feedback from students to the ALOs was positive, a more rigorous analysis of the quality of the ALOs was desired. Thus, for the Functional Programming module, the marks gained by students on the ALOs was monitored and compared to their results in the resit exam. This cannot absolutely prove the quality of the ALOs since no control group who had access to solo produced ALOs had their results analysed. However, the fact that the analysis does show a correlation between performance on the ALOs and performance in the resit exam demonstrates to our satisfaction that the pair produced ALOs are of suitable quality for their purpose. Since the original aim was to produce ALOs of sufficient quality within tight resource constraints, we feel this goal was shown to be achieved. It should be noted that the "usage statistics" for the ALOs was restricted to that gathered by Blackboard. In this case students were allowed to attempt each ALO as many times as they wished and the only data gathered was their total mark for each ALO on their final attempt.

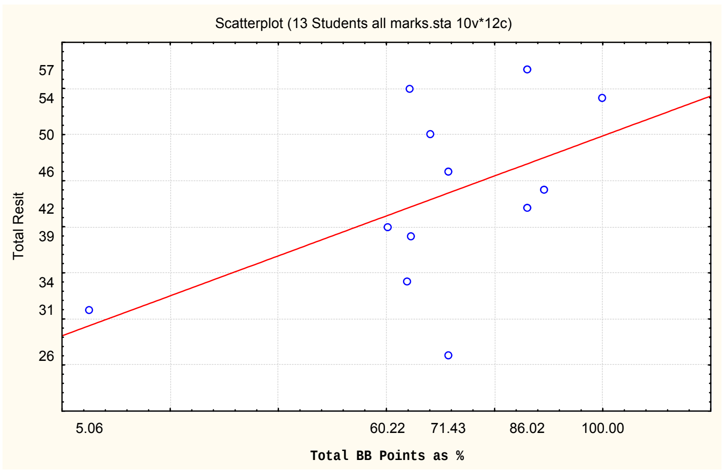
Fourteen students took the resit examination. One of these had not taken the original examination and was excluded as an anomaly from the analysis. The marks obtained in the Blackboard quizzes were analysed with respect to the marks obtained in the resit examination.

Statistic	Summary Value
Multiple R	0.55937
Multiple R²	0.31289
Adjusted R²	0.25043
F(1,11)	5.00918
p	0.04686
Std.Err. of Estimate	16.09483

	Regression Summary for Dependent Variable: Total Resit (Resit and all marks.sta) R= .55936942 R²= .31289415 Adjusted R²= .25042999 F(1,11)=5.0092 p<.04686 Std.Error of estimate: 16.095					
N=13	Beta	Std.Err. of Beta	B	Std.Err. of B	t(11)	p-level
Intercept			17.05606	14.36519	1.187319	0.260109
Total BB Points as %	0.559369	0.249928	0.42659	0.19060	2.238119	0.046855

From the above results we note that there is a positive correlation (Multiple R) between the quiz mark and the mark attained in the resit by the students of 0.55937. Those students who had relatively high marks in the quiz also had relatively high marks in the resit exam. Therefore this is an indication that the quiz was acting like a pretest and whoever failed, meant that he/she was likely to fail in the resit too and the reverse was true. Looking at the relationship between the two performances, we note that the p value =0.04686<0.5 hence indicating that there is a linear relationship between the two performances of the students.

A scatter plot showing the linear relationship is given below:



It should be noted within our examination system: 40% is the pass mark; marks between 30 and 40% are restricted passes that can be compensated by good performance elsewhere.

A wider analysis of the performance of this student cohort with respect to exams and

Blackboard usage can be found in (Lubega 2004).

The correlation between performance on Blackboard quizzes and the result in the resit examination imply (though they do not conclusively prove) that the ALOs derived from our pair production method were useful to students in preparing for their resit exam. Since it has already been shown that the lecturers felt that the pair production approach was a more efficient way to develop such revision aids, this strong indication that the ALOs were useful is good evidence that it is a useful approach. Given the time limitations of the ALO producers (which meant that fewer aids of a poorer "quality" could have been produced using a traditional single producer approach) we believe that this is sufficient evidence on which to recommend such an approach for general usage in the production of ALOs.

7. Future work and conclusions

7.1 Reuse of questions

Although these question pools and assessments have initially been produced with the aim of providing revision support for students who failed their initial examination, it is felt that the questions thus produced will certainly be useful in future years during the initial teaching. For both Functional and Introductory Programming, we expanded the question pools to ensure a minimal coverage of ten questions per topic and provide them as self-assessment for the students during teaching. In particular, the system of producing new tests comprising questions drawn from existing pools or existing tests is useful, as is the concept of a "random selection" of a limited number of questions from a pre-determined pool. In Functional Programming, the tests are open through the year and available for multiple retakes. In the case of Introductory Programming, the tests are set for a single attempt and only available for a short period. The results from Functional Programming are not part of the final assessment of the module, whereas the Introductory Programming tests do contribute (a very small amount, <3% overall) to the students' final mark. We will be analysing the usage of the tests in each module and any links to performance in the end of year examination. Although the formative utility of these tests in isolation is probably quite limited, in combination with students' self-directed exploration of the programming languages' capabilities we feel they should act as a force-multiplier, improving the efficiency and efficacy of student time spent learning.

7.2 Free form questions

An interesting point that arises is whether it would be possible to produce a customisable "fuzzy matching" algorithm which would allow authors to set the context for "fill in the blank" questions. As mentioned above, such questions could be very useful in providing a combination of both formative and summative assessment, particularly in an area where the aim is to improve students ability to apply their knowledge (to synthesis programs), rather than simply know syntactical rules.

7.3 Conclusions

We have demonstrated a novel approach to the production of ALOs using an analogue of the XP software development method of Pair Programming. This has been used in a real situation in order to efficiently produce reusable ALOs initially for self-testing during a revision period and then reused for testing during the main teaching period. Our statistical analysis demonstrates that students appear to have benefited from the ALOs produced, as has informal feedback from the students. Where high quality ALO material needs to be produced under time constraints, we can recommend the pair production approach.

References

Adams, A. Walmsley S. and Williams S. "Pair Programming for Rapid Development of Assessment Learning Objects" in the 2nd European Conference on e-Learning, Glasgow (November 2003)

Beck, K. (2000) Extreme programming Explained: Embrace Change; Addison-Wesley.

Blackboard (2003) http://www.blackboard.com

Guy Cousineau and Michel Mauny (1998) "The Functional Approach to Programming" Cambridge University Press (translator Callaway).

IEEE (1996) Learning Technology Standard Committee (LTSC) http://ltsc.ieee.org/index.html

IEEE (2001) LTSC WG12. http://ltsc.ieee.org/wg12/s_p.html

Lubega, J and Williams, S (2003) "The Effect of Using a Managed Learning Environment on the Performance of Students" to be published in the Proceedings of the International Workshop of Interactive Computer

Aided, Villach, Austria (September 2003).

Lubega, J (2004) "Analysis of Data relating to Students Re-sitting Functional Programming 2003" Internal Report: RUCS/2004/TR/01/001/A, Department of Computer Science, The University of Reading, UK.

Object Mentor (2001) "Pair Programming" http://www.pairprogramming.com

Sommerville I (2001). Software Engineering (6th edition). Addison Wesley.

Stapleton J. (1997) "DSDM: Dynamic Systems Development Method: The Method in Practice", Addison Wesley.

Walmsley S. and Williams S. (1990) "Basically Modula-2", Chartwell Bratt.

Walmsley S. and Williams S. (2002) "Discover Pascal in Delphi" Addison Wesley.

Wiley, D. A. (ed) (2000a). "The Instructional Use of Learning Objects": Online Version: http://www.reusability.org/read/

Wiley D.A. (2000b) Learning Object Design and Sequencing Theory. Doctor of Philosophy Dissertation, Brigham Young University, 2000.

Williams S. and Walmsley S. (1999) "Discover Delphi" Addison Wesley.