Scaffolding with Object Diagrams in First Year Programming Classes: Some Unexpected Results

Lynda Thomas, Mark Ratcliffe and Benjy Thomasson
Department of Computer Science
University of Wales, Aberystwyth, UK, SY23 3DB
+44 (1970) 622424
{ltt, mbr, bjt98@aber.ac.uk}

ABSTRACT
This paper reports on an experiment in which first year programming students were given explicit encouragement to use Object (Instance) diagrams when tracing code in multiple-choice questions. We conjectured that by providing scaffolding in this technique, students would be helped to understand the code better and that they would then continue to draw their own diagrams in similar situations. This turned out not to be the case. Although generally students who draw diagrams do better in questions that test their understanding of code behaviour and object referencing, our *intervention* does not appear to have helped students and the students who were exposed to the intervention were not more likely to go on to use the technique themselves.

Categories and Subject Descriptors
K.3 [Computers & Education]: Computer & Information Science Education – *Computer Science Education*.
D.2 [Software Engineering]: Coding Tools and Techniques – *Object-oriented Programming*

General Terms
Human Factors, Design, Algorithms

Keywords
Object Diagrams, Interaction Diagrams, First Year Programming, Tracing code.

1. INTRODUCTION
In this paper we report on an experiment with first year programming students.. A randomly chosen group of students attempting multiple-choice tracing questions were provided with partially completed Object (Instance) diagrams on paper as a scaffolding mechanism. We conjectured that the students who were given this scaffolding would do better than the control group on this type of test question since they had at least *some* help.

Later in the semester students were studied to see if those who had received this scaffolding continued to use the technique in subsequent tracing problems and if the intervention appeared to have improved their understanding as measured by their performance on such questions.

2. BACKGROUND
There seems to be a general agreement in the CS Education community that many of our students have problems in mastering programming [8]. This is not a new problem[1]. In addition to showing up when students write programs, this difficulty is also apparent when students have difficulty understanding program behaviour.

One of the manifestations of lack of understanding of program behaviour that we have observed in our classes, is that many students do not seem to be able to trace code. Most CS educators will recognise the experience of a student coming to them with a program and saying 'it doesn't work' but not having performed the most elementary trace through to determine why the program does what it does. In particular, in the context of Object-oriented programming, many students do not seem to be able to produce for themselves a diagram that demonstrates an understanding of object references.

Holland regards the misconceptions around object references as one of the most fundamental in learning Object-oriented programming and suggests that "the cleanest way to defuse this misconception is to teach reference as a first class concept …" [6]. In the introductory programming sequence in Aberystwyth, we certainly try to do this, demonstrating program behaviour in lectures and tutorials mainly by drawing pictures of variables and what they reference, as in Figure 1 (essentially rough Object diagrams as described in Section 3). So students *have* been exposed to the idea of tracing through Object diagrams – but when they might appropriately use it themselves, weaker students fail to do so. They are often impatient when the instructor resorts to drawing a diagram, then amazed that the approach works. Even in the more restricted situation of a test that asks the students to trace out what happens when just a few lines of code are executed, as in Figure 1, scratch sheets are often returned blank. In a previous investigation we discovered that only 36% of the sheets were returned with any kind of 'working out' on them [9].

[1] For an excellent summary of research see, for example, [11]

Students must be operating from some kind of construction of knowledge [1] in such situations – but it does not seem to be a productive one. We wanted to increase their understanding and change their behaviour to something more productive. We have observed that instructors and high-performing students nearly always draw some kind of picture when tracing code and it seems reasonable to encourage weaker students to follow this approach. But how can we encourage students to do this themselves? After all they already see their lecturer using this technique in lectures and lab sessions, yet have not adopted it.

Research has shown that methods of teaching and learning that combine significant student autonomy, conjecturing and articulation with dynamic scaffolding by the teacher are highly effective [14]. We conjectured that providing such scaffolding (in the form of partially completed Object diagrams) would help our students understand the concept of object referencing when tracing test code and also encourage them to use Object diagrams with their own code.

Obviously this is a complex field, ideally encompassing both students' understanding of object references and how they debug their own programs. In this paper we report on how we broke off a manageable piece of the field for study. Instead of the unconstrained arena of their *own* programs, we have simply looked at how students approach 'tracing questions' on multiple-choice tests.

3. RESEARCH QUESTIONS

Specifically, we focused on three questions:

1. Is drawing some kind of Object-like diagram correlated with success in solving multiple-choice tracing questions?
2. Does providing students with scaffolding in the form of partially completed Object diagrams help them correctly answer multiple-choice tracing questions?
3. Do students who have been provided with this scaffolding continue to use it in such multiple-choice questions?

In the first question we are looking at diagrams that the students may draw with or without our encouragement. With respect to the second question, we have to consider what we mean by 'help'. If we give students a piece of code and an incomplete Object diagram it seemed very unlikely that they would NOT do better than if we simply give them the code with no help whatsoever (but see later for the results). We wanted to confirm this conjecture but were originally, in fact, more interested in research question 3: whether giving the students incomplete Object diagrams in their early learning is helpful to their later ability to trace code.

4. WHY OBJECT DIAGRAMS?

We have found in our teaching that our students take on board the idea of a class diagram very easily, perhaps because it reflects the syntax of a Java class definition. Diagrams that illustrate dynamic behaviour in a program are more difficult to grasp. The simplest and most intuitive is probably the UML Object (or Instance) diagram [2]. These diagrams essentially provide a snapshot of the objects in a system at some point (see Figure 1). They show, not the static relationship between classes, but the relationship between current objects in a particular program at a particular time. They can be combined with interaction diagrams, which show how objects of different classes can collaborate in a particular behaviour.

Interaction diagrams have been used successfully to visualise Object-oriented programs. Lange and Nakamura have produced a tool to visualise Object-oriented traces that generates the Interaction diagrams (and simplifies them into something meaningful for large systems) from code [7].

Interaction Diagrams are not however the only way of understanding code. Sneed, for example, has used actual source code animation [12]. Hendrix reported on using the more static method of Control Structure Diagrams [5]. Although more complex diagrams are necessary for visualising more complex situations, Object diagrams were sufficient for our needs. In our decision to use these diagrams, we were influenced by the

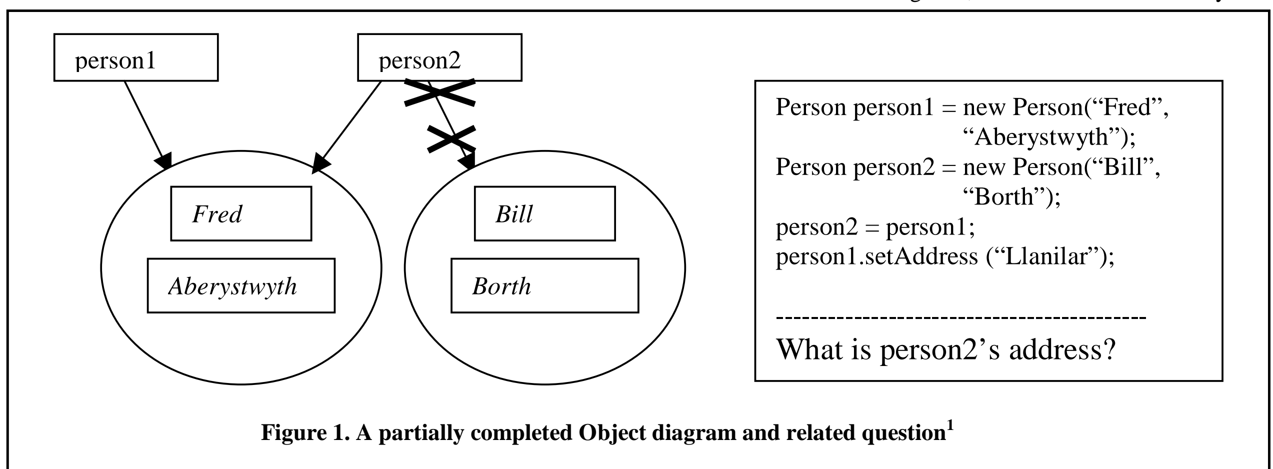

Figure 1. A partially completed Object diagram and related question[1]

[1] Note that this diagram only provides the flavour of the type of question. The actual questions involved two classes that were previously well known to the students and came with some basic code. In addition we are 'cheating' in the representation of the Strings, which are of course really references not primitive values. The idea was to make the diagrams 'sloppy' but effective.

work of Mary Hegarty who has made a study of cognitive models that impact on the understanding of mechanical systems. In a paper with Narayanan [3], they outline a cognitive model of understanding dynamic systems that supposes that the viewer:

- decomposes the system into simpler components,
- constructs a static model by making representational connections to prior knowledge and other components,
- integrates information between different representations (e.g. text and diagrams),
- hypothesizes lines of action, and finally
- constructs a dynamic mental model by mental animation.

Hegarty and Narayanan have empirically validated the design guideline that people learn more from being induced to mentally animate a system before viewing an animation than by passively viewing the animation [4].

If we examine Object diagrams we can see that the first two steps outlined in the Hegarty/Narayanan model are essentially the creation of the basic diagram, and the last three steps involve the actual tracing of the program code with reference to that diagram. This provides some justification that our approach in encouraging students to produce Object diagrams is a reasonable one.

5. APPROACH

Our experiment then was as follows. All the students in the introductory programming class were shown Object diagrams and many examples of their use in tracing code during their lectures.

We divided the class approximately in half for the experiment. One section, the control group, were given pieces of Java code and asked to trace them in a formative test (the intervention test). The other (experimental) group were provided with both the code and an incomplete Object diagram. We compared the results to see how correct the subsequent tracings were (expecting that the experimental group would do better since the control group had no help at all).

Several weeks later we gave students in both groups further questions that involved tracing (as part of the follow-up test). We compared their correctness results again and we also collected their scratch sheets to see if the experimental group used the Object diagram technique with which they had been scaffolded.

There were two sections of this introductory course – one section was comprised of students with little or no programming experience (called 'beginners' here), the other section was comprised of students with considerable high school experience (we call these students 'advanced'.)

6. RESULTS

The experimental group was chosen by where the students sat in the lab. First of all we checked to see that the groups were indeed comparable in terms of performance on previous tests. There was no significant difference in the average group marks up to the point of the experiment (see Table 1).

Table 1. Pre-Test results

Group	n	Pre-Test Average (included non-tracing questions)
Control	51	63%
Experimental	55	58%

With respect to research question 2, when we initially tested the experimental group (given incomplete diagrams) against the control group (given no diagrams and a blank scratch pad), we discovered that Object diagrams helped students trace code, but only for our beginning students and not in a significant way (see Table 2) . The students who had programmed before did not appear to be at all effected by the intervention.

Table 2. Intervention Test results

Group	n	Test Average
Beginners Control	28	28%
Beginners Experimental	40	36%
Advanced Control	23	63%
Advanced Experimental	15	65%

When we looked at the results in the follow-up test, both for tracing questions and the test as a whole (see Table 3), we see that students who had been given the diagrams did essentially the same as the control students. In fact they did slightly, but not significantly, worse.

Table 3. Follow-up Test results

Group	Average just on tracing questions	Test Average (included non-tracing questions)
Beginners Control	68%	58%
Beginners Experimental	56%	56%
Advanced Control	77%	75%
Advanced Experimental	66%	70%

With respect to research question 3, we wanted to see if the students who had been given the diagrams were in fact more likely to use them on the next test (see Table 4). In the beginners' group the answer appeared to be 'no'. In fact, the control group were more likely to create their own diagrams. In the advanced students the figures are reversed.

Table 4. Follow-up Test Behaviour

Group	Created their own diagram on next test	Did not create own diagram on next test
Beginners Control	21 (75%)	7 (25%)
Beginners Experimental	24 (60%)	16 (40%)
Advanced Control	10 (43%)	13 (57%)
Advanced Experimental	11 (73%)	4 (17%)

As we discuss in section 8, these results were very unexpected. In fact, they shook our belief that diagrams are in fact of use in this kind of problem – but when we look generally at students who use diagrams as opposed to those who don't (research question 1), there is in fact an indication that the technique *is* used by higher achieving students. The higher scores achieved by beginners who use diagrams, as opposed to those who do not, comes close to statistical significance (p=.07 with one tailed t-test). But note that advanced students seem to do better without diagrams!

Table 5. Correlation between Diagram Use and Performance

Group	n	Follow-up Test Average - Just on tracing questions
Beginners who do not use diagrams	23	47%
Beginners who do use diagrams	45	68%
Advanced who do not use diagrams	17	76%
Advanced who do use diagrams	21	70%

7. ANALYSIS OF POSSIBLE SOURCES OF ERROR

Since, as we discuss further in the next section, these results were so unexpected, we would like to address possible sources of error.

The student groups do indeed appear to have been chosen randomly (see Table 1).

The Object diagrams we gave the students were hand drawn specifically so that students would be encouraged to recognise the simplicity of the technique and use it themselves.

All the students were shown how to use Object diagrams and could have used them, and, as can be surmised from Tables 4 and 5 many students *did* use them who had not had the intervention. Clearly, students talk to each other about what goes on in the lab, and the control group was able to discover what the experimental group had been given. Possibly that made them *more* likely to create their own diagrams in the follow-up test. This might have encouraged control group students to create their own diagrams but it does not explain why the experimental group of beginners were *less* likely to do so.

We were so shaken by the results of this experiment that we have questioned the utility of Object diagrams, for *anyone*. We can only say that we use them ourselves – even in simple questions such as the ones on the test - and that all our student demonstrators (mainly second and third year high-achievers) do likewise.

8. SIGNIFICANCE OF RESULTS

Quite honestly, the results of this experiment were not what we had conjectured. In the rest of this discussion we will leave experienced students aside because they appear to have been little effected by the intervention and to be doing well enough in this context without any help. So, we turn our attention to the beginners group.

The results in Table 5 provide some evidence to support our belief that, for beginning students, drawing Object diagrams is in fact correlated with understanding object references and being able to trace code.

The poor results on the tracing questions, as can be seen in Table 2, show us that these students *are* in need of help with this kind of problem, but that providing them with what we considered to be helpful diagrams did not significantly appear to improve their understanding, as measured by performance on tracing questions. This was completely unexpected. We thought that we were 'practically doing the question for them' and expected that the experimental group would do brilliantly on the tracing questions. In fact, the students did not appear to gain much from the partially completed diagrams at all.

In addition, as shown in Table 3, beginning students who were given the scaffolding in the first test were *less* likely to then construct their own Object diagrams in the follow-up. At first, we were surprised by this finding. But in light of the previous paragraph it seems more explicable – if the diagrams don't help you, why *would* you construct your own? In fact, the control group students were more likely to construct diagrams. Perhaps they were afraid that they had been missing something in the intervention test and wanted to be sure not to in the next test?

If we return to the discussion in Section 3 we note that by *providing* the students with the diagrams we are perhaps removing the first few steps of Hegarty's model of understanding systems. Maybe it is necessary for the students to build the Object diagram (model) themselves in order to animate it. Scaffolding might be a useful idea – but we may have inadvertently provided the wrong kind of scaffolding, or the wrong level of scaffolding.

An interesting discussion with colleague Raymond Lister leads us to a related possible explanation [Lister, personal communication, June 2003]. He suggests that weaker students cannot systematically create code, or even trace code, because they do not understand the function of the code elements that they are manipulating. Instead they bolt together some coding elements that they have frequently seen but do not understand, using heuristics that they hope will be right often enough to get them a passing grade. Thus, such students can create buggy code, but lack the understanding to debug or even trace through that code. This has been borne out in the third author's work as a lab instructor. In a depressing number of situations he has asked "What does this code do?" and been answered "No idea, XXX told me to put it in - it has something to do with sorting." Lister intends to closely observe student behaviour when tracing code, and we look forward to the results of that research.

We note, however, that this is somewhat contrary to the classic work of Spohrer and Soloway which indicated that students have difficulties not so much in misconceptions about language constructs as in putting the 'pieces' of a program together [13].

9. IMPLICATIONS

This research was undertaken because we were concerned about how to help students understand program behaviour and object referencing. Results were not as we expected. In some sense that is in itself valuable information.

One of the important findings of the ITICSE working paper on assessment of first year programming skills [8], was that educators *believed* that their students could write a certain kind of program, when *in fact* many of them could not. This paper reports on a similar situation with respect to a simple technique for tracing code.

We believed that if we could only get our students to *use* Object diagrams, then all their problems with tracing code would be over. It seems that for, some of our students at any rate, there is a more fundamental problem in that they cannot trace code even when provided with a diagram that in the words of one of our lab assistants "does it all for them!". We have worried that this is "all our fault" as the educators in question, but we feel that the worrying inability to do this kind of tracing among beginning, weaker students is more likely a widespread phenomenon. We welcome others to try this kind of experiment with their own students and to the results of other research on tracing code. In the meantime, we have been led to certain practical conclusions with respect to our own teaching and research.

9.1 Implications for Teaching

This experiment has *somewhat* validated our belief that Object diagrams are useful for understanding the object references in a piece of code. We see the technique as fundamental and will continue to use it and encourage our students to use it themselves.

We now understand, however, that the utility of the technique and students' resistance to it is much more complex than we had realised. Providing beginning students with the diagrams does not appear to either help them much with the problem or to encourage them to use diagrams in the future. We need to convince students to construct the diagrams *themselves* by somehow providing more obvious utility for doing so. This must involve them in always building their own diagrams.

9.2 Implications for Learning Environments

For several years, the research group to which the authors belong has been working on the development of a learning environment for first year students that combines frequent feedback with support for students' explicit awareness of class learning goals and their individual learning styles [9]. We wished to take this further in the direction of supporting student learning [10], but before doing so we needed to determine what would be the most effective direction for such support.

It appears that the technique of scaffolding students in the way outlined in this paper is not as helpful to them as we had hoped. We had been intending to produce Object diagrams in the learning environment so that they might be used on-line by students as they program. We are no longer sure that this is the right approach since the creation of the diagrams appears to be fundamental.

10. ACKNOWLEDGEMENTS

Many thanks to the students and demonstrators of CS12230 and CS12320, academic year 2002-2003!

Thanks also to colleagues from *Bootstrapping Research in Computer Science Education 2003* who helped formulate those pesky research questions!

11. REFERENCES

[1] Mordechai, Ben-Ari. Constructivism in Computer Science Education. in Proceedings of SIGCSE 1998.

[2] Fowler, Martin with Kendall Scott. UML Distilled. Addison Wesley, 2000.

[3] Narayanan, N. Hari and Mary Hegarty. Communicating Dynamic Behaviors: Are Interactive Multimedia Presentations Better than Static Mixed-Mode Presentations? in Theory and Application of Diagrams, Diagrams 2000, September 2000, Michael Anderson, Peter Cheng and Volker Haarslev editors, Springer Lecture Notes in Artificial Intelligence 1889.

[4] Hegarty, Mary, N. Hari Naranayan and P. Freitas. Understanding Machines from Multimedia and Hypermedia Presentations. in J. Otero, A. C. Graesser & J. Leon (Eds.). The Psychology of Science Text Comprehension. Lawrence Erlbaum Associates, 2002.

[5] Hendrix, D., J.H. Cross II and S. Maghsoodloo. The Effectiveness of Control Structure Diagrams, in Source Code Comprehension. IEEE Transactions of Software Engineering, Vol 28, No 5, May 2002, pp.463-477.

[6] Holland, Simon, Robert Griffiths and Mark Woodman. Avoiding Object Misconceptions. in Proceedings of SIGCSE 1997.

[7] Lange, Danny and Yuichi Nakamura. Object-Oriented Program Tracing and Visualization. Computer, Vol. 30 No. 5, May 1997, pp 63-70.

[8] McCracken, M. et al.. A Multi-National, Multi-Institutional Study of Assessment of Programming Skills of first-year CS students. SIGCSE Bulletin, Vol. 34, No. 1, Mar 2002.

[9] Ratcliffe, Mark, Lynda Thomas and John Woodbury. A Learning Environment for First Year Software Engineers. in Proceedings of CSEE&T 2001.

[10] Ratcliffe, Mark, Lynda Thomas, Wayne Ellis, Benjy Thomasson. Capturing Collaborative Designs to Assist the Pedagogical Process. in Proceedings of ITiCSE 2003.

[11] Robins, A., J. Rountree, N. Rountree, Learning and Teaching Programming: A Review and Discussion, Computer Science Education, Vol. 13, Number 2, and June 2003.

[12] Sneed, Harry M. Source Animation as a means of Program Comprehension for Object-oriented systems. in Proceedings of the 8th International Workshop on Program Comprehension, June 2000.

[13] Spohrer, James and Elliot Soloway. Novice Mistakes: Are the Folk Wisdoms Correct? Communications of the ACM, July 1986.

[14] Tanner, Howard and S. Jones. Dynamic Scaffolding and Reflective Discourse: the Impact of Teaching Style on the Development of Mathematical Thinking. in Proceedings of the 23rd Conference of the International Group for the Psychology of Mathematics Education, Haifa, 1999.