Proceedings of the International Conference on Computer and Information Science and Technology
Ottawa, Ontario, Canada, May 11 – 12, 2015
Paper No. 132

Spinoza: The Code Tutor

Fatima Abu Deeb, Timothy Hickey
Brandeis University
Waltham,MA,USA
abudeebf@brandeis.edu;tjhickey@brandeis.edu

Abstract- The Spinoza Code Tutor is a web application that has been designed to support active learning in Introductory (CS1) Java programming classes. Spinoza has two main modes: Exercise and Homework. The Exercise mode is designed for use in a large fully-flipped CS1 class. It supports a pedagogy in which the instructor creates a programming exercise on-the-fly during the class session by providing a brief description and initial scaffolding-code. The students then use their browsers to write, compile, run, and debug their code. The Homework mode extends Exercise mode by running a suite of unit tests that provides a Test Driven Development experience for novice students and also serves as an auto-grading feature. The Homework mode requires the instructor to write both a solution for the programming problem as well as a unit test generator, in addition to the problem description and initial code. Spinoza offers an Instructor View of the assigned problem which shows the current code for every student in class. This view groups programs that are essentially the same (i.e. same class file) together so that the most common solutions can be shared with the class first. Incorrect student code can be used to provide in-class debugging practice, using the Think/Pair/Share methodology. Spinoza has been tested in two sections of an Introduction to Programming in Java class with about 150 students per section in Fall 2014. In this paper, we describe the features of Spinoza, report on our experience using Spinoza in those classes, and discuss future directions for the use of web-based Integrated Development Environments in Flipped Programming Classes.

***Keywords*:** Flipped Learning, Web-based IDEs, Educational Technology, Computer-aided Assessment, Audience Response Systems, Markov Models, Novice programmers, CS1

1. Introduction

Spinoza is a web-based pedagogical Integrated Development Environment (IDE) that teachers use to create programming problems in Java and students use to attempt to solve those problems. It also features sophisticated tools for the instructor to monitor the progress of the students, as a group or individually, in real-time and provides extensive scaffolding to help the students solve the problems. Spinoza aims to make flipped introductory programming classes (CS1) more effective and efficient for both students and teachers.

2. Design of the Spinoza Code Tutor

Spinoza can be used in three modes. In the Java IDE mode it is a simple web-based Integrated Development Environment (IDE) that allows students to write and run Java programs in their browsers without having to download a Java compiler or any code editors. This allows the instructor to introduce the students to Java coding in the first few minutes of the first class and it allows students to use a low cost Chromebook while learning to code. There are many web-based IDEs that allow students to code in a variety of languages using only their browser [(web-2),(web-3),(web-4)]. What makes Spinoza unique are the other two modes which are designed to support student coding with instructor supervision in a flipped classroom. Both of these modes provide a rich set of instructor views giving an overview of the progress of the entire class on a problem as well as a detailed history of any individual student's progress.

2. 1. Spinoza Exercises

In the Spinoza Exercise mode, the students are given a coding problem to solve along with some initial code (e.g. everything but the body of one method) and they write and run the program in their browser. If they run into problems, they message the instructor or one of the Teaching Assistants (TAs) using a separate system and the instructor and TAs view and run their code remotely using the Spinoza Instructor View without changing the student's view of their code.

A typical use of the Spinoza Exercise feature is in a Think/Pair/Share (Lyman, 1987) coding challenge in which the instructor provides a Spinoza exercise that allows the students to get practice with a concept from the reading. For example, after learning about arrays the students can be asked to write the body of a method that finds the sum of the positive elements in an array of integers. The instructor opens the Exercise creation view and briefly enters a description of the problem to be solved, as well as some initial code to use while explaining the problem to the students. The students then *Think* about the problem and use Spinoza to try to write their own solution. After a few minutes, the instructor suggests that they *Pair* up and talk to their neighbors (or chat with the TAs) about their attempted solution. Finally the instructor *Share*s their solutions with the rest of the class, using Spinoza to project the code from individual student's attempted solutions on the screen. If too many of the students are still confused, the instructor reviews the concepts and easily creates another similar Spinoza Exercise, on-the-fly, to allow them to test their understanding.

The instructor and the TAs can view the progress of the class dynamically using the Spinoza Instructor View which shows which students have written and tried to run the program, and which of those students still have syntax errors. This helps the instructor decide when to switch between the Think, Pair and Share phases of the Think/Pair/Share activity. Figure 1 shows a screenshot of this view (for a small recitation with the student names anonymized for publication). When a sufficient number of students have tried to solve the given problem, the instructor displays (and discusses) the code of individual students using the Instructor View projected at the front of the class.

The Instructor View groups similar programs together so that the most common solutions can be viewed and discussed first. In practice, many of the student programs will have syntax errors or logical errors, and Think/Pair/Share can be applied again to the debugging process as everyone in the class is enjoined to find the bugs in that particular student's code. Group debugging both exposes the students to the subtleties of programming and shows them that many of their classmates are making similar mistakes.

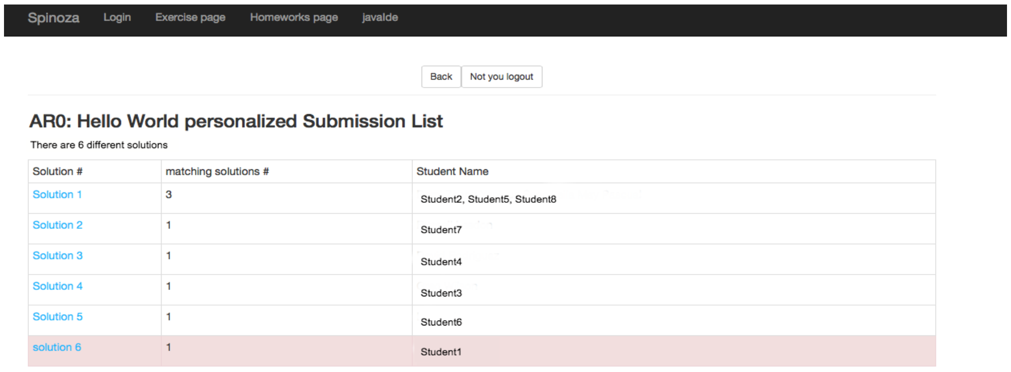
2. 2. Spinoza Homeworks

In the Spinoza Homework mode, the students again see a coding challenge which they solve in their browser, but each time they run their code the system also runs a large and varying set of instructor- supplied unit tests for one particular method and displays, for each input, the answer their method produced and the expected answer produced from an instructor-supplied solution.

Spinoza Homework mode introduces them to Test Driven Development and helps them eliminate logic errors in their program. The unit test cases are generated by an instructor-supplied program which typically contains some particular corner cases as well as a set of randomly generated tests which will change each time the program is run. Figure 2 shows the student view of a Spinoza Homework problem.

To create a Spinoza Homework problem the instructor must provide three programs. The first is the initial code the student sees. This allows the instructor to provide some scaffolding to get the students off to a good start. The second is a unit test case generator, which will generate a set of unit tests for a specific method in the program. The third is a solution to the problem, which is used to find the correct return value for the generated tests. This data is used to show the student whether or not the values computed by their programs are equal to the expected values.

We have also used Spinoza Homework mode to create inductive problems similar to those in Code Hunt (Tillmann et al., 2014) where the students are not told what the method is supposed to compute, rather they are challenged to examine the unit tests and infer what the method should return and then attempt to code up that solution.

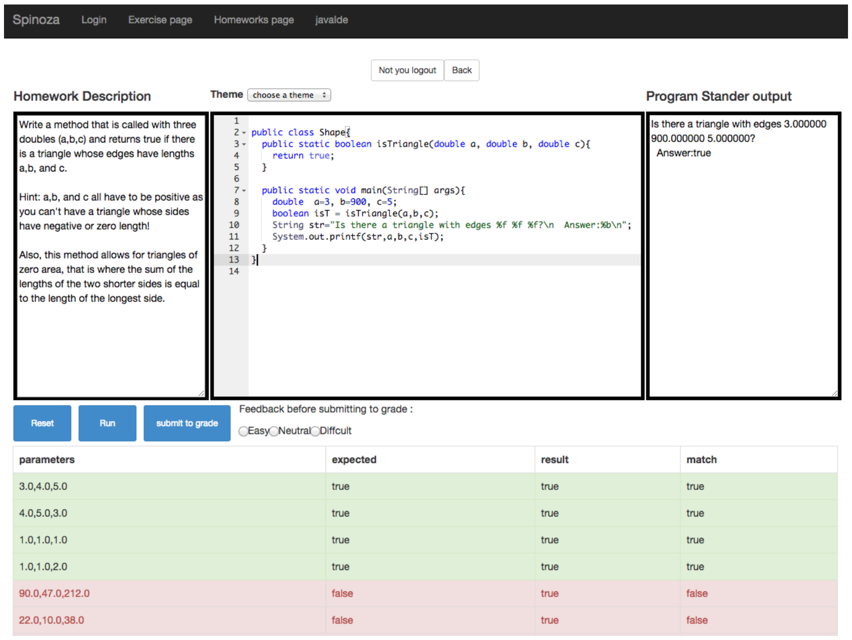


AR0: Hello World personalized Submission List

There are 6 different solutions

Solution #	matching solutions #	Student Name
Solution 1	3	Student2, Student5, Student8
Solution 2	1	Student7
Solution 3	1	Student4
Solution 4	1	Student3
Solution 5	1	Student6
solution 6	1	Student1

Fig. 1. The Spinoza Instructor View of an Exercise showing that there are 6 different attempted solutions' one of which still has syntax errors (Solution 6). It also shows that 3 students have submitted essentially the same code (Solution 1) where whitespace and local variable names are normalized and comments are removed.



Homework Description

Write a method that is called with three doubles (a,b,c) and returns true if there is a triangle whose edges have lengths a,b, and c.

Hint: a,b, and c all have to be positive as you can't have a triangle whose sides have negative or zero length!

Also, this method allows for triangles of zero area, that is where the sum of the lengths of the two shorter sides is equal to the length of the longest side.

```

public class Shape{
  public static boolean isTriangle(double a, double b, double c){
    return true;
  }

  public static void main(String[] args){
    double  a=3, b=900, c=5;
    boolean isT = isTriangle(a,b,c);
    String str="Is there a triangle with edges %f %f %f?\n  Answer:%b\n";
    System.out.printf(str,a,b,c,isT);
  }
}

```

Program Stander output

```
Is there a triangle with edges 3.000000
900.000000 5.000000?
  Answer:true
```

Feedback before submitting to grade :

Easy Neutral Diffcult

parameters	expected	result	match
3.0,4.0,5.0	true	true	true
4.0,5.0,3.0	true	true	true
1.0,1.0,1.0	true	true	true
1.0,1.0,2.0	true	true	true
90.0,47.0,212.0	false	true	false
22.0,10.0,38.0	false	true	false

Fig. 2. This shows the student view after clicking on the run button in a Spinoza Homework. This problem asks them to write a Boolean method to determine if the three inputs can be the edges of a triangle.

The Homework mode also has an Instructor View similar to that for Exercises and can be used in the same way to run Think/Pair/Share coding activities in class. It is more difficult to create a Spinoza Homework problem on the fly as the instructor must not only describe the problem and provide some initial code, he or she must also provide a solution and a unit-test generator. The Spinoza Exercise was designed to remedy that situation and allow instructors to rapidly create another programming problem when the instructor determines that they need additional practice with a concept.

3. Experience in Using Spinoza in a Large CS1 Class

We used Spinoza in Fall 2014 to teach a CS1 Introduction to Programming in Java course with 284 students in two sections (136 in one section and 148 in the other). The course was only partly flipped. Each class session consisted of a sequence of 4-6 activities, typically including a brief review of the concepts covered in the reading, followed by several Think/Pair/Share coding and debugging challenges. Students who were at risk of doing poorly in the class were encouraged to attend Advanced Recitations which combined pair programming and Spinoza practice to solidify understanding of programming concepts.

Spinoza is implemented as a NodeJS app, which compiles and runs student programs on a department server. It is easily able to handle the load of 150 simultaneous users. Every time a student runs their code, Spinoza stores a time-stamped version of their code in a database. This feature is used to provide the instructor with a much more nuanced view of student performance and understanding by revealing not just the current state of their programs, but their entire history of interaction with each problem. At the most detailed level it allows the instructor to view every version of the students program and to follow their coding and debugging process in detail. At a higher level, it allows the instructor to see the total number of attempts made by a student for each problem.

Spinoza was used in three ways in this course. The first, most common use of Spinoza was to manage Think/Pair/Share activities in class consisting of coding challenges and/or debugging challenges as described in the previous section. Spinoza was used as the programming platform for the students, a support tool for the TAs, and a monitoring and sharing tool for the instructor.

The second use of Spinoza was in smaller optional classes called Advanced Recitations (ARs). These typically consisted of 10-20 students who needed more help understanding the concepts being taught that week. In these classes, students worked in pairs on Spinoza Exercise or Homework problems, sharing a single computer (and alternating whose computer was used). The main difference between the use of Spinoza in Advanced Recitations, compared to its use in class, is that the student pairs were allowed to work until they solved the problem. In class, we typically waited until 50% had attempted a solution with no syntax errors and then entered the pair phase, encouraging them to talk to their neighbors. When 75% had attempted a solution with no syntax errors we switched to the Share phase and discussed and debugged, as a class, the most common attempts. Students, who had not completed the problem by that time, were shown their classmates' solutions, but they didn't get to discover the solution for themselves. This can lead to frustration for those students who are having difficulties understanding the material, especially if it occurs repeatedly. In the ARs students worked, in pairs, on one problem until they completed it. If they got stuck, they could get help from a TA if needed. When they completed their problem, they shared their solutions with a TA who then picked another Spinoza problem for them, ideally in their zone of proximal development (Vygotsky, 1987). We have some evidence that suggests this is effective for at-risk students.

The third use of Spinoza was as an auto-grading homework system. We created a set of Spinoza Homework problems and asked students to complete the problems before the next class. The main differences between Spinoza and other auto-grading systems [(Sherman et al., 2013), (Gotel et al., 2007)] are that students receive immediate feedback based on the unit tests and the problems they were given could be either highly prescribed or fully inferential as with Code Hunt (Tillmann et al., 2014), i.e. figure out what the program does by comparing the expected and actual results of the unit tests and write the method body to produce those results. One consequence of providing instantaneous feedback via the unit tests in this Test Driven Development model is that almost all students were able to achieve 100% accuracy in their homework submissions.

4. Data Collected and Discussion

The one semester course was composed of four units. Each unit culminated in a Quiz and a student survey. The course grades, survey answers, and a summary of the Spinoza interaction data for each student were all compiled into a single database for analysis.

4. 1. Students Find Spinoza Helpful

Students generally felt that Spinoza was a helpful tool. When asked to rank how well they felt they learned from Spinoza on a Likert scale from 1 (very little) to 5 (a lot), only 18% rated it as 1 or 2. Figure 3 shows the histogram of responses. The open responses also demonstrated that students appreciated the immediate feedback from the Spinoza Homework and the sharing made possible with the Spinoza Exercise mode.

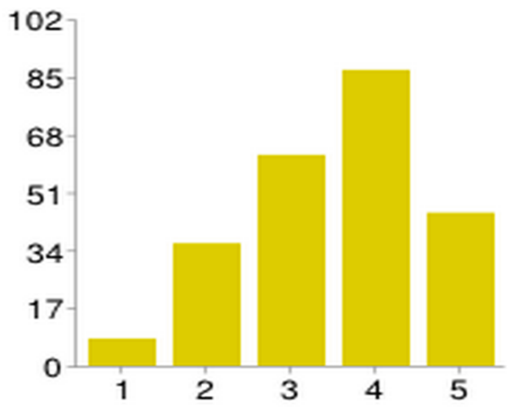


Fig. 3. Responses to the Question "How well do you learn from Spinoza lectures in this class" from 1(very little) to 5(a lot).

4. 2. Finding Spinoza Helpful Is Positively Correlated With Course Grade

We were also interested in students' self-assessment of how well they learn using Spinoza and whether that had any correlation with their course grade. The image on the right side of Figure 4 shows a scatter plot where the horizontal axis is their self-assessment of how well they learned using Spinoza ranging from 1 (not helpful) to 5 (very helpful) and the vertical axis is their final grade. The fitted line shows positive correlation with r(154)=.17 at p=.03 (p<.05). The low r value indicate that there is a weak relationship between the two variables (finding Spinoza helpful and course grade). We plan to run a larger study to further investigate this pattern.

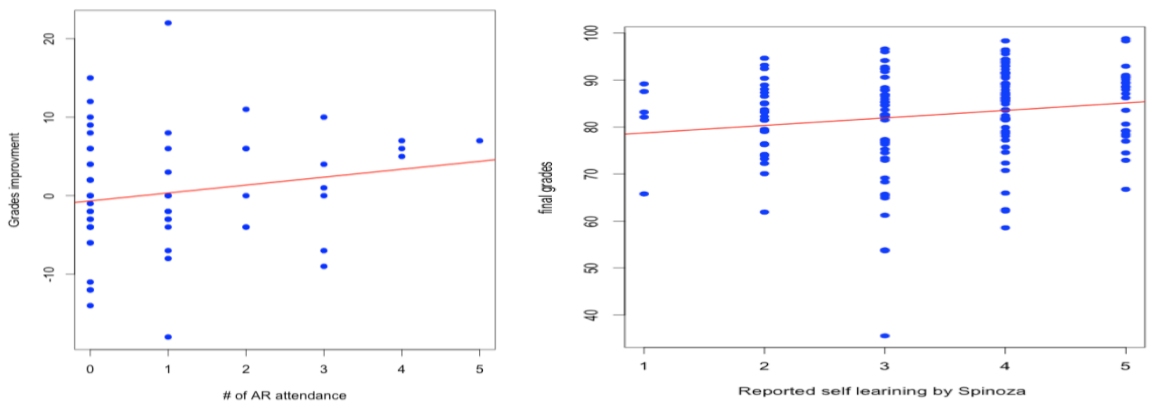


Fig. 4. On the left is a scatterplot of the improvement from Quiz 3 to Quiz 4 as a function of the number of AR sessions the student attended in Unit 4. On the right is a plot of the total course grade (before curving) in terms of students' self-assessment of how well they learn from Spinoza from 1 (very little) to 5 (a lot).

4. 3. Advanced Recitation participation with Spinoza seems to improve quiz scores

The Advanced Recitations were not started until after Unit 3 in the course when it was observed that a significant number of students did poorly on Quiz 3. We asked students who received less than 30/50 in Quiz 3 to attend Advanced Recitations as described in the previous section. The Advanced Recitation could have been run with another text editor or IDE, but Spinoza provides more scaffolding and better tools for monitoring student progress, so we required students who attended to use Spinoza. The data (on the left side of Figure 4) shows that the number of Advanced Recitations that students attended during

Unit 4 was positively correlated with their improvement (r(59)=.18 at p=.16)in their Quiz grades from Quiz 3 to Quiz 4. This is not a statistically significant result, so we cannot infer from this data that attending more ARs improved students' grades, but it does suggest further study is warranted.

5. Markov Models of the Average Student

Every time a student presses the run button while trying to solve a Spinoza problem, a time-stamped version of their code is stored in a database. When we gather together all of this data for the entire class, it allows us to create a Markov Model of how students in the course attempt to solve the problem. The nodes of the Markov Model are the versions of programs submitted by students. Each node is labeled with the number of times that program was submitted by the class and each edge (from A to B) is labeled with the number of students that submitted program A and then made one edit and submitted program B. If we divide the edge label by the node count, we get the probability that a student in the class would progress from program A to program B.

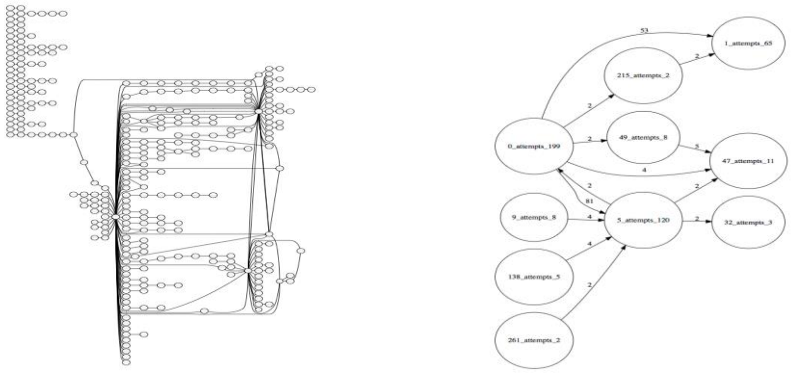


Fig. 6. On the left is the full Markov Model of the average student working on an inductive Spinoza Homework to write a method that returns the product of its three integer parameters, given only the unit tests. On the right is th subset of that model consisting of edges followed by at least two students.

The instructor can use Spinoza to download a directed graph representing this Markov Model for any Spinoza Homework or Exercise problem and we are just now beginning to look into ways that this model can be used to investigate how students in the course are learning to program. The image on the left of Figure 5 shows the Markov model for all students in the class attempting to write the body of a method with integer inputs given only the unit tests. The instructor's solution simply returns the product of the three inputs.

A surprising feature of this graph is that there are so many different attempted solutions and there are only three solutions that act as hubs with a large number of students making those attempts. Programs are put into a single cluster if they have the same Java class file (i.e. we ignore white space, comments, variable names). The reason there are so many different solutions in this case is because of the variation in the code students used to test hypotheses about what was producing the expected output of the unit tests. There are also several subgraphs that don't reach a solution representing students that tried and gave up.

The image on the right of Figure 5 shows the graph obtained by only looking at nodes which are connected to another node by an edge with a value of 2 or more. These represent consecutive versions of a program that were tried by at least two students. This is a much smaller graph and we see that there are only a few edges with large counts. Node **0_attempts_199** is the starting code and 199 students tried to evaluate this code at least once. There are two neighbors of this node: **5_attempts_120** which was reached in one step by 81/120 students, and **1_attempts_65** which is a slight variation and was reached in one step from node 0 by 53/65 students. We haven't yet tried to see if the particular paths students take can be predicted based on other known features (e.g. quiz grades). Although this is a simple example, it suggests that we may be able to gain a deeper understanding of how students are approaching and solving problems in CS1 classes using this kind of a Markov Model. We are currently working on enhancing this visualization of student progress and exploring different possible pedagogical uses of this information.

6. Related Work

Spinoza differs from other web-based IDEs that we describe below, primarily in that it was designed as a platform to support active learning in a flipped (or partially flipped) classroom. This design goal resulted in several design decisions that, taken together, differentiate it from other web-based IDEs available today.

- It provides a simple IDE with java syntax color coding and requires students to write the whole program rather than one method.
- It allows the instructor to create programing challenges on-the-fly in class promoting a more spontaneous and nimble approach to active learning in a programming class
- The instructor can provide some initial code that can include everything except for one small part of the program, or no initial code at all. This allows the instructor to choose the level of scaffolding to provide for this particular exercise.
- It generates random tests that differ each time the program runs to test student's inductive skills and none of the tests are hidden. Other systems, such as codingbat (web-1), have a fixed set of tests, but some are hidden to prevent students taking a brute force approach to solving the problem (where you check for the finite set of inputs and return the specified output with no computation!)
- It has a dashboard that allows the instructor to see the progress of everyone in the class in one screen in real-time
- It allows the instructor to view and modify the code of any student in the class without changing the students' view of their code.

Spinoza is a descendant of codingbat (web-1), a web-based Java IDE that was developed to provide students immediate feedback to problems in which students are writing the body of single method in a textbox. Codingbat generates a fixed set of tests and shows the student the result of (some of) these tests when the student presses the run button. Codingbat also keeps track of the student's history with a problem by producing a time-based graph showing each time the student tried to run the program and the percentage of unit tests that were successful. Students are able to specify a teacher who will have access to these graphs and who can track their progress.

Code Hunt (Tillmann et al., 2014) is another web-based IDE (from Microsoft Research) that uses unit tests in a web based game to give players an opportunity to exercise their inductive skills. Players are given the results of a set of unit tests (calculated result vs expected result) and are asked to write code that will pass all the unit tests without knowing the problem description. Spinoza can be used to generate this type of challenge but it does not provide the game-like interface of Code Hunt. Code Hunt, in turn, lacks the dashboard view of Spinoza.

CloudCoder (Papancea et al., 2013), BottleNose (Sherman et al., 2013) and webwork-JAG (Gotel, Scharff, & Wildenberg, 2007) are web-based platforms for creating programming exercises for a variety of languages, such as Java, C and Python. These systems allow the instructor to create (and share) programming problems with codingbat style unit tests, but they don't provide the same instructor-support as Spinoza with a dashboard and the ability to track every step in a student's effort to solve a problem.

Informa (Hauswirth & Adamoli, 2009) is an Iclicker-style desktop application designed for in class activity in a flipped classroom, just like Spinoza, but it is not a web-based application and needs to be installed in every student's computer. Moreover, students will use the tool to write a proposed solution to the given java coding problems in the text editor and the solution will be sent to Instructor app with they click the submit button, but the app does not run student's code so students don't get immediate feedback.

7. Conclusions and Future Work

Our experience demonstrates that Spinoza can be an effective tool for flipping a CS1 introductory programming class. It supports active learning of coding in very large classes while providing the instructor with a deep and nuanced view of student performance, both individually and as a class. The Markov Models that are constructed based on student interaction with the tool on particular problems provide a deeper understanding of student problem solving behavior than can be obtained by traditional homework practices, which typically require just the final program to be submitted.

We expect that tools similar to the Spinoza Code Tutor will become widespread in introductory programming classes since they provide both a highly flexible and easy to use platform for instructors and students, and they also provide a rich and nuanced view of student coding skills that includes their entire history of working on a problem in the context of all other students in the class. These features are valuable in both small and large programming classes.

References

Gotel, O., Scharff, C., & Wildenberg, A. (2007). Extending and Contributing to an Open Source Web-Based System for the Assessment of Programming Problems. NY.

Hauswirth, M., & Adamoli, A. (2009). Solve & evaluate with informa: a Java-based classroom response system for teaching Java.

Lyman, F. (1987). Think-pair-share: An expanding teaching technique.

Papancea, A., Spacco, J., & Hovemeyer, D. (2013). An Open Platform for Managing Short Programming Exercises.

Sherman, M., Bassil, S., Lipman, D., Tuck, N., & Martin, F. (2013). Impact of auto-grading on an introductory computing course. *Consortium for Computing Sciences in Colleges* (pp. 69-75). USA: Journal of Computing Sciences in College.

Tillmann, N., de Halleux, J., Xie, T., & Bishop, J. (2014). Code Hunt: Gamifying Teaching and Learning of Computer Science at Scale. *Learning @ Scale Conference, L@S* (p. 2). NY: L&S.

Vygotsky, L. (1987). Zone of proximal development.

Web Sites:
Web-1: http://codingbat.com
Web-2: https://ideone.com
Web-3: http://www.compilejava.net
Web-4: http://repl.it/languages