Code-Memory Diagram Animation Software for Teaching Computer Programming: contributions for and by dyslexic students

Mark Dixon

University of Plymouth, School of Computing, Communications, & Electronics
Drake Circus, Plymouth, UK PL4 8AA
mark.dixon@plymouth.ac.uk

Abstract

Software development projects are heavily reliant on user-testing to identify and therefore resolve usability issues. However, user-testing is costly and often restricted by limited user availability. Methods have been developed to minimise user-testing, but none can completely replace it. Recently, software was developed to support lecturers teaching computer programming, by animating the impact individual lines of code have on variables in memory. User-testing was particularly important in this educational setting, due to the potentially life long impact of confusion caused by software defects. Difficulties recruiting test-users resulted in a single student being used for a large part of the formative evaluations prior to 'live' use with actual target users (first year students). Results indicated that the software could enhance student learning, and also suggested that this student (who happened to have dyslexia) was able to articulate aspects of the learning and teaching process not mentioned by (but relevant to) other students. However, with only a single student it was impossible to determine whether this was representative of a general trend or a uniquely talented individual.

This paper presents the results of an evaluation of the software with four final year students (two dyslexic, two non-dyslexic) independently following a pre-determined protocol. A large number of important issues were identified (at least one per student). However, both dyslexic students seemed to report more issues in more detail than other students. The issues reported by the dyslexic students were relevant to all students. This suggests that in education (and possibly other domains) the inclusion of a disproportionately high number of dyslexics in user-testing may enhance the evaluation process to the benefit of all.

1 Introduction

1.1 User Testing

User testing is widely regarded as vital to all software development projects, with many concrete examples of large numbers of significant problems being identified and consequently resolved (such as Nielsen, 1993). However, it is often costly and may be inhibited by limited availability of the actual users themselves (Nielsen, 1994). Many methods have been developed as substitutes for user testing (such as expert review, and cognitive walkthrough), but they can only reduce the amount of user testing required, they cannot replace it entirely **(**Dix, Finlay, Abowd & Beale, 2004). The value of user testing is especially important in education, where poor software that confuses students can have a lasting and potentially life long impact.

1.2 User Testing in Education: Code-Memory Diagram Animation software

Over the past three years software has been developed that animates the changes made by individual lines of program code to variables in memory. This code-memory diagram (CMD) animation software was designed to overcome the difficulties of teaching computer programming, especially the limitations of the white board and oral narratives. An initial study of its use in revision sessions of a stage one computing degree module found that it could increase students understanding (Dixon, 2004a). The software was then used over a year long module, which confirmed its ability to enhance student understanding. However, its use was limited by the time taken to create animations, and subsequently the animation editor software was enhanced to address this (Dixon, 2004c). Currently, the software is undergoing a year-long field trial with 4-users (computing lecturers).

Over the duration of the project many evaluations were undertaken with both staff and students. However, access to target users for the two software applications was limited. Computer programming lecturers (the target user for the editor software) are extremely busy, more so recently with changes in higher education resulting in larger numbers of students, and far more variation in background (Biggs, 1999). Initially, the target user group for the viewer was seen as stage 1 computing students, however as the work progressed it became clear that the actual target group was students who don't understand the specific concepts being animated. This presented several problems. Identifying which students didn't understand each of the concepts would require large scale educational assessment (an impractical task). First year students have a tendency to be passive and reluctant to provide constructive criticism of teaching methods and tools. This phenomenon, often referred to as the 'A level' effect (Bryson, 1997), limits their potential as evaluators of software designed to support the learning and teaching process. Also, students with a poor understanding would be particularly vulnerable to any potential confusion caused by software defects. Hence, although lecturers and stage one students were involved, much of the regular informal reviews involved a single final stage student.

1.3 Dyslexic Students as Evaluators

The student who participated in a large portion of the CMD animation software evaluation happened to have dyslexia (a fact that played no part in their selection as an evaluator). They articulated difficulties experienced during the computer programming learning and teaching process that were not mentioned by other students or staff (such as losing position when jumping around in blocks of program code). This led to the suggestion that 'disabled people can identify subtle hidden aspects of human activity, that the target user population is unable to articulate' (Dixon, 2004b, p. 254). However, with only one dyslexic student (who received a far larger exposure to the software that other students) it was difficult to determine whether this was an unusual student, part of a general trend, or related to other factors, such as gender, age, or (lack of) computer programming experience.

1.4 Comparison of Dyslexic and Non-Dyslexic Students as Evaluators

This paper presents a comparison of the feedback given by dyslexic and non-dyslexic students during software user testing. This sought to evaluate the software, as well as providing a better basis for investigating the differences in issues raised by different evaluators (such as dyslexic compared with non-dyslexic). It describes the contributions of the software to dyslexic (and other) students understanding of computer programming concepts, as well as the contribution made by dyslexic (and other) students to the software's development.

2 Method

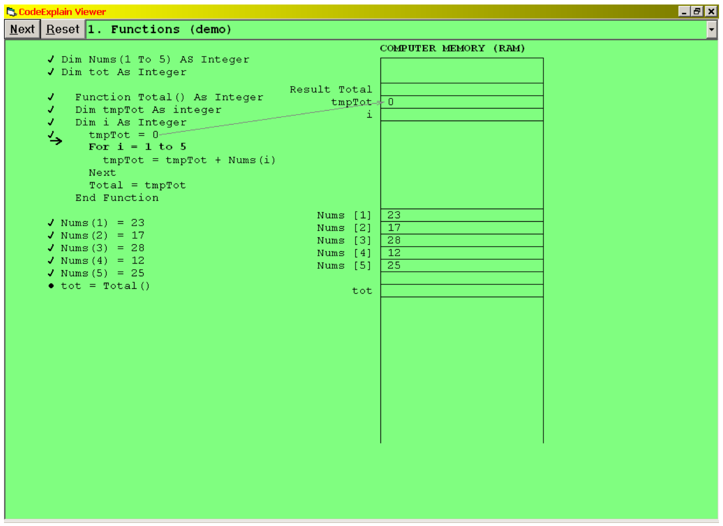
2.1 Participants (Students)

Four final year students were selected on the basis of availability. All students had undertaken modules in computer programming during their undergraduate course, where the CMD animation software was not used. All students were native English speakers, and had an established working relationship with the interviewer/researcher. Two students (one male and one female) had dyslexia. One of the dyslexic students was the student who had been involved in the evaluation of the software since its conception.

2.2 Software to be Evaluated

The students were asked to use and evaluate two versions of the CMD Animation Viewer (Vi1 and Vi2) and two versions of the CMD Animation Editor software (Ed1 and Ed2). A single example was used for all pieces of software, which involved 9 concepts: variable declaration and assignment, array declaration and assignment, local variables, module level variables, a for loop, and a function declaration and call.

Figure 1 shows version 1 of the CMD animation viewer software (Vi1), which was designed as a presentation aid to support narrative explanations of code execution delivered by lecturers to students (Dixon, 2004a). As each line is executed its impact on variables in memory is reflected in the diagram to its right, and a tick appears to its left.

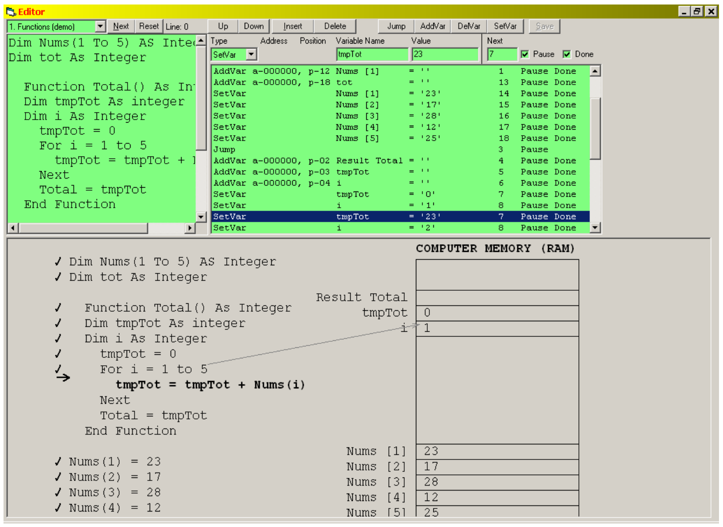


```
✓ Dim Nums(1 To 5) AS Integer
✓ Dim tot As Integer

✓   Function Total() As Integer
✓   Dim tmpTot As integer
✓   Dim i As Integer
✓     tmpTot = 0
→     For i = 1 to 5
        tmpTot = tmpTot + Nums(i)
      Next
      Total = tmpTot
    End Function

✓ Nums(1) = 23
✓ Nums(2) = 17
✓ Nums(3) = 28
✓ Nums(4) = 12
✓ Nums(5) = 25
● tot = Total()
```

```
tmpTot  0
Nums [1]  23
Nums [2]  17
Nums [3]  28
Nums [4]  12
Nums [5]  25
```

Figure 1: CMD Animation Viewer – version 1 (Vi1).

Figure 2 shows version 1 of the CMD animation editor software (Ed1), which was designed to help lecturers create animations and modify them to meet the specific needs of their student groups (described in Dixon, 2004c).



Type: SetVar | Variable Name: tmpTot | Value: 23 | Next: 7 | ☑ Pause ☑ Done

```
Dim Nums(1 To 5) AS Inte
Dim tot As Integer

  Function Total() As In
  Dim tmpTot As integer
  Dim i As Integer
    tmpTot = 0
    For i = 1 to 5
      tmpTot = tmpTot + 1
    Next
    Total = tmpTot
  End Function
```

```
AddVar a-000000, p-12 Nums [1]     = ''     1  Pause Done
AddVar a-000000, p-18 tot          = ''    13  Pause Done
SetVar                Nums [1]     = '23'  14  Pause Done
SetVar                Nums [2]     = '17'  15  Pause Done
SetVar                Nums [3]     = '28'  16  Pause Done
SetVar                Nums [4]     = '12'  17  Pause Done
SetVar                Nums [5]     = '25'  18  Pause Done
Jump                                        3  Pause
AddVar a-000000, p-02 Result Total = ''     4  Pause Done
AddVar a-000000, p-03 tmpTot       = ''     5  Pause Done
AddVar a-000000, p-04 i            = ''     6  Pause Done
SetVar                tmpTot       = '0'    7  Pause Done
SetVar                i            = '1'    8  Pause Done
SetVar                tmpTot       = '23'   7  Pause Done
SetVar                i            = '2'    8  Pause Done
```

```
✓ Dim Nums(1 To 5) AS Integer
✓ Dim tot As Integer

✓   Function Total() As Integer
✓   Dim tmpTot As integer
✓   Dim i As Integer
✓     tmpTot = 0
→     For i = 1 to 5
        tmpTot = tmpTot + Nums(i)
      Next
      Total = tmpTot
    End Function

✓ Nums(1) = 23
✓ Nums(2) = 17
✓ Nums(3) = 28
✓ Nums(4) = 12
```

```
tmpTot  0
i  1
Nums [1]  23
Nums [2]  17
Nums [3]  28
Nums [4]  12
Nums [5]  25
```

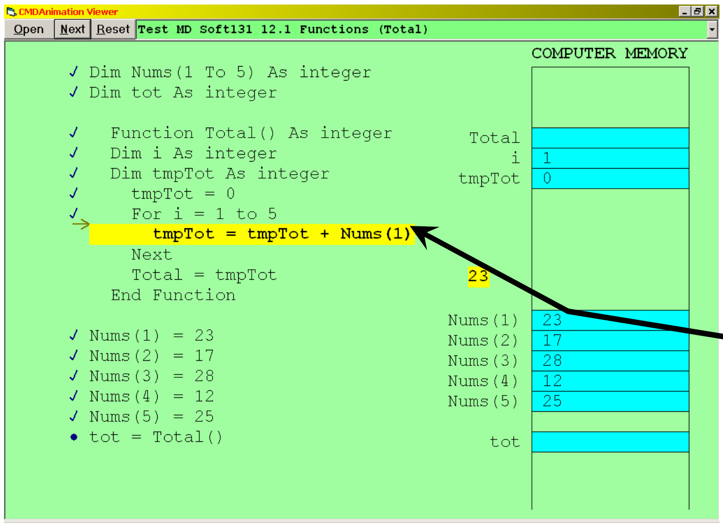
Figure 2: CMD Animation Editor – version 1 (Ed1).

Figure 3 shows version 2 of the CMD animation software viewer (Vi2), which incorporated additional functionality (described in Dixon, 2004c) as a result of suggestions from users (both students and lecturers):

- **Variable font size** – allowed the lecturer to change the size of the font, in order to present the animations as large as possible, while still fitting on the screen.
- **Transitional animation** – animated the process of variable creation (where the variable name flew from the line of code which created it to the position in memory which it occupied), variable assignment (where the value of the expression on the right-hand side of an assignment statement flew from the line of code to the variable on the memory diagram, mentioned in the left-hand side of the assignment statement), and

variable removal (where as a variable went out of scope, it was removed from the memory diagram by flying from the diagram to the line that caused it to go out of scope).

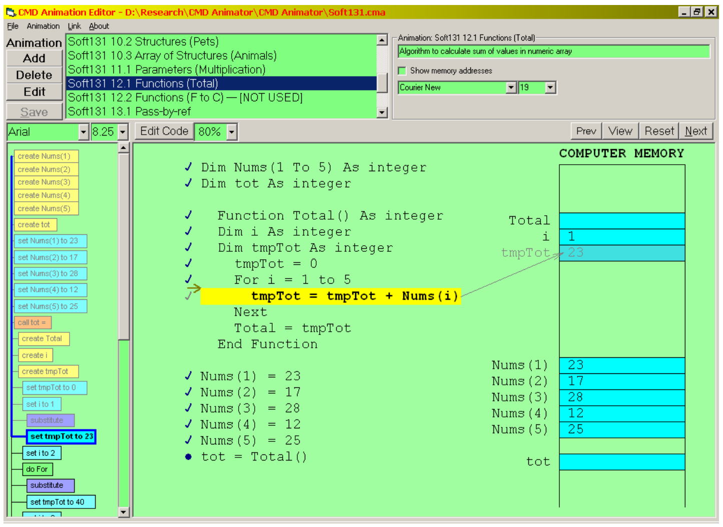
- **Current line highlighted background** – The current line of program code was highlighted yellow.
- **Variable name-value substitution animation** – The process that replaces the names of variables in expressions with their current values was animated, so that the values moved from their position in the memory diagram to replace the names of their respective variables in the line of code which referred to it.



```
✓ Dim Nums(1 To 5) As integer
✓ Dim tot As integer

✓   Function Total() As integer
✓   Dim i As integer
✓   Dim tmpTot As integer
✓     tmpTot = 0
✓     For i = 1 to 5
        tmpTot = tmpTot + Nums(1)
      Next
      Total = tmpTot
    End Function

✓ Nums(1) = 23
✓ Nums(2) = 17
✓ Nums(3) = 28
✓ Nums(4) = 12
✓ Nums(5) = 25
• tot = Total()
```

Figure 3: CMD Animation Viewer – version 2 (Vi2).

Figure 4 shows version 2 of the CMD animation editor software (described in Dixon, 2004c), which employed a direct-manipulation (Shneiderman, 1998) user-interface to increase usability, and hence make the process of creating animations faster and easier for lecturers. The example animation included all 6 frame types available in this version of the editor: create variable, set variable, remove variable, call, do, and substitute (not available in Vi1 or Ed1).



Soft131 10.2 Structures (Pets)
Soft131 10.3 Array of Structures (Animals)
Soft131 11.1 Parameters (Multiplication)
Soft131 12.1 Functions (Total)
Soft131 12.2 Functions (F to C) — [NOT USED]
Soft131 13.1 Pass-by-ref

Algorithm to calculate sum of values in numeric array

create Nums(1)
create Nums(2)
create Nums(3)
create Nums(4)
create Nums(5)
create tot
set Nums(1) to 23
set Nums(2) to 17
set Nums(3) to 28
set Nums(4) to 12
set Nums(5) to 25
call tot =
create Total
create i
create tmpTot
set tmpTot to 0
set i to 1
substitute
set tmpTot to 23
set i to 2
do For
substitute
set tmpTot to 40

```
✓ Dim Nums(1 To 5) As integer
✓ Dim tot As integer

✓   Function Total() As integer
✓   Dim i As integer
✓   Dim tmpTot As integer
✓     tmpTot = 0
✓     For i = 1 to 5
✓       tmpTot = tmpTot + Nums(i)
      Next
      Total = tmpTot
    End Function

✓ Nums(1) = 23
✓ Nums(2) = 17
✓ Nums(3) = 28
✓ Nums(4) = 12
✓ Nums(5) = 25
• tot = Total()
```

Figure 4: CMD Animation Editor – version 2 (Ed2).

2.3 Evaluation Protocol

A protocol was developed to ensure that the same process (in particular the same set of questions) was applied consistently to all participants, and leading questions were avoided (Oppenheim, 1992). The evaluations were conducted independently one student at a time, outside normal teaching sessions. Participation was voluntary and started with the interviewer briefing the participant. Students were invited to 'think-aloud' (Nielsen, 1993), and interrupt the process to comment on any aspects of the software's impact on their understanding of the computer programming concepts in the example animation, as well as aspects of the software that they felt were good and aspects that could be improved. The students were then asked some background questions regarding age, number of years programming experience, and whether they had any registered special needs. The four software applications were then demonstrated, and in the case of the editors the student was invited to use the software. After each software application was demonstrated the student was asked to comment. Audio recordings were made of each session, and transcribed. Recurrent themes were then identified, with particular attention to issues reported by the students regarding difficulties in computer programming learning and teaching, positive characteristics of the software, and criticisms/suggestions for improvement of the software. These themes/issues were emergent from the data (rather than being pre-determined).

3 Results

3.1 Student Experience and Ability

Three students (S1, S2, and Sd2) had about 4 ½ years experience of full-time formally taught computer programming (in all cases 2 years at further education, and 2 ½ years higher education). Student Sd1 had about 4 years experience of part-time formally taught computer programming (2 years at school, and 2 years in higher education). Student Sd1 described learning computer programming earlier at school (from about 8 years old), but with no form of summative assessment. Student S1 described learning computer programming as a hobby prior to being formally taught, and was the only student to indicate that they understood functions 'quite well'. The other three students said they were familiar with functions, but probably would not be able to explain how they worked.

3.2 Computer Programming Learning and Teaching Difficulties

Between them, the four students described 15 difficulties that they had experienced with the computer programming learning and teaching process. Table 1 shows which students reported each difficulty, and the total number of difficulties reported by each student. The students with dyslexia (Sd1 and Sd2) described 6 and 7 difficulties respectively, and the other students (S1 and S2) described 3 difficulties each.

Table 1: Summary of difficulties with the learning and teaching process reported by the students with dyslexia (Sd1, and Sd2) and other students (S1, and S2).

No.	Difficulty	S1	S2	Sd1	Sd2
1	Jumping out of sequence in code		✓		✓
2	Keeping track of current position in code			✓	✓
3	Keeping up with lecturer			✓	✓
4	Explanations on the white board don't explain things clearly			✓	✓
5	Lecturer's don't explain variables	✓			
6	Lecturer's assume too much prior knowledge	✓			
7	Lecturer's overlook advanced aspects of programming	✓			
8	Lecturer sets pace for students who understand already		✓		
9	Students don't understand loops from oral narratives		✓		
10	Hard to keep up if pace is too fast			✓	
11	Thinking you've missed something can distract you			✓	
12	Variable name-value substitution is difficult			✓	
13	Difficulty visualising what code does from oral narratives				✓
14	Object oriented programming is difficult because of jumping				✓
15	Process of program development is not taught.				✓
		3	**3**	**6**	**7**

The following examples are representative of the type of comments made by the four students:

- **Jumping out of sequence in code**: One student (S2) reported being 'confused' with viewer 1 when the current line of program execution 'jumped up to [the function declaration at] the top' as a result of the function call (Total) on the last line. Student Sd2 also described this, but as an example of a common problem (difficulty learning object-oriented concepts is described below).

- **Keeping track of current position in code**: The two dyslexic students (Sd1, and Sd2) indicated that they generally found it 'hard to keep up' with narrative explanations of program code during lectures, and that part of this (with Vi1) was that they found it 'difficult to keep track' of the current line of code.
- **Lecturer's assume too much prior knowledge**: One student (S1) mentioned that 'programming lecturer's … assume too much [prior] knowledge'.
- **Object oriented programming difficult because of jumping**: One student (Sd2) commented that 'object oriented thing … always confused me', and suggested that this was because of the amount of jumping about in the code involved.

3.3 Software: Positive Characteristics

The students described 24 positive characteristics of the four software applications. Table 2 shows which students reported each positive characteristic, and the total number of positive characteristics reported by each student for each software application (Vi1, Vi2, Ed1, Ed2). The students with dyslexia (Sd1 and Sd2) described 15 and 13 positive characteristics respectively, and the other students (S1 and S2) described 7 positive characteristics each.

Table 2: Summary of positive characteristics of the four software application's (Vi1, Vi2, Ed1, and Ed2) described by students with dyslexia (Sd1, and Sd2) and other students (S1, and S2).

No.	Positive Characteristic	S1	S2	Sd1	Sd2
1	Vi1: Visual element	✓	✓	✓	✓
2	Vi1: Step-by-step explanation	✓	✓		✓
3	Vi1: Ticks		✓	✓	✓
4	Vi1: Contents of variables	✓			
	Total Vi1	**3**	**3**	**2**	**3**
5	Vi2: Yellow Highlight of Current Line	✓	✓	✓	✓
6	Vi2: Transitional Animation		✓		✓
7	Vi2: easier where	✓		✓	
8	Vi2: easier to keep up			✓	✓
9	Vi2: Substitution			✓	✓
10	Vi2: Larger Font			✓	✓
11	Vi2: Actually doing			✓	
12	Vi2: Predict what happens next			✓	
13	Vi2: Arrows			✓	
14	Vi2: Memory Variable Background				✓
15	Vi2: Changed perception of substitution				✓
	Total Vi2	**2**	**2**	**8**	**7**
16	Ed2: Drag and drop	✓	✓	✓	
17	Ed2: Font Size			✓	✓
18	Ed2: Help lecturer's understanding of students	✓	✓		
19	Ed2: Colour coded frame nodes			✓	
20	Ed2: Small font for overview of large animations			✓	
21	Ed2: Immediate visual feedback			✓	
22	Ed2: Save button				✓
23	Ed2: Slows delivery				✓
	Total Ed2	**1**	**2**	**5**	**3**
	Overall Total	**7**	**7**	**15**	**13**

The following examples are representative of the type of comments made by the four students:

- **Yellow Highlight of Current Line**: All four students commented positively about the highlighting of the current line in version 2 of the viewer. Student S2 said 'it was a little better than the first one', whereas student S1 described it as 'excellent'. Students Sd2 described it as 'extremely helpful', and student Sd1 commented that it 'leaves you with no doubt as to what line you're actually dealing with'.
- **Help lecturer's understanding of students**: Student S1 commented that the software would 'keep the learning in the mind of the lecturer of the students', and student S2 commented that 'it would be very useful to the lecturer, especially if it's someone who doesn't understand a first year programming student'.

3.4 Software: Negative Characteristics and Suggestions for Improvement

The students described 27 negative characteristics/suggestions for improvement of the four software applications. Table 2 shows which students reported each characteristic/suggestion, and the total number of characteristics reported by each student for each software application (Vi1, Vi2, Ed1, Ed2). The students with dyslexia (Sd1 and Sd2) described 17 and 9 characteristics respectively, and the other students (S1 and S2) described 6 and 1 characteristics respectively.

Table 3: Summary of negative characteristics/suggestions for improvement of the four software application's (Vi1, Vi2, Ed1, and Ed2) described by students with dyslexia (Sd1, and Sd2) and other students (S1, and S2).

No.	Negative Characteristic	S1	S2	Sd1	Sd2
1	Vi1: Colour Coded Variables	✓			
	Total Vi1	**1**	**0**	**0**	**0**
2	Vi2: Background colour			✓	✓
3	Vi2: Arrows startling			✓	
4	Vi2: Colour Blindness (disability specific)			✓	
5	Vi2: Speed up later iterations			✓	
6	Vi2: Substitution confusing				✓
	Total Vi2	**0**	**0**	**4**	**2**
7	Ed1: Text box too small			✓	✓
8	Ed1: Buttons' tool-tips	✓			
9	Ed1: Instruction labels	✓			
10	Ed1: Buttons very small			✓	
11	Ed1: Edit boxes small			✓	
12	Ed1: Initially time consuming			✓	
13	Ed1: Prone to typing error			✓	
14	Ed1: Text repeated confusing			✓	
15	Ed1: Writing different sizes			✓	
16	Ed1: Data entry labels				✓
17	Ed1: Hard to follow				✓
	Total Ed1	**2**	**0**	**7**	**3**
18	Ed2: Easy Array Creation		✓	✓	✓
19	Ed2: User training	✓		✓	
20	Ed2: Narratives - pop-up	✓			✓
21	Ed2: Graphics on buttons	✓			
22	Ed2: Drag frame pane			✓	
23	Ed2: Font Sizes inconsistent			✓	
24	Ed2: Frames diagram small			✓	
25	Ed2: Instruction Dialogue box font			✓	
26	Ed2: Position of frame buttons				✓
27	Ed2: Easy access to properties (lost from Ed1)				✓
	Total Ed2	**3**	**1**	**6**	**4**
	Overall Total	**6**	**1**	**17**	**9**

The following examples are representative of the type of comments made by the four students:

- **Substitution Confusing**: One student (Sd2) indicated that although the animation of variable name-value substitution made the process make 'much more sense', it was 'probably the most confusing part' of the second viewer application. They suggested that it needed to 'be split up a bit more', done 'slower', and 'broken down into smaller steps'.
- **Buttons' tool-tips**: One student (S1) suggested that the buttons in the first editor (Ed1) software could have 'a little tip that pops up [that explains] what it is'.
- **Easy Array Creation**: Three students (S2, Sd1, and Sd2) suggested providing a facility to help when 'setting up an array' that 'makes them [the array elements] all up for you' (currently the user has to create each element of the array individually). Student Sd2 commented that 'that would save a little bit of time'. Student Sd2 commented that 'it would be easier', and student Sd1 commented that it could 'get rid of the necessity to write the brackets, and that 'brackets are one of the most awkward things to put in'.
- **Easy Access to Instruction Properties**: One student (Sd2) commented that in version 2 of the editor 'something had been taken away from version 1', and that they thought it was the 'properties bar' which allowed direct access to change all of an instruction's properties.

3.5 Software's Contribution to Learning

All students indicated that the software would generally enhance student learning. One student (S1) indicated that they had not learnt anything from using the software because they already understood the concepts in the example animation. The other three students (S2, Sd1, and Sd2) indicated that their understanding of functions had improved as a result of using the software.

4 Conclusions

4.1 Issues Reported

Overall, the four students identified 15 difficulties with the computer programming learning and teaching process, and 24 positive and 27 negative software characteristics. Both dyslexic students mentioned difficulty keeping track of the current position in the code and keeping up with the lecturer, and indicated that the software was easier to follow than oral narratives because it slowed the pace and the ticks helped to maintain position (corroborating the findings of Dixon, 2004b).

Table 4 shows the total number of issues reported by each participant, along with their assessed level of experience. All students identified important issues across all three categories. However, the two dyslexic students consistently identified a larger number of issues than the other two students. Student Sd1 (the student from the previous work) identified the most issues, which may be due to their familiarity with the software. However the other dyslexic student (Sd2) identified a comparable number of issues.

Table 4: Experience level, and total number of issues reported by each participant (student).

Student	S1	S2	Sd1	Sd2
Experience Level	High	Medium	Medium	Medium
Teaching and Learning Difficulties	3	3	6	7
Positive Software Characteristics	7	7	15	13
Negative Software Characteristics	6	1	17	9

Table 5 shows a summary of which students reported issues regarded by the developer as both subtle (surprising to the developer, and not considered prior to feedback from others) and significant (relevant to the general student population and would directly influence future software design and/or teaching practice).

Table 5: Summary of issues regarded by developer as both subtle and significant to the general student population, with indication of which students reported them.

	No.	Positive Characteristic	S1	S2	Sd1	Sd2
Difficulties	1	Jumping out of sequence in code		✓		✓
	2	Keeping track of current position in code			✓	✓
	14	Object oriented programming is difficult because of jumping				✓
Positive Characteristics	5	Vi2: Yellow Highlight of Current Line	✓	✓	✓	✓
	7	Vi2: easier to see where	✓		✓	
	8	Vi2: easier to keep up			✓	✓
	6	Vi2: Transitional Animation		✓		✓
	9	Vi2: Substitution			✓	✓
	15	Vi2: Changed perception of substitution				✓
	10	Vi2: Larger Font			✓	✓
	18	Ed2: Help lecturer's understanding of students	✓	✓		
Negative Characteristics	6	Vi2: Substitution confusing				✓
	18	Ed2: Easy Array Creation		✓	✓	✓
	22	Ed2: Drag frame pane			✓	
	27	Ed2: Easy access to properties (lost from e1)				✓
Total			**3**	**5**	**8**	**12**

All students described at least one of these issues. However, the dyslexic students (Sd1 and Sd2) seemed to describe more issues to a greater degree of depth. For example, all students identified the 'current line highlighting' as particularly beneficial, yet the problems this related to (keeping track of current position when jumping out of sequence) were only deeply described as a generic problem by the dyslexic students. Also, student Sd2 described difficulty with object-oriented programming due to the increased amount of 'jumping' compared to conventional programming. There is evidence of this in a study where data showed students performing poorly in control structure questions, although the issue is not described in detail in the text (Wiedenbeck, Ramalingam, Sarasamma & Corritore, 1999).

In contrast, both non-dyslexic students (S1 and S2) made comments that may be interpreted as suggesting the software helps the lecture understand the student's view of code better. It is still not clear what the students meant, however, it could mean that the software would remind the lecturer of the individual steps involved in program execution, which is potentially very important to the learning and teaching process. Over time programmers (lecturers) stop thinking in terms of the actions of individual lines of code and start thinking in terms of blocks of code. If the lecturers then explain code in terms of overall function the students may not develop an understanding of the impact of each individual line, and may erroneously believe that individual lines don't do specific things, but only act as part of a group structure. However, the students' comments were brief, unclear, and went unnoticed during the evaluations. Their potential significance was noticed during the subsequent transcript analysis.

4.2 Factors Influencing Issues Reported

There are several factors that may influence the ability of participants to identify issues: working relationship with interviewer, age, gender, computer programming experience, and presence of dyslexia. All students had an established working relationship with the interviewer, so this can be regarded as static across all participants. The oldest student (Sd1) identified the most issues, which may suggest that older students are better able to articulate themselves. However, student Sd2 (who was the same age as students S1 and S2) identified the largest number of subtle yet significant issues, so age does not appear to be a strong factor. Students Sd1 and Sd2 were different genders and performed comparably. Student S1 had the highest level of computer programming experience and identified the lowest number of subtle-significant issues, which suggests that experience may inhibit an evaluators ability to identify key issues. This makes sense as the more experienced an evaluator is the more distant they are from the target user group (novice programmers).

The largest variation in both number of issues and number of subtle-significant issues, occurred between the dyslexic (Sd1, and Sd2) and non-dyslexic (S1, and S2) students. All students were able to say whether they liked the software or not. However, non-dyslexic students seemed to give a slightly down-played impression of learning and teaching difficulties, and the impact of small design modifications, whereas the dyslexic students seemed to be more explicit and detailed about their problems, and how/why features of the software helped or hindered them. The dyslexic students identified a larger number of issues and subtle-significant issues than the non-dyslexic students. In particular, student Sd2 (who was not involved in earlier evaluations) performed at the same level as the dyslexic student involved in the earlier work (Sd1). This supports the idea that dyslexic students are able to identify subtle aspects of human activity, which are relevant to (but not articulated by) the wider target-user population.

It is not clear why this is the case. Dyslexic students are used to considering their problems, so they may be less inclined to play their problems down, whereas non-dyslexic students indicate that 'it's only a little thing' when in fact the issue had quite an impact on usability or usefulness. There may be a relationship between dyslexia and being a novice. Dyslexic students find the process of learning computing programming more difficult than other students, which means their experience of being a novice may last longer. This may make them more familiar with the difficulties associated with being a novice learning computer programming concepts, and are therefore more sensitive to usability issues as well as good software characteristics.

4.3 Implications

The power of user-testing, and the importance of including evaluators drawn from the target user population is widely recognised, which suggests that dyslexic students should be included in evaluations of educational software, where they form part of the target user population, so that the software meets their needs. The present work suggests that there is also benefit to all students from including a slightly disproportionately higher number of dyslexic students in educational software evaluations, as a richer picture of the usefulness and impact of the software is likely to be elicited. However, it also indicates that non-dyslexic students also contribute significantly to this picture, and therefore should not be excluded from evaluations. It is not clear whether this is limited exclusively to dyslexia and this specific software application, or is equally applicable to educational software in general, other application domains, and/or other forms of 'disability'.

In usability testing the developer is often presented with conflicting comments, or simply too many issues to act upon. Therefore a decision has to be made on which issues to prioritise. Although good interviewing skills encourage the asking of follow-on questions, the interviewer has to recognise the issues as they arise in order to be able to follow them up. The dyslexic students gave more detailed information regarding the context of issues they described, which at times made supplementary questions unnecessary, and at other times highlighted these issues as important and increased the likelihood of the interviewer asking supplementary questions.

4.4 Limitations and Further Work

It is difficult to be certain that the protocol was followed exactly for each participant, especially given that the interviewer was aware of the presence of dyslexia in each student prior to the evaluations. The study would have been strengthened if the interviewer did not know who was dyslexic at the start, even if it came out during interview.

There are two key parts of the analysis that depend upon the subjective interpretation of the author. Firstly, the relevance of the identified issues to the wider student community were judged on the basis of the author's experience of lecturing for ten years. These issues need to be confirmed with stage one students. Secondly, the experience of the participants, which was judged on the basis of the students' own perceptions of their ability. In future work a more objective assessment of this would be preferable, as well as the inclusion of first year students. The familiarity of each student with the specific concepts in the example used may have affected their responses, it would be interesting to see how the same students respond to an object oriented programming example (which would be less familiar to them, and generally more complex and difficult to understand). Also, in order to consider the combined affect of experience and dyslexia the evaluation would need to be conducted with a very experienced dyslexic student. Possibly the most important further work would involve investigating whether this phenomenon extends to other educational software, other application domains, and other disabilities.

References

Biggs, J. (1999). Teaching for Quality Learning in University. Open University Press.

Bryson, J. (1997). Breaking through the A Level Effect: a first-year tutorial in student self-reflection. *Journal of Geography in Higher Education*, 21 (2), 163-169.

Dix A, Finlay J, Abowd G, & Beale R (2004) Human-Computer Interaction (3rd edition). Pearson Education Limited, London.

Dixon, M. (2004a). Code-memory diagram animation software tool: towards on-line use. Proceedings of the IASTED International Conference on Web-based education, Innsbruck, Austria, 14-16th February, 601-603, ACTA Press.

Dixon, M. (2004b). Disability as a Vehicle for Identifying Hidden Aspects of Human Activity: Inclusive Design and Dyslexia in Educational Software Development. Revised selected papers from the 8th ERCIM workshop on User Interfaces for All, Vienna, Austria, 28-29th June, 254-261, Springer-Verlag, Berlin.

Dixon, M. (2004c). Generic Code-Memory Diagram Animation Creator: An Aid for Effective Computer-Programming Teaching. Proceedings of the 7th IASTED International Conference on Computers and Advanced Technology in Education, Kauai Island, USA, 16-18th August, 179-184. ACTA Press.

Nielsen, J. (1993). Usability Engineering. Academic Press Inc, London.

Nielsen, J. (1994). Usability Inspection Methods. John Wiley, New York.

Oppenheim, A. N. (1992). Questionnaire design, interviewing and attitude measurement. Pinter, London.

Shneiderman, B. (1998). Designing the user interface: Strategies for Effective Human-Computer Interaction (3rd edition). Addison-Wesley Publishing Company Inc.

Wiedenbeck, S., Ramalingam, V., Sarasamma, S., & Corritore, C. (1999) A comparison of the comprehension of object-oriented and procedural programs by novice programmers. *Interacting with Computers*, 11 (3), 255-282.