On Teaching Arrays with Test-Driven Learning in WebIDE

Michael Hilton, David S. Janzen
California Polytechnic State University
San Luis Obispo, California USA
{hilton,djanzen}@calpoly.edu

ABSTRACT

Test-driven development (TDD) has been shown to reduce defects and to lead to better code, but can it help beginning students learn basic programming topics, specifically arrays? We performed a controlled experiment where we taught arrays to two CS0 classes, one using WebIDE, an intelligent tutoring system that enforced the use of Test-Driven Learning (TDL) methods, and one using more traditional static methods and a development environment that instructed, but did not enforce the use of TDD. Students who used the TDL approach with WebIDE performed significantly better in assessments and had significantly higher opinions of their experiences than students who used traditional methods and tools.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Computer Science Education*

General Terms

Experimentation

Keywords

CS0, CS1, computer science education, intelligent tutor

1. INTRODUCTION

Test-driven development (TDD)[1] is a software engineering best practice that involves writing fine-grained automated unit tests prior to corresponding code, then refactoring in short, rapid increments. Numerous studies have examined the efficacy of test-driven development (TDD)[13, 3] with encouraging, but sometimes mixed results. For instance, TDD studies generally report improved software quality, but sometimes at the expense of lower productivity [2]. In addition, while many of these studies report very promising results with advanced students, incorporating TDD with beginning students has been more challenging. Desai et al. [4] report that introductory students who were taught TDD wrote higher quality code as measured through code-coverage. However, there was not significant change in the quality of source-code, the time spent on projects, attitudes towards testing or overall comprehension of material. It was their conclusion that simply incorporating TDD into current course materials with existing tools was not ideal, and that what was needed was some re-ordering and re-emphasizing of the material. We consider whether teaching an introductory programming topic, namely arrays, in a test-driven manner with the support of WebIDE, a novel web-based intelligent tutoring system that can enforce the test-driven approach, has any effect on student learning. Section 2 discusses related work, establishing context and motivation for this work. Section 3 describes the lab we created for teaching beginning programming students about arrays. Section 4 reports results from a controlled experiment to evaluate the new instructional materials, and Section 5 suggests some conclusions and future work.

2. RELATED WORK

2.1 Test-Driven Learning

Test-Driven Learning (TDL) [6] was proposed as an approach to teach computing students new topics using automated tests as examples. The basic idea is that TDD can be taught for free by simply incorporating a TDD approach to the material being taught. When an instructor introduces a topic such as recursion, as they present examples they can include automated tests as the first step of writing a source code implementation.

2.2 WebIDE

Although demonstrating TDD to students can be motivating, actually getting them to write code in a test-first manner can be challenging. WebIDE [5] is a novel intelligent tutoring system framework for delivering interactive web-based labs designed specifically for applying TDL in the first few weeks of introductory programming courses. WebIDE helps students during these difficult early weeks by offering a one-button interface in a ubiquitous and familiar web context that requires no additional installation.

Significant work on intelligent tutors exists [14], but WebIDE appears to be the first open and scalable system designed to incorporate TDL. WebIDE's architecture separates lab specification, lab rendering, and automated evaluation. Labs are specified in an xml file located at any URL. Labs embed references to evaluators that process and respond to student submissions. WebIDE includes internal evaluators such as regular expression matching. An unlimited number of external evaluators can be located on any Internet-accessible server. External evaluators can range from generic evaluators that compile and/or execute complete programs or test suites, to custom evaluators that parse very specific expressions (e.g. the sum of two integers).

Dvornik et al. [5] conducted a pilot study in 2010 using WebIDE to teach students using TDL. Although WebIDE was still under development they did see significant improvement in students developing a beginning Android application over the non-WebIDE students. However, the WebIDE students did not perform better on programming tasks overall. During the experiment students experienced significant evaluator timeouts on some labs that prevented them from completing all labs. This was a significant threat to validity. The timeout issue and other reliability issues were resolved prior to conducting the study reported in this paper.

2.3 Beginning Programmer Experiments

While most of the research into TDL and TDD has been with advanced students, there has been some research into TDL and TDD for beginning students.

2.3.1 Effects on Student Learning

Janzen and Saiedian [6] conducted a short experiment in two sections of a CS1 course that used C++. The experiment was conducted in three fifty-minute lectures and one fifty-minute lab that covered the introduction of classes and arrays. While both sections had been introduced previously to the assert() macro, during this experiment the first section was instructed using TDL and the second section was presented examples in a traditional manner using standard output with the instructor explaining the expected results. At the end of the experiment, all students were given the same short quiz. The quiz covered concepts and syntax from the experiment topics. The section that had been instructed using TDL scored a higher average on the test than the non-TDL section by a margin of 7.84 to 7.14. The sample size was relatively small with only 27 participants, so broad conclusions cannot be made.

2.3.2 Beginning Programmer Resistance

Many TDD studies address student appreciation of TDD and willingness to adopt TDD with mixed results. Janzen and Saiedian [7] performed an examination of acceptance of TDD by students at various levels. They found that after learning about TDD in the study, over 60% of mature developers would choose test-first over test-last, while only 10% of beginning developers would choose test-first methods. However, among several confounding factors they note that the mature developers used Java and JUnit while the beginners used C++ and assert statements.

2.3.3 Few choosing TDD

Garcia et al. [8] report that teaching beginning students in a test-driven manner resulted in only 10% of students choosing to use automated tests. However, there were several differences in their approach that may account for why so few students chose to use TDD. The students were taught in a classical procedural language and therefore they were taught a specially designed testing framework called tpUnit. The creators of the study chose not to use JUnit because they felt it would be too complex for beginning students. By using WebIDE we were able to abstract out the more complex features, and thereby teach beginning students Java and testing with JUnit at the same time. Garci et al. did find that the students who choose to write the tests scored higher than the students who did not, but due to low participation rates they couldn't draw definitive conclusions.

2.3.4 TDD least popular of XP practices

A study performed by Keefe [9] introduced students to four XP practices: Pair Programming, Test Driven Development, Simple Design, and Refactoring. Of the four practices TDD was least preferred by the students in the study. There were several reasons why students did not like TDD. They found that students did not have a sufficient understanding of testing to be able to write tests effectively. Many of the students found JUnit to be too complicated for them. They also found that many of the students had difficulty understanding why testing was important, and felt as if it was a waste of their time. While the authors recommended that testing needs to be introduced to programming students, they believed that their study was inconclusive on whether beginning students are capable of test first development.

2.3.5 Age and attitudes towards TDD

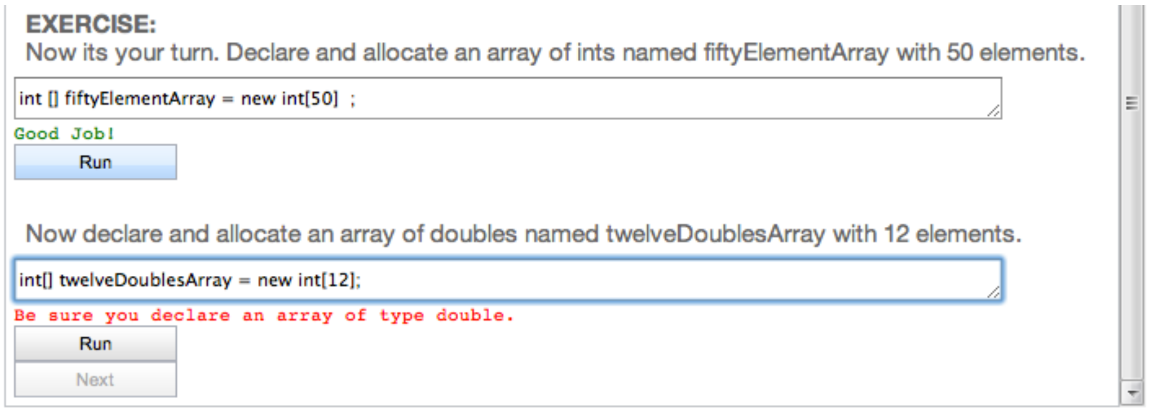
An investigation into students' perceptions of Extreme Programming (XP) practices conducted by Melnik [10] found a positive correlation between a students' age and a positive attitude towards TDD. They considered that this might be explained by a higher level of discipline of more mature students. They believe that this is because TDD is not about testing but about design. They feel that the older students are better at design, which in turn leads to them having a better opinion of TDD than beginning students.

2.3.6 Proper incentives needed

To help students appreciate TDD, Spacco and Pugh [11] developed a system called Marmoset for student project programming and submission. They would distribute projects to students including documentation, skeleton code, and several test cases in order to give the students a starting place. They found that as students had not yet learned the value of TDD, they needed to incentivize the students to write tests. In order to do that, they based a part of the students' grades on the amount of automated testing code coverage delivered. They also tried to design the system to encourage students to write tests. They found that as a result of their efforts, the students did indeed write and submit test cases as part of their final submission. However, upon further exploration, they found that a significant number of students did substantial work on their test cases after they had passed all the submission tests. It is their conclusion that without the proper incentive to write test cases early, many students do not adopt a test-first mentality, but rather stick to the test-late mentality of writing their implementation and then testing it at the very end.

3. TDL ARRAY LESSON

We created a new lab to teach introductory students about the topic of arrays in Java using the TDL approach and WebIDE. Each step is described below.


EXERCISE:
Now its your turn. Declare and allocate an array of ints named fiftyElementArray with 50 elements.

```
int [] fiftyElementArray = new int[50] ;
```

Good Job!

Now declare and allocate an array of doubles named twelveDoublesArray with 12 elements.

```
int[] twelveDoublesArray = new int[12];
```

Be sure you declare an array of type double.

Figure 1: WebIDE lab step 1 exercise with sample error feedback

3.1 Step 1: Introduction to Arrays

The first step of the WebIDE lesson is titled "Introduction to Arrays". The purpose of this step is to introduce the students to the basics of Java arrays including the concepts of indexes and declaration, along with basic syntax. The lab targets beginning students so we assume the students do not have any knowledge about arrays before coming into this class. After seeing examples of array declarations, students are asked to declare two simple arrays before being allowed to move to the next step. Figure 1 shows a sample input where the student entered the first declaration correctly, but forgot to change the data types in the second declaration. These very simple first steps are to help the student build confidence and to not be overwhelmed too early. This will attempt to break down some of the initial resistance to TDL other studies have found to be common in beginning students [9, 8].

3.2 Step 2: Populating Arrays

The second step is "Populating Arrays". We emphasize how indices start at 0, a common source of confusion when dealing with arrays. We also introduce the first test in an exercise for the students. We assume that they have already been exposed to tests as part of their coursework in this class, so we do not need to go into detail about what the tests are or how they will work. Others [7, 10] have reported that beginning students feel test-last is more intuitive to them than test-first, so we start with one test-last example to help them feel comfortable before we move on to test-first. The exercise in this step is a simple task, to populate an array with the first five prime numbers. We give them the first five prime numbers to minimize external barriers to success with the topic at hand. Since we have assumed familiarity with the concept of variables, we do a simple assignment much like they have done with integers in the past.

3.3 Step 3: Retrieving Values from Arrays

The next step is "Retrieving Values from Arrays" and is shown in Figure 2. Here we want to introduce students to the concept of retrieving a value from an array and using that value. The exercise that the students will need to do this time is to sum all the values that they put into the array in the previous step. Before they write this code, they will set up a test to ensure that they have the correct sum. We have gradually been introducing testing, but at this point they are now doing test-first development. In order to avoid the confusion that has bothered other students in the past [9] we attempt to make the tests as straight forward as possible and directly related to the outcome of the code.

3.4 Step 4: Out of Bounds

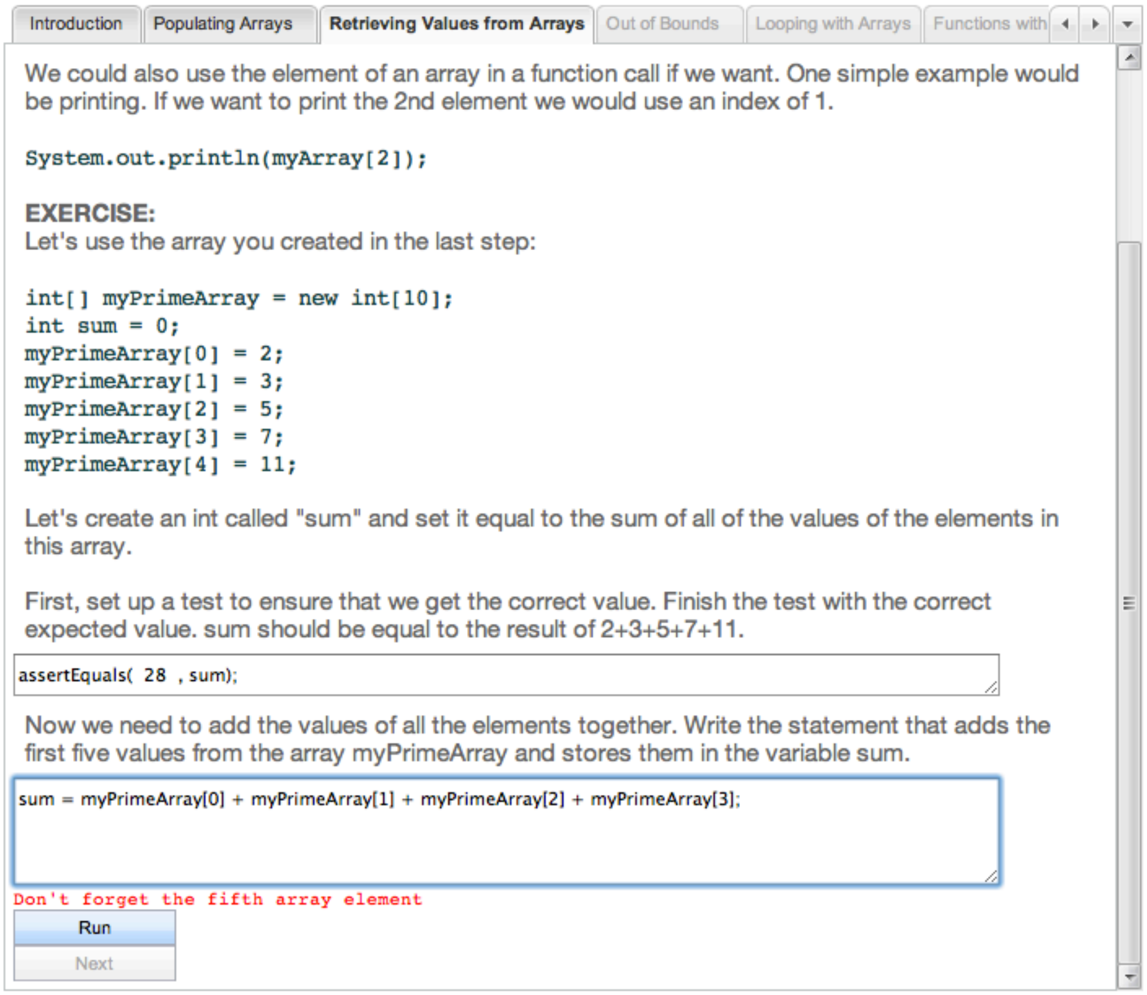
Step 4 is designed to help students learn more about array indices. It also is designed to introduce them to the concept of Java exceptions. This step uses a WebIDE external Java Evaluator so that students could see an actual Java exception being thrown based on the code. We assume that the students at some point would write some code in which the array index is out of the bounds of the array, resulting in an exception being thrown. This step also provides an additional opportunity to work with array indices which must be mastered before they move on to more advanced exercises. This step starts with erroneous code that throws an exception. After running the code and seeing the exception, the student must fix the defect and see the code run successfully.

3.5 Step 5: Looping with Arrays

Now that we have been over the basics of arrays, we will start to cover concepts that are a little more complicated. This step will teach students how to work with arrays in loops. Since the students have already covered loops in this course, we do not need to introduce them as a new concept. Our focus is on teaching students to understand how they can use a loop to work with an array. We are also using this as an opportunity to continue to develop their understanding of test-first development. Since TDD has been shown to help programmers design better code [6], we will guide the students to write tests that will help them with their code design, as the exercises get to be more complicated. In this step, we give them the first test, then we use the internal WebIDE Regular Expression evaluator to ensure that they write the correct test for each test case, which in this instance corresponds to each index of the array that they will be populating. Once the students have written the correct test cases, they will get feedback as they write their code. If their design is incorrect, the test cases will show them.

3.6 Step 6: Functions with Arrays

This step focuses on teaching students how to use arrays with functions. As with the previous step, since they have



We could also use the element of an array in a function call if we want. One simple example would be printing. If we want to print the 2nd element we would use an index of 1.

```
System.out.println(myArray[2]);
```

EXERCISE:
Let's use the array you created in the last step:

```
int[] myPrimeArray = new int[10];
int sum = 0;
myPrimeArray[0] = 2;
myPrimeArray[1] = 3;
myPrimeArray[2] = 5;
myPrimeArray[3] = 7;
myPrimeArray[4] = 11;
```

Let's create an int called "sum" and set it equal to the sum of all of the values of the elements in this array.

First, set up a test to ensure that we get the correct value. Finish the test with the correct expected value. sum should be equal to the result of 2+3+5+7+11.

```
assertEquals( 28 , sum);
```

Now we need to add the values of all the elements together. Write the statement that adds the first five values from the array myPrimeArray and stores them in the variable sum.

```
sum = myPrimeArray[0] + myPrimeArray[1] + myPrimeArray[2] + myPrimeArray[3];
```

```
Don't forget the fifth array element
```

Figure 2: WebIDE lab step 3 with sample error feedback

covered functions previously in the course we will not have to introduce functions from scratch, but will focus on what the students will need to know to work with functions and arrays. The two main concepts that we want to teach the students is how to pass arrays into functions as parameters and how to return functions from arrays. We provide the students with some example code that is very similar to the code that we wish them to write. Then we set them up with some partial test cases so that they will be able to develop the test cases before they write the code. The code required to complete this exercise involves writing a function that takes an array as a parameter and then returns an int that is the count of the number of positive integers in the array. This exercise was designed with the goal of not only forcing the student to understand the information that was just given to them about arrays and functions, but also to reinforce the last step, where they learned how to use loops to deal with arrays. By building on the previously taught concepts we hope to reinforce the students' understanding.

3.7 Step 7: Array of Objects

We will continue to build on previous information with this step. Up until this point we have only used arrays of primitive types. In this step we will introduce arrays of objects. As before, we are assuming that they are familiar with objects and so we are only concerned with this lesson covering the information that the students will need to be able to use an array of objects. We give them an example of an array of PlayingCard's. For the exercise we will ask them to write a piece of code that will declare an array of type President and fill it with some information about four presidents. However, before they write this code we will give them an example test and a partial test, and ask them to finish the partial test. Once they have written this code, we will use this array for the next step. However, since we can use WebIDE to force them to finish this step correctly before they move on, we know that they will have answered this step correctly when we arrive at the next step.

3.8 Step 8: Operations On Array Elements

As the eighth and last step, we will tie in all the concepts that we have taught in the previous steps of this lesson. We will have the students iterate through the array that they created in the previous step which contains four presidents and the years that they started their presidency and the year that they ended their presidency. We will start the students off with part of a test, which they will need to complete. This test will be straight forward as it will test the sum of all the years in office of all the presidents in the array. This test was chosen because in order to complete it, the student will have to figure out on paper how they will come up with a number of years based on the information that they have been given. This computation is the type of design work that we would like them to do before they begin to write code. It also benefits the student because once they have that number, it will provide them with a test so that they know that their code is working properly once they have written it. In order to complete this assignment, the student will have to use concepts that we have previously taught in the lesson such as array indices, looping, functions, arrays of objects and array values. By continuing to use the same concepts repeatedly, we hope to reinforce the learning.

4. RESULTS

In order to evaluate the effects of the TDL arrays lab, we conducted a controlled experiment with 72 students in two sections of an introductory computing course (CS0) for students who had selected a computing major. Students were given a pre-study programming quiz at the beginning of the course to determine their programming skills entering the course. Results on this quiz indicated no statistically significant difference in previous programming experience between the WebIDE and control groups, although the WebIDE group did score slightly lower. Students completed the new array lab in the eighth week of the ten week course during a 90 minute closed lab period. This particular course introduced students to computing in the context of mobile applications. Prior to the lab on arrays, students completed labs using Scratch, App Inventor, and Java including Java basics (data, operators, expressions), methods, selection, looping, and classes. Each section contained 36 students and was taught by the same instructor with identical content. The independent variable was the use of WebIDE. Students in the control section were presented the same content in static html lab pages, and they used the Eclipse integrated development environment and were provided boiler-plate code for completing the exercises (e.g. Java and JUnit classes). Students in the control group had been using Eclipse for three weeks prior to this lab, whereas students in the WebIDE section had been introduced to Eclipse, but were completing their labs in WebIDE. Students in both sections were required to demonstrate proficiency in Eclipse by creating an Android app in a six-week group project.

4.1 Overall Quiz Scores

At the end of the lab period, students were given a six question on-line quiz on array concepts and syntax in Java worth a total of 22 points. The control group had 36 students take the quiz and the experimental group had 35 students take the quiz. One week later, students were also given a comprehensive lab quiz that included a question on Java arrays and loops. Students were encouraged to redo and study the lab outside of class in preparation for the comprehensive lab quiz. Table 1 reports student performance on the computer programming assessments, demonstrating the weaker performance on the pre-experiment assessment, and better performance after the introduction of WebIDE. Statistical significance was determined with $p < .05$ using a one-tailed univariate analysis of covariance test controlling for each student's score on the pre-study programming quiz.

4.2 Individual Question Scores

Since TDD has been shown to help improve software design [6], we would expect that TDL would be more beneficial for the more complicated problems, and there to be less of a difference on the less complicated quiz questions. That is exactly what we see in Table 2. Questions two and three were the simplest questions, requiring the least recall and corresponding to steps one and two. Questions two and four were both multiple choice, requiring more recognition over recall, but question four covered the more difficult topic of arrays of objects, corresponding to step seven. Questions five and six were the most open-ended and challenging, requiring students to write entire tests and methods based only on a problem description. Consequently, the control group even performed slightly better than the experiment group on question two, which was a very simple question. On the most complicated question (number six) we see the largest difference of 19%.

4.3 Student Opinions

Students were asked their opinions on the course, the labs, and their perspectives on the course topics (Java and Android) in a post-experiment on-line survey. Table 3 reports these results based on a five point Likert scale with one being "Strongly Agree" and five being "Strongly Disagree." The WebIDE students were more positive about the array lab than the control group.

4.4 Threats to Validity

As with any academic study there are unavoidable threats to validity such as the course time of day and external social or academic factors. One main threat to validity is that we are unable to determine if the better test scores were due exclusively to students using WebIDE, and that if they have used WebIDE with a different learning style they would have shown similar improvement over the control group.

5. CONCLUSIONS AND FUTURE WORK

This experiment has shown that students who were taught arrays in a TDL manner with a system that enforced the TDL approach performed better on assessments of the array topics than students who were taught the same material in a more traditional manner. Further, the students who used WebIDE were more positive about the course and the labs, than students who used a traditional development environment.

Future work includes extending the same concepts that were used to teach beginning students arrays in a test-driven manner, to other CS0/CS1 topics, such as functions and loops. Also, since WebIDE is not intrinsically built on TDL, an experiment could be devised that compares teaching the same material using WebIDE and TDL, and also using WebIDE but with a different teaching paradigm.

6. ACKNOWLEDGMENTS

We would like to thank Clark Turner for his work reviewing early versions of this paper, John Clements for his work developing WebIDE, and Olga Dekhtyar for reviewing all statistical analyses.

Assessment	Total Possible	WebIDE Mean	Control Mean	Significant?
Pre-Study Programming Quiz	25	2.72	3.50	No
Lab 8 Quiz: Java Arrays	22	13.75	11.44	Yes
Comp. Lab Quiz: Java Arrays/Loops	12	10.17	8.75	Yes

Table 1: Student Performance on Programming Assessments

Question	WebIDE Mean	Control Mean	Difference	Total Possible	Percent Difference
1 (declare array)	1.80	1.58	.22	2	10.83
2 (last element m/c)	1.83	1.94	-.12	2	-5.80
3 (first element)	1.29	1.19	0.091	2	4.56
4 (array of objects m/c)	1.6	1.39	.211	2	10.55
5 (test for getSmallest method))	3.49	2.722	0.76	6	12.72
6 (method getSmallest)	4.14	2.61	1.53	8	19.15

Table 2: Quiz Scores by Question

Question	WebIDE Mean	Control Mean	Significant?
I liked this course	1.59	2.03	Yes
I am comfortable with Java	2.38	2.47	No
Labs helped learn Java	2.21	2.64	Yes
Liked Lab 8	2.34	2.81	Yes

Table 3: Student Opinions

This material is based upon work supported by the National Science Foundation under Grant No. 0942488. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

7. REFERENCES

[1] K. Beck "Test Driven Development: By Example," *Addison Wesley*, November 2002.

[2] S. Kollanus, "Test-driven development - still a promising approach?" *7th Int. Conf. on the Quality of Information and Communications Technology (QUATIC)*, Porto, Portugal, Oct. 2010, pp. 403 – 408.

[3] C. Desai, D. Janzen, and K. Savage "A survey of evidence for test-driven development in academia," *SIGCSE Bulletin,* vol. 40, no. 2, 2008, pp. 97 – 101

[4] C. Desai, D. Janzen, and J. Clements, "Implications of integrating test-driven development into CS1/CS2 curricula," in *Proceedings of the 40th ACM technical symposium on Computer science education.* New York, NY, USA: ACM, 2009, pp. 148–152

[5] T. Dvornik, D. Janzen, J. Clements, O. Dekhtyar, "Supporting introductory test-driven labs with WebIDE" *Software Engineering Education and Training (CSEE and T), 2011 24th IEEE-CS Conference on* 22–24 May 2011, pp. 51–60.

[6] D. Janzen and H. Saiedian, "Test-Driven Learning: Intrinsic Integration of Testing into the CS/SE Curriculum." *In Proc. 37th Technical Symposium on Computer Science Education (SIGCSE),* pages 254–258. ACM, 2006.

[7] D. Janzen and H. Saiedian, "Test-Driven Learning in Early Programming Courses." *In Proc. 39th Technical Symposium on Computer Science Education (SIGCSE).* ACM, 2008.

[8] E. Barriocanal, M. Urban, I. Cuevas, and P. Perez, "An Experience in Integrating Automated Unit Testing Practices in an Introductory Programming Course." *ACM SIGCSE Bulletin,* 34(4):125–128, December 2002.

[9] K. Keefe, J. Sheard, and M. Dick, "Adopting XP Practices for Teaching Object Oriented Programming." *In Proc. 8th Australian Conf. Computing Education,* volume 52, pages 91–100, 2006.

[10] G. Melnik and F. Maurer, "A Cross-Program Investigation of StudentsÕPerceptions of Agile Methods." *In Proc. 27th Intl. Conf. on Software Eng. (ICSE),* pages 481–488. ACM Press, 2005.

[11] J. Spacco and W. Pugh, "Helping Students Appreciate Test-Driven Development (TDD)." *In Companion to 21st. ACM SIGPLAN Conf. Object-Oriented Prog. Systems, Languages, and Applications (OOPSLA),* pages 907–913, 2006.

[12] D. Janzen and H. Saiedian, "A Leveled Examination of Test-Driven Development Acceptance." *In Proc. 29th Intl. Conf. on Software Engineering (ICSE),* pages 719–722. IEEE Press, 2007.

[13] R. Jeffries and G. Melnik, "Guest Editors' Introduction: TDD–The Art of Fearless Programming." *IEEE Software,* volume 24, pages 24–30. IEEE Computer Society, 2007.

[14] R. Raga and J. Raga, "A Survey on Intelligent Tutoring Systems (Its) and it's Authoring in the Context of Features Promoting the Human-Teacher Factor." *Journal of Business, Education and Law,* volume 14, no 1, Jose Rizal University, 2010.