Improving learning outcomes for first year introductory programming students

Author
Venema, Sven, Rock, Andrew

Published
2014

Conference Title
Proceedings of the 17th International First Year in Higher Education Conference

Copyright Statement
Copyright remains with the authors 2014. The attached file is posted here with permission of the copyright owners for your personal use only. No further distribution permitted. For information about this conference please refer to the publisher’s website or contact the authors.

Downloaded from
http://hdl.handle.net/10072/68462

Link to published version
Improving learning outcomes for first year introductory programming students

Sven Venema
School of Information and Communication Technology, Griffith University, Brisbane, Australia

Andrew Rock
School of Information and Communication Technology, Griffith University, Brisbane, Australia

Abstract

The first year transition to university can be particularly difficult for students in technical degree programmes such as Information Technology (IT). These programmes are generally acknowledged to have high attrition rates. It has been suggested that poor understanding of threshold concepts in introductory computer programming courses may be partly responsible for this attrition. The current project attempts to improve student learning outcomes for the computer assignment threshold concept. To achieve this outcome, an innovative visualisation application was developed and incorporated into the curriculum of a first semester first year introductory programming course. Substantially improved student learning outcomes were observed after its successful deployment. Subsequent tracking indicates the innovation continues to successfully scaffold student learning of the computer assignment threshold concept.

Introduction

The first year transition to university can be particularly difficult for students in technical degree programmes such as Information Technology (IT) which are generally acknowledged to have high attrition rates, around 30-50% (Denning & McGettrick, 2005). It has been suggested by Ma, Ferguson, Roper, and Wood (2007) that these attrition rates are most likely due to poor performance in first year introductory programming courses.

The technical nature of these introductory programming courses can lead to distinct difficulties in understanding. One established way of considering these sorts of difficulties is Perkins' (1999) notion of troublesome knowledge. Meyer and Land (2005) argue that the defining characteristics of this knowledge are that it is counter-intuitive, conceptually difficult, and alien. They also suggest that this type of troublesome knowledge forms a gateway through which a more advanced way of thinking about a topic area can be developed. These gateway concepts are now widely referred to as threshold concepts (Meyer & Land, 2005).

Since the early 2000's educators have become increasingly concerned regarding the level of competence of students undertaking introductory computer programming courses that are typically undertaken in the first year of an IT degree programme. In 2001, the McCracken group (McCracken et al., 2001) reported on a multi-national, multi-institutional study of assessment of computer programming skills of first year computer science students. The
study was carried out at 8 higher education institutions in the United States of America, Australia, Israel, Wales, England, and Poland. The project aimed to assess the computer programming skills of undergraduate students after completing one or two courses in computer programming. The outcomes of this investigation indicated that most students performed well below expectations on problems that "students in any type of Computer Science programme should be able to solve" (McCracken et al., 2001, p. 128). One striking result from this investigation was an average score of 22.89 for a test that was scored out of 110.

Lister et al. (2004) led another group of educators who examined first year introductory programming courses from 12 educational institutions including Australia, England, the United States of America, Finland, New Zealand, Sweden, Denmark, and Wales. They found that many of the students in their study were weak at analysing and predicting the outcome of short Java computer code fragments. They suggested that rather than being weak at problem solving, these students "have a fragile grasp of both basic programming principles and the ability to systematically carry out routine programming tasks, such as tracing (or “desk checking”) through code." (Lister et al., 2004)

In 2006, Dehnadi suggested that another possible explanation for poor performance in first year programming courses could be related to lack of facility with the computer assignment threshold concept (Dehnadi, 2006). An example of a computer assignment statement written in the Java programming language as used by Dehnadi is given in (1).

\[
\begin{align*}
(1) & \quad \text{int } a = 10; \\
& \quad \text{int } b = 20; \\
& \quad a = b;
\end{align*}
\]

Given (1), \(a = b\) (read as \(a\) is assigned \(b\)), so \(a\) now has the value 20, and \(b\) would retain its value of 20. Dehnadi subsequently developed and made available a set of multiple choice test questions based on computer assignment and more complex questions involving assignment and sequencing of operations, see (2).

\[
\begin{align*}
(2) & \quad \text{int } a = 5; \\
& \quad \text{int } b = 3; \\
& \quad \text{int } c = 7; \\
& \quad a = c; \\
& \quad b = a; \\
& \quad c = b;
\end{align*}
\]

Given (2), the sequence of assignments is important; \(a = c\) (\(a\) is now 7, \(c\) remains 7); \(b = a\) (\(a\) remains 7, \(b\) is now 7); \(c = b\) (\(c\) is now 7, \(b\) remains 7).

In 2008, Bornat et al. (Bornat, Dehnadi, & Simon, 2008) investigated whether this threshold concept could be used as a predictor for success in first year introductory programming courses. After six experiments involving more than 500 students at six institutions across three countries they determined that the Dehnadi test questions were not predictive of successful performance in first year introductory programming courses. However, the types of questions on the test do provide a simple way of determining whether students understand assignment and sequencing concepts.
Improving learning outcomes for first year introductory programming students. Refereed paper.

Ford and Venema (2010) used the Dehnadi test questions at an Australia metropolitan university to determine whether students who have passed a first year introductory programming course (P1) had achieved an understanding of the computer assignment threshold concept. The Dehnadi test proved a useful tool in identifying that many students who had passed P1 could not consistently answer correctly questions related to computer assignment. In fact, despite passing P1, only 49.55% of the students in that study correctly answered 10 or more out of 12 questions on the test. This poor performance prompted an intervention in the P1 course in subsequent offerings leading to improved learning outcomes for students.

The remainder of this paper is organised as follows. First the details of a programming course (P1) are discussed and a threshold concept is identified. Next an intervention is described that can be used to successfully target the threshold concept and improve associated learning outcomes. This is followed by an outline of the research methodology and a discussion of the findings. Final remarks conclude the paper and indicate how this approach could be used in a wider context.

The Programming Course (P1)

The first year, first semester introductory programming course (P1) has undergone many changes over the years. Currently, the course uses LEGO® MINDSTORMS® NXT robots to allow students to solve physical real-world problems. In addition, only a small subset of the Java programming language is introduced to the students in this first introductory programming course. The subset language uses only valid Java syntax and is called MaSH (Making Stuff Happen). MaSH includes everything except for user defined classes and allows students to remove unnecessary complexity until it is required. This has the effect of reducing cognitive load through the reduction of irrelevant information. MaSH allows the beginning student to gradually introduce complexity as needed and to focus only on the requirements of the problem without becoming bogged down by the complexity of the language.

The course materials include a special compiler mashc that has been developed in-house. The compiler converts the MaSH source code into its equivalent legal Java source code so that it can be executed by the user, on any system that can run Java. MaSH also presents limited versions of other Java Application Programming Interfaces (APIs), making it easier to find methods to use in various programming environments, such as a PC or a robot. This has the effect of further reducing the amount of detail that a student faces at any given time, consequently reducing the cognitive load. Environments are defined by substitution rules that may be developed by any instructor which allows for further technologies targeted at improving learning to be added into the student workflow as required. Students taking the course are required to attend a 2 hour lecture, a 1 hour workshop, and a 2 hour laboratory session per week. The assessment of the course has moved away from the traditional assignment and final examination model and focuses instead on ten weekly 5% laboratory exercises totalling 40%, five in-lecture 5% multiple choice quizzes totalling 25%, and a final 35% capstone project incorporating the key learning objectives in the course. P1 covers programming concepts, problem solving and design, implementation using a subset of the Java programming language, and using programming tools and environments (including MaSH). Programming examples are discussed and worked through in lectures and students apply the concepts in the weekly laboratory sessions as well as in a weekly one hour group workshop.
An example of the type of program worked through in the lectures and workshops (this example from week 4 of the 13 week course) is given in (3).

(3) import console;
    print("How much is the loan for ($)? ");
    double amount = readDouble();
    print("What is the annual interest rate (%)? ");
    double annualRate = readDouble();
    print("how many months? ");
    int months = readInt();
    double rate = annualRate/(12*100);
    double repayment = amount*rate/(1.0-pow((1.0+rate),-months));
    double totalInterest = months*repayment-amount;
    println("Monthly repayment = $"+repayment);
    println("Total interest = $"+totalInterest);

In (3) note the MaSH subset of standard Java statements. For example: print("How much is the loan for (?$)?"); simplifies the standard print statement System.out.print("How much is the loan for (?$)?");. Reading an integer from the keyboard, as in int months = readInt() reduces the standard way of reading integers in Java from multiple lines down to a single statement. By week 7 of the course, programs begin to require encapsulation with methods, but not classes.

The intervention

One of the difficulties that programmers face, particularly those just beginning, is that it is not possible to see inside the machine. When a sequence of statements needs to be traced through, as in the Dehnadi questions (Dehnadi, 2006), the student has to step through the sequence of assignments and keep track of the values associated with each variable in order to determine the outcome. This is usually done in the student’s working memory and may incur significant cognitive load even for simple sequences of assignments. Recall that Lister (Lister et al., 2004) identified tracing through a program as a key weakness of beginning programming students. We posit that this weakness is in part due to the cognitive load requirements of this activity and that student learning outcomes may improve by providing assistance targeted at reducing this cognitive load.

Enabling the beginning student to visualise the internal state of the machine, that is, keep track of the variables and their values as well as the current statement to be executed next, would reduce the demand on working memory and thereby reduce the overall cognitive load. By seeing the statements and their impact on the internal state of the machine, the learner can focus on the problem at hand rather than the newly learnt rules for solving the problem. Sweller (1999) refers to this source of cognitive load as problem solving load. A second source of cognitive load identified by Sweller (1999) is referred to as split attention. This type of cognitive load occurs when tracing through a program by having to remember the current values of all variables whilst focusing on the current statement which is changing those values.

To reduce both the problem solving and split attention cognitive load whilst tracing through a computer program, the PC programming environment was extended in-house to provide a visual trace and debugging window. The application displays, in real time, both the contents
of the variables and how those contents change for each statement in a program as it is executed by the student. As part of the intervention, the visualisation application was incorporated into the P1 curriculum as a tool for students to get a better sense of the internal workings of their programs, particularly as it relates to sequences of assignments. Students can use the visualisation software as they execute their program via a single alternate command instead of the standard Java compile command. Figure 1 illustrates a simple program in this environment.

![MaSH Debugger Example](example.png)

**Figure 1 – Visualisation of a sequence of assignment statements.**

In Figure 1, the window on the left shows the program source code (the blue arrow indicates the line of code that is just about to be executed). The source code consists of an `import` statement to gather all the required resources required by the program to compile and execute, two variable assignments, and a single statement assigning the contents of variable `b` to variable `a`. The window on the right shows the variables currently residing in memory and their contents. Programs with methods and therefore different scopes show local variables and parameters being added and removed from the stack, another threshold concept.

Note that the blue arrow is before the `a = b` assignment statement indicating that this statement has yet to execute. Hence, the values of 3 and 5 for variables `a` and `b`, respectively. Students can step through the program a single statement at a time at various speeds from slow to fast as well as run the program normally. The right window is updated each step and always reflects the current internal state of the machine. This allows the learner to access all relevant information at the same time, reducing cognitive load.

In addition to developing the visualisation software, in Semester 2, 2010, all course material related to assignment was reviewed and examples the visualisation software was integrated into the content. Walkthroughs and additional exercises were also developed for use in workshops and students were trained in the use of the software. However, use of the software was not made mandatory in Semester 2, 2010 due to it being used for the first time that semester. After a successful trial in Semester 1, 2010, use of the visualisation software became mandatory in Semester 1, 2011.
In order to verify whether the intervention led to an improvement in student learning outcomes related to computer assignment, the two questions detailed above were embedded in the assessments from Semester 2, 2010 onwards. The next section details the methodology used to verify the effectiveness of the innovation.

Methodology

Participants

Students enrolled at one of the metropolitan campuses of the university where P1 was taught were given two questions related to assignment (see the section detailing the questions below) during the in-lecture quizzes that formed part of the normal assessment regime for the course. The experiment was conducted over 5 semesters from second semester 2010 through second semester 2012. The number of participants per question per semester of the experiment is shown in Table 1.

<table>
<thead>
<tr>
<th></th>
<th>Q1</th>
<th>Q2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semester 2, 2010</td>
<td>43</td>
<td>46</td>
</tr>
<tr>
<td>Semester 1, 2011</td>
<td>190</td>
<td>150</td>
</tr>
<tr>
<td>Semester 2, 2011</td>
<td>61</td>
<td>46</td>
</tr>
<tr>
<td>Semester 1, 2012</td>
<td>155</td>
<td>111</td>
</tr>
<tr>
<td>Semester 2, 2012</td>
<td>55</td>
<td>42</td>
</tr>
</tbody>
</table>

Table 1: The number of participants per question per semester of the experiment.

Procedure

Each quiz was given under examination conditions in a 30 minute time slot. Question 1 was given immediately after the concept had been introduced in Quiz 2 held in week 5 of semester. Question 2 was given towards the end of semester in week 12 as part of the final Quiz number 5. The second author taught the P1 course over all 5 semesters and administered the quizzes. Marking of the quizzes was carried out electronically. Each of the two questions is outlined in the next section.

The Questions

The following sections show the two questions given as part of the experiment.

Question 1

Figure 2 shows Questions 2. This question was given immediately after the concept of assignment was introduced in week 5.
Question 1: After these statements, what will be the contents of the variables?

```cpp
int a = 1;
int b = 2;
a = b;
b = a;
```

A) a is 1, b is 1  
B) a is 1, b is 2  
C) a is 2, b is 1  
D) a is 2, b is 2  
E) Nothing, because there will be an error.

Figure 2 – Question 1

Question 2

Figure 3 shows Question 2. This question was given in week 12 as part of the final Quiz. This question is essentially the same as question 1 but adds the concept of variable scope.

```cpp
void main() {
    int i = 1;
    int j = 2;
    i = j;
    j = i;
    // HERE
}
```

A) i is 1, j is 1  
B) i is 1, j is 2  
C) i is 2, j is 1  
D) i is 2, j is 2  
E) Nothing, because there will be an error.

Figure 3 – Question 2

Results

Students in each of the 5 offerings of P1 were marked either correct or incorrect for each of the 2 questions. The percentage of students who correctly answered each question per semester is shown in Table 2.
<table>
<thead>
<tr>
<th></th>
<th>Q1 % correct</th>
<th>Q2 % correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semester 2, 2010</td>
<td>21</td>
<td>35</td>
</tr>
<tr>
<td>Semester 1, 2011</td>
<td>28</td>
<td>71</td>
</tr>
<tr>
<td>Semester 2, 2011</td>
<td>48</td>
<td>74</td>
</tr>
<tr>
<td>Semester 1, 2012</td>
<td>45</td>
<td>77</td>
</tr>
<tr>
<td>Semester 2, 2012</td>
<td>40</td>
<td>67</td>
</tr>
</tbody>
</table>

Table 2: percentage of students who correctly answered each question per semester.

**Discussion**

Recall that, prior to the intervention, 49.55% of P1 students could not consistently answer the above questions correctly. Table 2 shows that in Semester 2, 2010, when the visualisation software had been introduced in a non-mandatory trial capacity, initially only 21% of students could answer Question 1 correctly. Also recall that Question 1 was given immediately after the concept of assignment had been introduced. The number of correct responses improved to 35% for Question 2 by week 12. Whilst there was a small improvement, it was noted that uptake of the new visualisation software was low and that this could reduce the effectiveness of the intervention.

To improve its uptake, use of the visualisation software was made mandatory and further integrated into the laboratory exercises, workshop activities, and lectures in Semester 1, 2011. After this change initially only 28% of students in Semester 1, 2011 could answer Question 1 correctly. This is a slight improvement over the previous offering and could be due to early adoption of the visualisation software.

However, the number of students who could answer Question 2 correctly in week 12 jumped markedly to 71%. This is a strong indicator that the curriculum enhancements, in particular the mandatory use of the visualisation software have a positive effect on student learning outcomes. From Semester 2, 2011 onwards, almost half the students could answer Question 1 correctly in Week 5. It is likely that the improvement for Question 1 is due to the early use of the visualisation software in lectures, workshops, and laboratories when first presenting the concept. Since Semester 1, 2011, the percentage of correct responses for Question 2 in week 12 has stayed at around 70% and is further indication that the intervention has improved the learning outcomes related to computer assignment for the majority of P1 students.

In this case, the problem faced by students is directly related to the underlying technology with which they interact. Therefore it is legitimate to consider adjusting the technology as part of the educational design.

**Conclusion**

Today's Information Technology educators are under increasing pressure to improve learning outcomes for first year students in introductory programming courses. We present an innovative visualisation tool that assists students in reducing the cognitive load associated with tracing through a computer program. Use of this tool has been demonstrated to improve learning outcomes of a threshold concept related to computer assignment for students undertaking an introductory programming course. We believe that this tool, and its associated reduction of student cognitive load, could also be used successfully to improve student learning outcomes in other programming related activities.
The general principle espoused in this paper is the reduction of cognitive load using an appropriate technological intervention. Despite the success of the intervention discussed in this paper, cognitive load is still an issue in many areas of technology education and further work in adjusting the technology as part of the educational design is recommended.

References


