Pre-publication draft of paper accepted for publication in the Proceedings of the International Conference on Computers in Education (ICCE'99) Vol. 2 pp 385-392 IOS Press

Evaluation of Low-Level Program Visualisation for Teaching Novice C Programmers.

Philip A. Smith* and Geoffrey I. Webb**

* *School of Information Technology and Mathematical Science*

University of Ballarat, Victoria, Australia

p.smith@ballarat.edu.au

** *School of Computing and Mathematics*

Deakin University, Victoria, Australia

webb@deakin.edu.au

While several program visualisation tools aimed at novice programmers have been developed over the past decade there is little empirical evidence showing that novices actually benefit from their use [3]. Bradman [7] is a low-level program visualisation tool. We present an experiment that tests the efficacy of Bradman in assisting novice programmers learn programming concepts. We show that students with access to this low-level program visualisation tool achieved greater understanding of some programming concepts than those without access.

keywords: **Program Visualisation, Empirical Evaluation**

1 Introduction

Learning to program can be a difficult and frustrating process. Novice programmers must learn concepts and skills that often bear little relation to their past experiences. Hence the knowledge structures, onto which they must assimilate the information they are learning, are often inappropriate for the purpose.

The current methods of teaching programming include lecturing, and laboratory sessions in which the students reinforce what they have learned in lectures by developing small programs of their own. A further way of assisting students is to provide computerised aids that are designed specifically for the novice programmer. While much research has gone into the development of such tools, their use is not widespread [3]. One of the possible reasons for this is that little empirical data has been obtained that show the benefit of such machines [4].

Program Visualisation tools are systems that make visible aspects of program execution that are usually hidden from the user. As a result, they are potentially of great benefit to novice programmers who can have trouble conceptualising the internal mechanics of program execution. Program visualisation tools have been developed that provide graphical representations of high abstraction. However, [3] has pointed out that novice programmers can have trouble extracting information implicit in graphical representations. As a result, this study concentrates on low-level program visualisation tools that provide simple representations without high-level abstractions.

To this end, a low-level program visualisation system (called Bradman) has been developed to provide a test-bed for evaluation of the effect of such tools on novice programmers. Bradman is an interpreter that makes visible aspects of the programming process that are normally hidden from the user. For this reason, we call it a "Glass-box Interpreter". Bradman contains many of the features of other low-level program visualisation tools, such as program animators, but also has several novel features designed specifically to be of assistance to novice programmers. The most significant of these features is a representation of the variables that is designed to reinforce a model of program execution as an active process that achieves its results through the change of program states, brought about by the execution of program statements [7].

Bradman was developed to redress the dearth of empirical evaluation of such tools and their effect on novice programmers. The following section describes an experiment that evaluates whether access to a glass-box interpreter assists students to develop a better understanding of program execution.

2 Experiment

Volunteers were sought from Deakin University's introductory programming course that teaches programming concepts using C. The experiment ran over a three week period in first semester. Subjects were required to attend three two-hour laboratory sessions (one per week). They were told that they would be testing ways to improve programming environments for novice programmers. A payment of thirty Australian dollars was made to the participants who completed all three sessions. Twenty-six people volunteered to participate. They filled out an appropriate consent form before commencing the experiment.

These sessions required the students to perform desk-checking (attempting to diagnose a program’s function by examination of the program code) of pre-written programs. The first session was an introduction to the format that the subsequent sessions would take. All experimental data were collected in the second and third sessions. The students were required to perform the following tasks.

- Desk-check a program prepared for them that was designed to illustrate and reinforce a concept that was introduced to them in lectures.
- Calculate the outputs of this program and answer multiple choice questions regarding these outputs.
- When finished, compile and run the program and compare the output to their calculations.
- Attempt to discover the reason for any mistakes that they made. They were allowed to use textbooks, ask questions of the tutors and analyse the program using the symbolic debugger, gdb. After the students had had some time to analyse the program code, the instructors gave a demonstration on the whiteboard of how the program achieved its results.

- When they finished working on their program and had watched a demonstration on the whiteboard they were given a similar program to desk-check and with regard to which to answer multiple choice questions.

Thus the experimental period involved four tests - one at the beginning and one at the end of both sessions 2 and 3. For convenience sake these tests were numbered 1, 2, 3 and 4 in the order in which they were performed. Hence Test1 was conducted at the beginning of session 2 before the participants had received either treatment. Test4 was conducted at the end of session 3 after all experimental interventions had been performed. Test2, conducted immediately after the first intervention, and test3, conducted a week later immediately before the second intervention allow us to map student progress through the experiment.

The students' proficiency was judged by the number of correct responses to the multiple-choice questions. The twenty-six volunteers were split into two groups of thirteen. This was done based on the order in which they volunteered to participate - the first volunteer was put into the test group, the second into the control group, the third into the test group and so on. However, two subjects from the control group withdrew after week two, leaving unequal groups - one of thirteen and the another of eleven. Experimental intervention related to the normal laboratory activities that were conducted between the tests conducted at the beginning and end of each session. The test group had access to Bradman during this phase in which they attempted to gain an understanding of the program. The control group did not have access to Bradman during this phase.

2. 1 Session 1

Session 1 was an introductory session in which the students were introduced to the format that would be used in sessions 2 and 3. The students were required to desk-check a very simple program and answer multiple choice questions regarding its output. During the treatment period the test group was instructed on the use of Bradman - how it was invoked and how it could be used. At the end of the session the students were required to desk-check another program and answer multiple choice questions. This session was intended solely to familiarise the students with the format of the following two sessions. The results were not collated or analysed. The control group underwent the same process as the test group except that they were not given exposure to Bradman. Although this first session might be seen as having an influence on the overall result, the effect is probably minimal because the students were given simple exercises. The comparative performance of the two groups through the experiment seems to support this. Even if this is not accepted, and the exposure to the low-level program visualisation tool in session 1 is viewed as a significant experimental intervention, it in no way devalues the final result which should then be viewed as the result of three rather than two experimental evaluations.

2. 2 Session 2 - The Scope of Variables

The scope of variables is a concept that students can have trouble. Students have to understand pieces of code in which a name refers to a global object in one statement while in another statement the same name refers to a different, local, object. They need to understand that a local variable in a calling function cannot be accessed (except through the use of pointers with which they were not as yet familiar) during the life of the called function. Hence, desk-checking such a program is not a trivial exercise and presents difficult challenges to the novice programmer.

The program used in session 2 was prepared by the authors of the introductory programming unit as part of their course. It was incorporated (with their permission) into the experiment and was used as Test1. This program was designed to illustrate many of the concepts of scoping including how different variables can have the same name, how variables have a certain "life" or scope during which they are valid and how they lose their validity outside of this scope.

The program was developed by professional programming instructors to teach novice programmers about scoping. It consisted of four functions including the main function. The functions did not perform a meaningful task. They simply assigned and reassigned values to variables outputting them at various stages. None of the functions or variables had meaningful names. While the students were desk-checking this program they were required to answer five multiple choice questions. The students were instructed to circle the alternative that best answered the question.

An additional program was developed specifically for the study and used as Test2. It was similar to Test1 but had certain values changed so that students would not answer the multiple-choice questions from their memory of Test1. Again the students were asked to desk-check the program and again answer five multiple choice questions regarding its output.

2. 3 Intervening Period Between Sessions 2 and 3

There was an interval of one week between the performance of Test2 and Test3. This break was significant because it gave an opportunity to judge whether Bradman indeed provided the student interval an adequate framework onto which to attach new information. During this period the students attended lectures in which the concepts used in session 3 were taught. If the glass-box interpreter was efficacious in enhancing a student's mental model of program execution then one would expect the test group to perform better than the control group on Test2. One would also expect the test group to show an improvement on the initial test program in session 3.

2. 4 Session 3 - Parameter Passing

In session 3 the subjects were called upon to analyse programs that involved pass-by-reference parameter passing. The instructors of the introductory programming unit thought this concept to be of such difficulty and importance that they gave the students the following warning:

> "The concept of `pass-by-reference' (pointers) is probably the biggest hurdle you'll come across in C programming. When you absorb and UNDERSTAND this topic, you will have overcome the biggest learning curve in C. The rest of C programming will seem somewhat easier." [1].

These sentiments have been echoed by other researchers [2]. Pass-by-reference parameter passing in C involves the use of pointers that were a relatively new concept for the students being tested. They needed to develop a model of how, through the use of the pointers, the values of variables in the calling function would change rather than those in the called function.

Test3 consisted of two programs prepared by the instructors of the introductory unit, for use in the regular laboratory session. The first program consisted of an incorrect version of a swap program in which the parameters passed to the function swap were pass-by-value parameters. The second is the correct version in which pass-by-reference parameter passing is used. Test4 comprised three additional programs developed specifically for the study. They differed from the Test3 programs only in the arguments that were passed to the function swap. The students were asked to answer two questions at the end of each program. The questions were the same for all five programs. These questions revolved around the values of the parameters after the swap had been completed but before the function returned and the values of the arguments of the call to swap after the function had returned. Unlike the programs in session 2, the function and variable names were more meaningful, reflecting their tasks.

2. 5 Results and analysis

The multiple-choice questions were collected and marked. The numbers of correct and incorrect responses for both groups are summarised in Tables 1 and 2. It must be remembered that the students had a choice of four possible responses from which to choose. All of the incorrect responses were grouped into one tally. Hence, for every test the distributions represent better than random performance by the participants. The results seem to show an improvement on the part of the test group. As expected the number of correct responses for both groups increased after the laboratory session but the improvement of the test group was greater. The test group performed less well on Test1 (equal correct but more incorrect) than the control group but performed better on the Test2 (more correct and fewer incorrect).

However, it could be misleading to compare the groups in this manner because each individual was required to answer five questions. Thus a change in performance by just one subject could affect as many as five question responses. In order to compare the two groups the individuals were given a ranking according to the number of correct response they made. Rankings were made for both Test1 and Test2. Mann-Whitney U-tests comparing the two groups were performed on both of these rankings. The results were as follows:

Test1	$z = 0.70$	$p = 0.25$
Test2	$z = 0.96$	$p = 0.17$

Hence, although the results from Table 1 appeared to show an improvement in favour of the test group, statistical analysis of the rankings did not show that a situation was reached in which the test group performed significantly better than the control group.

The results for session 3 were summarised in Table 2. Again the individuals were given a ranking according to how many correct responses they made. Rankings were made for both Test3 and the Test4 results and Mann-Whitney U-tests were performed on both of the rankings.

The results of these tests were as follows:

Test3	$z = 1.42$	$p = 0.08$
Test4	$z = 2.03$	$p = 0.02$

The results show a definite bias towards the test group. In test1 the test group performed slightly worse than the control group, although this difference was not statistically significant and hence does not necessarily indicate that the test group were less able than the control group prior to the experimental intervention. Immediately after the first experimental intervention the test group now outperformed the control group, although again the difference was not statistically significant preventing us from concluding that this was not a chance outcome. However, we feel that the result is sufficiently interesting to warrant further research in experiments with more students. In Test4 the test group performed significantly better (at the 0.05 level) than the control group.

Test1				
	Test Group		Control Group	
	Correct	Incorrect	Correct	Incorrect
Q1	4	9	5	6
Q2	7	6	4	7
Q3	6	7	8	3
Q4	4	9	4	7
Q5	5	8	5	6
Totals	26	39	26	29

Test 2				
	Test Group		Control Group	
	Correct	Incorrect	Correct	Incorrect
Q1	13	0	9	2
Q2	11	2	8	3
Q3	9	4	7	4
Q4	9	4	7	4
Q5	5	8	3	8
Totals	47	18	34	21

Table 1: Summary of Session 2 Results

As mentioned earlier, two students withdrew from the experiment after session 2 was conducted. As a result, their results could not be used and the groups became unequal with thirteen in the test group and eleven in the control group. However, there is no reason to believe that this would have confounded the results. The results show that in test1 the control group, without the two subjects who withdrew, performed slightly better than the test group although this difference was not significant. Subsequent results show a clear improvement of the test group's ability to perform the given tasks as compared to that of the control group. It is not clear how these results could be interpreted as indicative of a confound introduced through the subjects' withdrawal.

Test 3					
		Test Group		**Control Group**	
		Correct	**Incorrect**	**Correct**	**Incorrect**
Program 1	**Q1**	7	6	6	5
	Q2	7	6	4	7
Program 2	**Q1**	8	5	3	8
	Q2	10	3	6	5
	Totals	32	20	19	25
Test 4					
		Test Group		**Control Group**	
		Correct	**Incorrect**	**Correct**	**Incorrect**
Program 3	**Q1**	11	2	7	4
	Q2	8	5	3	8
Program 4	**Q1**	10	3	7	4
	Q2	10	3	4	7
Program 5	**Q1**	10	3	5	6
	Q2	9	4	7	4
	Totals	58	20	33	33

Table 2: Summary of Session 3 Results

3 General Observation and Discussion

Bradman was enthusiastically accepted by the students many of whom asked for it to be made generally available. More than one Bradman user told the experimenter that they had no idea about how programs worked until they used Bradman. This response is something that we expected, however, from people who were perhaps unsure about how well they were doing in their course. We expected them to appreciate the visibility and simplicity of Bradman after struggling with tools like gdb that are designed for expert programmers.

One interesting observation occurred during session 3 that was a session in which an assignment was due. Several of the students used Bradman, not to develop their assignments, but simply to watch their own programs execute. There was no need for them to do this because they were not looking for errors - they knew that their programs gave correct results. This is consistent with an observation of [5] who noted similar behaviour even in expert programmers and engineers. People seem to enjoy watching how their creations work.

Free-form responses elicited from subjects in the test group revealed that many of them liked watching the way that the variables changed value as the program executed. They did not mention the other part of the program state change - the change of execution point. One student said that the marker in the code window should be changed to highlighting because it was too difficult to see.

5 Conclusions

Programming is a difficult skill to master and novices need to develop appropriate knowledge structures to enable them to cope with it. Program visualisation tools have been used to teach students in classroom and laboratory situations. However, there is little empirical evidence as to their efficacy and their use is not widespread.

The experiment that we conducted provides evidence that low-level program visualisation tools, such as glass-box interpreters, can be beneficial in teaching novice programmers. It provided concrete empirical evidence that such a tool can provide assistance in learning new programming concepts. It also indicated that the use of such a tool could enable students to assimilate new information more effectively. This suggests that our tool presents a conceptual model that provides an appropriate framework onto which learners can assimilate new information. Bradman provides information in a dynamic manner while being very simple to use and makes visible different views of the program execution that are normally hidden. We believe that these features are important in enabling students to better visualise how a program works as it executes. We further believe that the benefits shown in comprehension will be mirrored in benefits to program development and program debugging and this will be the basis of further research.

References

[1] Dew RA, Newlands DA. Basic Programming Concepts. Deakin University study guide, (1996).
[2] Fleury A. Parameter Passing: The Rules the Students Construct. Communications of the ACM, 283-286, (1991).
[3] Mulholland PA. A framework for describing and evaluating software visualisation systems: A case study in PROLOG. Phd Thesis, Knowledge Media Institute, Open University, (1995).
[4] Price BA, Small IS, Baecker RM. A Principled Taxonomy of Software Visualisation. Journal of Visual Languages and Computing, Vo l 4, Num 3, 211-266, (1993).
[5] Ross RJ. Experience with the DYNAMOD Program Animator. Special Interest Group on Computer Science Education, Vol 23, Num 1, 35-42, (1991).
[6] Smith PA, Webb GI. Transparency Debugging with Explanations for Novice Programmers, Proceedings of the 2nd Workshop on Automated and Algorithmic Debugging, St Malo, France, 1995.
[7] Smith PA, Webb GI. Overview of a low-level Program Visualisation Tool for Novice C Programmers. Proceedings of ICCE'98, 213-216, 1998.