INTRODUCING INTELLIGENT EXERCISES TO SUPPORT WEB APPLICATION PROGRAMMING STUDENTS

Joe Appleton
Southampton Solent University
United Kingdom

Abstract

Using computer based tutoring software to assist students to learn traditional programming languages has been widely explored. With the widespread growth of the Internet, universities are teaching more web specific programming languages. Computer based tutoring systems provide limited support for such languages. This paper presents a prototype system that aims to support students in learning the web language JavaScript. The potential of this system is explored by using a mixed methods survey design completed by 40 students and 10 staff members. Results show that our system can aid students in learning to program the web-based language JavaScript.

Keywords: computer-based learning environments, intelligent tutoring systems, JavaScript, introductory programming course

Introduction

Within the computing department at Southampton Solent University, we share a widespread and yet to be solved problem: the challenge of teaching first year students to program.

While programming is a fundamental topic taught in university computing courses, most agree it is a complex skill to master. It is widely accepted that it takes an estimated ten years of experience to progress from a beginner to an expert programmer (Winslow, 1996). It is therefore imperative that the start of a student's programming journey is as smooth as possible. Evidence would suggest however this is not the case. A study by Bennedsen and Caspersen (2007) presents the finding that the worldwide pass rate for introductory courses is just 67%.

Compounding this problem is the proliferation of a variety of courses including a programming element into their syllabus. For example, around 120 students a year undertake the introductory web programming unit at Southampton Solent University. The participants are diverse, not only in initial ability, but also in their areas of study. Courses range from the more business aligned business information technology to the more technical discipline of software engineering.

The issues we encounter delivering the unit goes beyond just the pass rate and expands into the variability of the results, with some courses averaging over 70% and others below 37%. Herein lies a familiar dilemma when teaching programming to a diverse group of students (Cooper, Dann, & Pausch, 2000). A seemingly contradictory problem presents itself, how can we move at a pace such that the weaker students do not fall behind, while at the same time challenging the stronger students? Remarkably, there is a solution to this seemingly paradoxical problem proposed by (Bloom, 1984).

Over 30 years ago Bloom (1984) presented a seminal study demonstrating that providing certain conditions are met, almost all students can learn topics regardless of complexity. Firstly, what Bloom calls *mastery of learning* must be applied. Mastery of learning involves breaking subject matter down into manageable progressive chunks. Each chunk must be mastered before the next one is attempted. Secondly, each student must receive high-quality one-to-one tutoring (Bloom, 1984). In meeting these two conditions Bloom demonstrated that students could outperform those in a traditional classroom setting by 2 standard deviations. This equates, to the average student exposed to previously mentioned conditions, to outperforming 98% of students receiving traditional classroom teaching.

Such an increase in learning is indeed a tantalising proposition. However, it has been difficult to find replication studies that incorporate both tutoring and mastery of learning. Vanlehn (2011) found that although tutoring has a positive effect, Bloom's (1984) claim of 2 standard deviations might be too high. Kulik and Fletcher (2015) compared five meta analyses looking at the effects of peer tutoring in secondary and primary schools. The median effect of the improvement was 0.4 standard deviations.

Given the economic constraints faced by most higher education institutions, the cost of rolling out one-to-one support for all students is prohibitive. Bloom (1984) therefore derived the "two sigma problem." The premise is simple: Can group instructional methods be as effective as one-to-one tutoring? A more cost effective alternative is to utilise computer based technology and develop software that will provide students with enhanced feedback while at the same time still maintaining the traditional classroom setting.

In this paper, we present a proposed solution that demonstrates the implementation of portable intelligent exercises and trial them on our introductory web programming course at Southampton Solent University. We then go on to validate our system by surveying staff and students. Our initial findings are positive, suggesting that using such exercises support a diverse range of programming abilities.

Background

In order to set the scene of the problem we are trying to solve, Southampton Solent University must firstly be explored from the context of its place in the wider higher education ecosystem.

Southampton Solent University

Southampton Solent is a post-1992 UK based institution[1]. Typifying its post-1992 counterparts, the university has a strategy of widening university participation (Solent University, 2015). As such, the university has a diverse student population, with many of the students being considered *non-traditional* in terms of socio-economic and educational background (Read, Archer, & Leathwood, 2003). The widening higher education sector has presented challenges that were previously not encountered by more traditional institutions. A study by Thomas and Quinn (2006) found that non-traditional students, when compared to their traditional counterparts are often unprepared for their university experience. It is therefore not surprising that universities most successful at widening participation have some of the highest dropout rates (HESA, 2016). This problem is amplified when trying to teach complex topics such as programming, where even more traditional institutions have low pass rates (Bennedsen & Caspersen, 2007).

Computer Assisted Learning

The use of computer based instructional software to assist students in learning is not new. Computer based tutoring software can be traced back to the late 1960s (Atkinson, 1968). Vanlehn (2011) broadly categorized such systems into two groups, *computer based instruction* and *intelligent tutoring system*. Computer based instruction (CBI) aims to provide immediate feedback to students around some problem they are trying to learn. Intelligent tutoring systems (ITSs), aim in part to simulate a human by giving feedback and hints in the form of natural language (Vanlehn, 2011).

Both CBI and ITS systems generally consist of three core models (Hamed & Abu Naser, 2017).

> **Domain model.** This represents the body of knowledge that assists the students in learning.
>
> **Student Model.** This represents the actual student. It contains information that measures the student's mastery of specific topics belonging to the domain model.
>
> **Dialog Model.** This is the interface between the intelligent tutor and the student. It facilitates communication with the user of the system.

There is no agreed consensus on the increased learning effect that such systems have. A widely cited meta-analysis suggested that CBIs increases test scores 0.3 standard deviations over a standard classroom setting (Kulik & Kulik, 1991). A more recent meta-analysis by Kulik also Fletcher (2015) also recognised this to be the case.

The analysis by Kulik and Kulik (1991) had wide inclusion criteria and reviewed 245 studies. The studies covered a wide range of subjects, with participants ranging from every level of education. Kulik and Kulik stated the limitation of such large-scale meta-analysis is the time it takes to set up. When such studies are being constructed, rapid advancements in computing power and technology can occur. The consequence of the speed of such advancements means that the latest studies are often omitted from the analysis.

The latter meta-analysis by Kulik and Fletcher (2015) had a more stringent selection criteria. Studies were required to have a control group receiving conventional instruction, and CBI achievement outcomes must have been measured quantitatively. Like the earlier studies by Kulik and Kulik (1991), subject selection and age was wide ranging. Vanlehn (2011) revealed that the

ITSs are thought to outperform their CBI counterparts, producing increased test scores of 1.0 standard deviations. Vanlehn went on to note that these beliefs stem from an influential article by Anderson, Corbett, Koedinger,and Pelletier (1995) that summarised several ITS studies in a higher education setting. Studies were run with a programming, geometry and algebra ITSs over the course of a decade.

Given the generally accepted learning benefits of using such systems, their use in supporting higher education students to learn programming has been widely explored. Some examples that have shown positive feedback include a ITCs developed by Al-Bastami and Abu Naser (2017). It aims to assist university students in learning the programming language c#. A further example is JTITS, an ITCs system to assist in learning the programming language Java (Sykes & Franek, 2003). BITS is an example of a web based ITCs system (Butz, Shan Hua, & Maguire, 2004).

It must be noted that computer tutoring systems in the context of higher education are generally based on supporting traditional programming languages such as C, C#, C++ and Java. Such languages existed long before the widespread growth of the Internet. This has led to the ever-increasing popularity of web applications. At the time of writing, two web programming languages PHP and JavaScript are ranked as the 7th and 8th most popular programming languages used in industry (TIOBE, 2017). More and more universities are therefore teaching web-focused languages to beginner programmers. Surprisingly, considering the widespread use of such languages, computer based tutors to support students in a higher education setting have been lightly explored. Weragam and Reye (2013) claimed to have developed the first PHP intelligent tutoring system. However, there are no current systems to support students in learning JavaScript. We therefore deemed it necessary to investigate if such a system can be used to teach the specific web focused programming language of JavaScript.

System Architecture

When creating our system, we not only wanted to assist students in learning to program the web based programming language JavaScript, but also to encourage them to engage with the course content. We therefore decided to create an intelligent tutoring system that was not tightly coupled to the domain model (the course content).

The goal is to complement the body of information being delivered rather than replace it with an automated tutor. We shall refer to our system as smart intelligent exercises, as unlike the more traditional intelligent tutoring systems, it consists of only two modules; a domain model and dialog model.

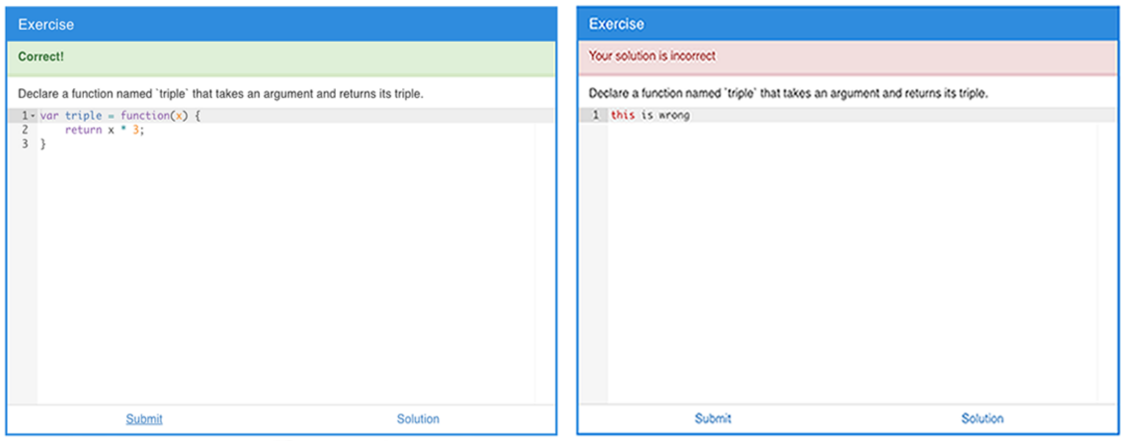
Domain model. To create the domain model, the course content was broken down into standalone sections and subsections. The goal is to keep the students focused on a very specific concept at a time. Breaking a subject matter down is such a way is one of the cornerstones of Bloom's mastery of learning (Bloom, 1968).

The entire body of information was structured using a tool called gitbook[2]. This tool allows notes to be distributed over the Internet. Students are not required to download any specific content to access the material. Furthermore, the content is device and operating system agnostic.

Dialog model. The interface for our intelligent tutoring system (Figure 1) had to be easily embedded into a web page. It was developed to be stand alone, in that it can function with or without the domain model.

The interface was developed from scratch using the programming language JavaScript, along with a number of freely available JavaScript tools. The ace text editor[3] was utilised to provide a realistic programming environment. The reader will note when observing Figure 1, that the code used to complete the exercise is multi colored. This is known as syntax highlighting and greatly increases the readability of programming code. Another key tool was a sandbox environment[4]. Such a tool allows the compilation of code and the evaluation of that body of code; this enables the correctness of a solution to be processed.

In order to create the questions, the operator must first define a problem and then map that problem to a solution. When student complete that problem, they are provided with instant feedback. Due to time constraints and the early stage of this research, the feedback is simplistic, indicating if the question is correctly answered and if any programmatic errors exist. Once the questions are created, they can simply be embedded into any web based content. See Figure 2.



Exercise

Correct!

Declare a function named `triple` that takes an argument and returns its triple.

```
var triple = function(x) {
    return x * 3;
}
```

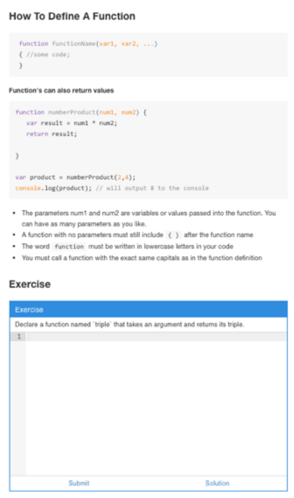
Exercise

Your solution is incorrect

Declare a function named `triple` that takes an argument and returns its triple.

```
this is wrong
```

Figure 1. Example of a how the exercises appear to students.



How To Define A Function

```
function functionName(var1, var2, ...)
{ //some code;
}
```

Function's can also return values

```
function numberProduct(num1, num2) {
   var result = num1 * num2;
   return result;

}

var product = numberProduct(2,4);
console.log(product); // will output 8 to the console
```

- The parameters num1 and num2 are variables or values passed into the function. You can have as many parameters as you like.
- A function with no parameters must still include `( )` after the function name
- The word `function` must be written in lowercase letters in your code
- You must call a function with the exact same capitals as in the function definition

Exercise

Declare a function named `triple` that takes an argument and returns its triple.

Figure 2. An example of a smart exercise embedded into the notes.

Evaluation

To validate the potential of our smart exercises, a study was conducted at Southampton Solent University during the academic year of 2016-2017. The study was delivered to first-year undergraduate students undertaking an introductory web-programming course. A total of 40 participants opted to take part in our evaluation survey.

Over a 4-week period smart exercises were embedded into subsections of the notes. Each subsection of the notes contained the necessary information to complete its corresponding embedded exercise (Figure 2).

The exercises were used in the weekly 2-hour practical sessions. At the start of the session the tutor would instruct the student to complete the smart exercises and demonstrate their solutions. They were required to complete these exercises before getting on with a main larger task. The hope was to in effect force the students to re-engage with the course content.

At the end of the 4-week evaluation, an ethically approved survey was distributed. The aim of the survey was twofold: firstly, to determine if students found intelligent exercises useful and secondly to tie this sentiment to their programming ability. We also distributed the survey to staff members who had experiences of delivering technical topics. Staff were asked to fill the survey out from the perspective of a novice programming student.

The survey questions were split into three sections. The first section was designed in order to determine how strong students are at programming. It consisted of six questions whereby students were required to rate how well they understood various fundamental programming concepts. Respondents were required to rank their understanding on a 5 point Likert scale, of 1 no understanding to 5 total understanding. Questions for this were based on a similar survey by Lahtinen, Ala-Mutka, & Järvinen (2005), which in part measures student's understanding of various programming techniques. Questions for the second and third sections were specific to our system and therefore developed by the authors.

The second section was developed to measure if the exercises were well received. Respondents were required to answer three questions. Questions were about the usefulness of the intelligent exercises. A 5-point scale was again used, with 1 meaning total disagreement to 5 meaning total agreement. The third section was a single open-ended question, asking for any further opinions. This allowed us to combine qualitative and quantitative feedback into a mixed methods single survey.

Results

The survey was distributed via email to 120 students: 40 students and 10 staff members completed it. The mean answers were calculated and are presented in Table 1..

Table 1

Mean Survey Results

Question	Student (40)	Staff (10)
CURRENT UNDERSTANDING OF PROGRAMMING		
(1 no understanding to 5 total understanding)		
1) Loop structures such as for and while loops	3.55	2.25
3) Understanding how to structure a program	3.55	2.28
4) Using variables and their scope	3.44	2.4
5) Designing a program in order to solve a given task	3.47	3.12
6) Designing and using functions	3.55	2.22
USING EMBEDDED EXERCISES TO SUPPORT YOUR LEARNING		
(1 strongly disagree to 5 strongly agree)		
7) Do you feel using embedded exercises could make JavaScript easier to understand?	4.00	4.50
8) Do you feel embedded exercises enhance the notes?	4.13	4.52
9) Do you feel using these exercise would make the learning process more enjoyable?	4.28	5.00

In Table 2, two sub student groups are identified: those that feel they understand the fundamental programming concepts and those that do not. We assumed students who scored 3 or above for each of the programming understanding questions feel confident in all the core programming topics. Those scoring below 3 on each of the programming understanding questions, we categorised as not confident. This process yielded a group of 24 confident programmers and 10 not confident programmers.

Table 2

Confident vs. Not Confident Programmers

Question	Confident (24)	Not Confident (10)
USING EMBEDDED EXERCISES TO SUPPORT YOUR LEARNING		
(1 strongly disagree to 5 strongly agree)		
7) Do you feel using embedded exercises could make JavaScript easier to understand?	4.16	4.2
8) Do you feel embedded exercises enhance the notes?	4.33	4.2
9) Do you feel using these exercise would make the learning process more enjoyable?	4.04	5.00

Discussion

Students overwhelmingly felt they had a stronger grasp of programming than the academic staff felt they had. Students consistently rated their actual understanding higher than that of the perception of the students.

The feedback both from the lecturers and students was positive. There was a slight weighting with regards to lecturers thinking that students would find the system slightly more useful than they actually did. Interestingly, when we split the students into groups that understood and did not understand the fundamental programming concepts, there was little difference in terms of sentiment. This promising result, suggests that our exercises can support programmers of all abilities. In fact, students who felt they were strong at programming thought they would get slightly more use out of the exercises than their not so confident counterparts. The only question where weaker students rated higher was question 9, which assesses whether using such a system is more enjoyable. One potential inference from this result is that perhaps weaker students want a more enjoyable learning experience whereas the stronger ones want greater challenges.

Out of the 40 student responses, 15 responded to the open question. Again the general feedback was very positive with comments such as:

> "This is definitely something I would use."
> "Makes learning to program much easier."
> "It's similar to codecademy which I like and find very useful."

With regards to academics, a similar sentiment to that of the students was shared with comments such as:

> "This will encourage students to engage with content."

Conclusion

We began this paper by presenting the widespread problem of students struggling to learn to program. We then proposed that the solution came in the form of the “two sigma problem” (Bloom, 1984), which presents us with the challenge of creating group instructional methods as effective as one-to-one tutoring.

In our search for a solution, we explored several computer based tutoring tools. We identified that such tools had limited support for web programming languages. Subsequently, smart exercises that could be embedded into the content were developed to assist students in learning to program in the web based JavaScript language.

Students used these exercises over a 4-week period. The feedback was overwhelmingly positive, and we therefore consider these exercises to have potential in the wider context of our web-programming course.

Limitations

Our survey was distributed to 120 student participants, however only 40 responded. Due to this sample not being a randomised, selection bias is a potential issue and must be taken into consideration. To gather qualitative data, students were required to respond to optional open-ended questions. Responses in this case were 15 students, this allowed for only limited inferences due to lack of responses.

Future Work

As future work we plan to implement two further modules into our system. The first is a student module that can map students’ learning paths and make recommendations based on their ability and performance. The further additional module will be an analytics module that will measure student engagement with the system. Such data will allow continual feedback to academic staff on how the cohort of students are performing. Following the implementation of these modules, long-term research addressing previous limitations could be run on the effectiveness of our system. The analytics module would allow us to come up with a measurement of student engagement, which could further enhance our research methodology.

Notes

1. The Conservative Party first issued university charters to a number of former polytechnics and higher education institutions in1992.
2. www.gitbook.com
3. ww.ace.c9.io
4. https://github.com/gf3/sandbox

References

Anderson, J. R., Corbett, A. T., Koedinger, K. R., & Pelletier, R. (1995). Cognitive tutors: Lessons learned. *Journal of the Learning Sciences*, *4*(2), 167-207.

Atkinson, R. C. (1968). Computerized instruction and the learning process. *American Psychologist, 23*, 225–239.

Bennedsen, J., & Caspersen, M. E. (2007). Failure rates in introductory programming. *ACM SIGCSE Bull*, *39*(2), 32-36.

Bloom, B. S. (1968). Learning for mastery. Instruction and Curriculum. Regional Education Laboratory for the Carolinas and Virginia, Topical Papers and Reprints, Number 1. *Evaluation comment*, *1*(2), n2.

Bloom, B. S. (1984). The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher*, *13*(6), 4-16.

Butz, C. J., Hua, S. & Maguire,R. B. (2004). A web-based intelligent tutoring system for computer programming. *Web Intelligence, 2004. WI 2004. Proceedings. IEEE/WIC/ACM International Conference on*. IEEE, 2004.

Cooper, S., Dann, W. & Pausch, R. (2000). Alice: a 3-D tool for introductory programming concepts. *CCSC'00: Proc. 5th Ann. CCSC Northeastern Conf. of the J. of Computing in Small Colleges* (pp. 107-116), Mahwah, NJ, USA: Consortium for Computing Sciences in Colleges.

Hamed, M. A., & Abu Naser, S. S. (2017). An intelligent tutoring system for teaching the 7 characteristics for living things. *International Journal of Advanced Research and Development* , *2*(1), 31–35.

Kulik, C.-L. C., & Kulik, J. A. (1991). Effectiveness of computer-based instruction: An updated analysis. *Computers in Human Behavior*, *7*, 75–94.

Kulik, J. A., & Fletcher, J. D. (2015). Effectiveness of intelligent tutoring systems a meta-analytic review. *Review of Educational Research*, *86*(1), 42-78.

Lahtinen, E., Ala-Mutka, K., & Järvinen, H. M. (2005, June). A study of the difficulties of novice programmers. *SIGCSE Bull, 37*(3), 14-18

Read, B., Archer, L., & Leathwood, C. (2003). Challenging cultures? Student conceptions of 'Belonging' and 'Isolation' at a post-1992 university. *Studies in higher education*, *28*(3), 261-277.

Sykes, E. R., & Franek, F. (2003, June). A prototype for an intelligent tutoring system for students learning to program in Java (TM). In *Proceedings of the IASTED International Conference on Computers and Advanced Technology in Education* (pp. 78-83).

Solent University. (2015). *Building an excellent university*. Available from: https://goo.gl/FUmREe.

Thomas, L., & Quinn, J. (2006). *First generation entry into higher education*. McGraw-Hill Education (UK).

TIOBE. (2017, April). TOBIE Index for April 2017. Retrieved March 24, 2017 from https://www.tiobe.com/tiobe-index/

Weragama, D., & Reye, J. (2013, July). The PHP intelligent tutoring system. In *International Conference on Artificial Intelligence in Education* (pp. 583-586). Berlin & Heidelberg, Germany: Springer.

Winslow, L. (1996). Programming pedagogy– A psychological overview. *SIGCSE Bull*, *ACM 28*(3), 17–22.

Vanlehn, K. (2011). The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems, *Educational Psychologist, 46*(4), 197–221.

Author Details

Joe Appleton
Joe.appleton@solent.ac.uk