An Evaluation of Interactive Test-Driven Labs with WebIDE in CS0

David S. Janzen, John Clements, Michael Hilton
California Polytechnic State University
San Luis Obispo, California 93407
{djanzen,clements,hilton}@calpoly.edu

Abstract—WebIDE is a framework that enables instructors to
develop and deliver online lab content with interactive feedback.
The ability to create lock-step labs enables the instructor to guide
students through learning experiences, demonstrating mastery as
they proceed. Feedback is provided through automated evalu-
ators that vary from simple regular expression evaluation to
syntactic parsers to applications that compile and run programs
and unit tests. This paper describes WebIDE and its use in a CS0
course that taught introductory Java and Android programming
using a test-driven learning approach. We report results from
a controlled experiment that compared the use of dynamic
WebIDE labs with more traditional static programming labs.
Despite weaker performance on pre-study assessments, students
who used WebIDE performed two to twelve percent better on
all assessments than the students who used traditional labs. In
addition, WebIDE students were consistently more positive about
their experience in CS0.

I. INTRODUCTION

WebIDE is a framework for creating a variety of labs that
provide rich, rapid feedback to students as they learn new
concepts and gain or practice new skills. WebIDE provides a
scalable, open, and distributed infrastructure for lab authors
to create, host, and render labs, using their own or provided
automated evaluators. Figures 1 and 2 give examples of steps
in existing WebIDE labs. Lab authors can establish depen-
dencies between steps, requiring that students successfully
complete a step before proceeding in the lab. Alternatively,
we have set up WebIDE labs as a programming playground
with a single step containing editor windows for writing
code and corresponding unit tests, enabling a student to write
simple programs completely in the web, even with compiled
languages like C or Java.

At least twenty-four labs along with at least fourteen auto-
mated evaluators have already been created for introductory
programming students learning C, Java, Android, Python, and
Ruby. Instructors can write their own labs (or modify existing
labs) using the WebIDE XML specification, then host the labs
on their own web sites, pointing WebIDE to the lab files
in order to parse, render, and run their own labs. Similarly,
instructors can use provided automated evaluators to process
student submissions and provide meaningful feedback in their
labs, or they can write their own automated evaluators and
host them on any internet-connected server. A video demo
of WebIDE is available at http://www.web-ide.org/demo, and
links to WebIDE labs organized into courses are available at

WebIDE was developed and evaluated through an NSF
CCLI Type 1 award with additional support from Google and
Amazon. This paper will highlight key aspects of WebIDE
including its support for Test-Driven Learning, along with
its unique architecture for incorporating automated evaluators.
Evaluation results from a semi-controlled experiment will be
reported.

II. TEST-DRIVEN LEARNING

A key component of WebIDE’s architecture is Test-Driven
Learning (TDL) [24]. Closely related to Test-Driven Devel-
opment [6] (TDD), TDL uses the construction of test cases
to drive the learning process. In particular, TDL observes that
students who write small test cases before the corresponding
implementation tend to understand their goal before getting
tangled up in code. TDL-based WebIDE labs require the stu-
dents to write examples and/or test cases, and check these test
cases against instructor code, before students tackle the more
bulky task of writing the code itself. Figure 2 demonstrates an
example of this approach.

Currently, TDD is becoming a widespread software engi-
neering best practice. Previous studies indicate benefits from
applying TDD, but note challenges of actually getting fledgling
programmers to write code in a test-first manner[25]. Prior
studies have shown that TDL can be applied in entry level
classes without removing current course material, and that
students who try TDL, tend to like TDL[24]. However, prior
to WebIDE, finding ways to enforce a test-driven approach
with beginning programmers has proven to be elusive [9].

The inspiration for WebIDE was the desire to require
students to demonstrate understanding of a problem by writing
examples and tests, prior to solving a problem. The goal
was to instill software engineering best practices from the
beginning of learning to program. Several secondary benefits
were quickly discovered with WebIDE. In particular, WebIDE
provided the opportunity to point students to a super-simple,
web-only programming environment on day one. Students
are able to delay the learning curve associated with editors,
compilers, and integrated development environments, and con-
centrate on programming and problem solving from the very
beginning.

WebIDE is not restricted to TDL, or even to computer pro-
gramming for that matter. WebIDE provides an infrastructure
designed so that anyone can create new or modify existing
laboratory exercises and evaluators—written in any language and providing customized error messages—that help teach a wide range of concepts, languages, or software engineering techniques.

III. WEBIDE ARCHITECTURE

WebIDE uses Google Web Toolkit (GWT) and is currently deployed on Amazon’s EC2 cloud platform, although it has also been hosted on Google’s App Engine in the past. The labs and automated evaluators can be located on any web-accessible host.

Figure 3 illustrates the general architecture of the system. The solid lines represent HTTP connections. The dashed line indicates an implicit dependency; specifically, the lab supplied by the Lab Source embeds within it URLs that allow WebIDE to locate the evaluators.

The WebIDE architecture is focused on extensibility. Lab specifications are written in a well-defined XML language, so that labs may be edited and contributed by third parties. Additionally, the presentation of the lab is completely decoupled from the evaluator using a service-oriented architecture (SOA) where URLs identify evaluators.

To be more concrete, suppose an instructor wants to create a lab on FORTRAN/77. The instructor first formulates the text of the lab, using an XHTML subset. Then, (unless there’s an existing FORTRAN/77 evaluator floating around), the instructor writes a script that couples a FORTRAN evaluator with the given submission and reports the result. Finally, the instructor embeds the URL of the evaluator within the lab itself. Later FORTRAN labs can re-use the instructor’s existing evaluator.

In order to enforce structure and prevent ad-hoc extension, labs are written using a XML language defined using the Relax NG specification language. So, for instance, the lab is specified to contain a name, an optional description, and zero or more

---

**Fig. 1.** WebIDE lab with sample error feedback

**Fig. 2.** Students create JUnit tests in WebIDE to demonstrate problem understanding

**Fig. 3.** WebIDE Architecture
V. AUTOMATED EVALUATORS

In the WebIDE framework, students provide answers to questions in web forms. Each answer is then sent to one or more “evaluators” to provide feedback. Effective and helpful feedback is the linchpin of a successful WebIDE interaction, and a variety of evaluators and evaluator families for use in different situations are already provided. The WebIDE framework accommodates the use of multiple evaluators for a single question box. Evaluators can reside on any server, but the provided evaluators are currently hosted on Amazon EC2 instances behind a load balancer, providing flexibility for handling inconsistent demand. We present two general categories of automated evaluators: syntactic and semantic.

A. Syntactic Evaluators

The “syntactic evaluators” are those that evaluate student code at a syntactic level, by considering it either as a stream of characters or as an abstract syntax tree. The simplest evaluators are those that simply perform textual comparison. Regular expressions provide a reasonably flexible way of performing such comparisons, and WebIDE includes a regexp-based evaluator. To take the simplest possible example, a lab asking a student to enter a positive integer might deliver the student’s answer to the regexp evaluator matching the PERL regex 

```perl
\^s*[1-9]\d*\$
```

For more complex problems, a typical syntactic evaluator is one based on a parser for the given language. The WebIDE team has developed a syntactic evaluator for C that parses student input and compares the resulting tree to one obtained by parsing an expected answer. This kind of evaluator has at least three advantages over one based on simple textual comparison. First, it has the advantage that it is insensitive to whitespace, the presence or absence of optional delimiters and parentheses, etc. Second, it can provide source code positions and source code highlighting along with its errors. Finally, it can abstract over certain observational equivalences; for instance, for terms A and B, the programs A + B and B + A may be considered equivalent. This assumes the language in question is purely functional, or is restricted to a purely functional subset, or that the sub-programs A and B can be statically analyzed to ensure they do not have side effects.

B. Semantic Evaluators

The “semantic evaluators” are those that consider the evaluated meaning of the student code, rather than its syntactic form. For instance, suppose a student is asked to produce a function called `smaller` that accepts two numbers and returns the smaller of the two. A semantic evaluator would evaluate the student code and then apply it to one or more test inputs, verifying that it produces the expected output. The WebIDE team has produced semantic evaluators for a variety of Java problems, and general purpose semantic evaluators for C, Java, Python, and Ruby.

Semantic evaluators have advantages and disadvantages. On the one hand, they are more robust, in the sense that they can accept answers that a syntactic evaluator would not. On
The first two tests are completed for you. Your job is to write testfToC2 and testcToF2.

```java
class TempConverter {
    public static long fToC(int f) {
        // replace this comment with the body of the fToC method
    }
    // replace this comment with the cToF method
}
```

Now write the code to make the tests above pass. Be sure to define everything including the class with both methods.

Click the button at the bottom to run your tests above on your code below.
the other hand, regarded as provers of correctness, they are unsound; the halting problem demonstrates that no evaluator can decide for all problems whether an arbitrary program behaves correctly on all inputs.

An important subcategory of semantic evaluators are those that examine student-written test cases. In this case, the student’s test case is run against the lab-writer’s function. Test cases that do not correctly predict the output of the function do not succeed. More interestingly, a set of student test cases can also be evaluated for coverage; do they exercise all of the instructor’s code?

VI. EVALUATION

Three studies were conducted to evaluate WebIDE at Cal Poly - San Luis Obispo. In all three studies, two sections of the same course were taught by the same instructor. Approximately seventy students were divided between two sections. One section was randomly selected as the control group, and the other as the experimental group. Both sections were given weekly labs that contained identical content. The only difference was that the experimental group completed the labs in WebIDE, using the lock-step framework and incremental feedback. The control group used traditional development environments (e.g. Eclipse, vim/gcc).

The first two studies were pilot studies conducted in Fall 2010 and Spring 2011 while WebIDE was still being developed and was still a bit unstable. Preliminary results reported in May 2011 [15] that the WebIDE group showed a significant improvement in performance when writing a simple Android application. Additionally, among students with some programming experience, the WebIDE group was more proficient in writing unit tests.

The most complete evaluation of WebIDE was conducted in Fall 2011 in an Introduction to Computing (CS0) [20] course that taught beginning programmers how to build Android apps. Seventy-two students participated in the study in two sections of the same course. The course contained nine labs. The first three labs used visual drag-and-drop programming environments (Scratch and App Inventor). The fourth, fifth, sixth, and eighth labs introduced Java programming concepts.
(data types, variables, operations, methods, classes, selection, loops, arrays). The seventh and ninth labs introduced Android. Table I summarizes the topics covered in the nine labs. Labs four through nine are highlighted to indicate the use of WebIDE and focus of this study. A more thorough analysis of just the arrays lab was presented by Hilton et al [21].

A. Evaluation of Impacts on Student Programming Skills

Students were given a pre-study quiz to assess their programming skills. Table II presents the pre-study quiz questions. The WebIDE section had weaker programming skills coming into the course (see Java PreQuiz in Figure 5 and Table III), and they performed worse on the initial Scratch and App Inventor labs (prior to WebIDE). Just prior to lab four, a coin was flipped to determine which section would be control/experimental, without having analyzed results on the Java pre-quiz or the first three labs. All students were given the same lectures by the same instructor. The lectures included instruction and exercises on Java, Android, and using the Eclipse IDE. Although there may be better introductory development environments, Eclipse was chosen because of its strong support for Android development which all students would need in their team projects. The WebIDE section students then completed their labs using WebIDE. The control group completed identical labs using Eclipse. The control group was provided with shell programs that contained comments indicating where the students were to enter their code as instructed in the labs.

Labs four through eight included online quizzes (using Blackboard http://www.blackboard.com/) at the end of closed lab sessions. Figure 5 shows the average percentage score on the assessments. Figure 6 visualizes this same data a bit differently, showing the percentage differences on these quiz assessments. Students who used WebIDE scored between 2.51% and 12.19% better on all assessments after WebIDE was introduced.

Table III reports additional detail on student performance in the controlled experiment. In particular, scores are reported on computer programming assessments, demonstrating the weaker performance on the two pre-WebIDE assessments, and consistently 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 Java pre-study quiz.

B. Evaluation of Impacts on Student Opinions

Students were asked their opinions on the course, the labs, and their perspectives on their chosen computing major in a post-experiment on-line survey. Table IV reports these results based on a five point Likert scale with one being “Strongly Agree” and five being “Strongly Disagree.” Smaller averages indicate more positive responses. The WebIDE students consistently were more positive about the labs and their major than the control group. Statistical significance was determined with \( p < .05 \) using a Two sample t-test. A Wilcoxon rank-sum test was also run on the first eight questions, identifying statistical significance with rows 2, 3, and 8.

Table V reports student responses to questions specifically about the IDE that they used in their labs (WebIDE or Eclipse). The numbers reported reflect the percent of students who answered 1-Strongly Agree or 2-Agree to the corresponding questions. Although the Eclipse users liked their IDE a bit more, the WebIDE users reported much stronger agreement on the positive affect of the IDE on their learning and its ability to provide helpful error messages. Based on student comments in free-response forms, there are still many improvements we can make to WebIDE labs. In particular, some students indicated frustration with the lock-step nature of the labs. If they got stuck, they couldn’t go on until they got help (is this a good thing?). Other students wished that they could see the entire contents of their solution at the end of a lab, thus giving better context. This can be easily incorporated, and is under the purview of the lab author.

C. Threats to Validity

Like many academic studies, this evaluation is susceptible to several validity threats such as external factors that may have influenced students in the two groups (e.g. friendships formed...
### TABLE I
LAB TOPICS; * INDICATES USED WEBIDE

<table>
<thead>
<tr>
<th>Lab #</th>
<th>Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Programming with Scratch, a visual programming environment for creating 2D animations, Control structures, event-driven programming, concurrency, variables.</td>
</tr>
<tr>
<td>2</td>
<td>Programming with App Inventor for Android, a visual programming environment for creating Android apps, Visual screen components (e.g. buttons, labels, images), click listeners, media components (e.g. audio), 2D drawing on a canvas.</td>
</tr>
<tr>
<td>3</td>
<td>More App Inventor for Android, Creating apps with multiple screens, integrating Google Maps, launching activities, creating procedures.</td>
</tr>
<tr>
<td>4*</td>
<td>Java ints and operators, Built-in Java data types, corresponding operators, declaring variables, Control flow, selection control structures: if-then, if-then-else, switch, Conditions, comparison and boolean operators, Methods/functions, parameters, return types, Testing functions with examples and JUnit.</td>
</tr>
<tr>
<td>5*</td>
<td>Java classes, Static methods, instance methods, constructors, Testing methods in classes with examples and JUnit.</td>
</tr>
<tr>
<td>6*</td>
<td>Java looping control structures, Testing loops with examples and JUnit, While loops, for loops, nested loops, loops with nested selection.</td>
</tr>
<tr>
<td>7*</td>
<td>Android basics, creating Hello World and Rock-Paper-Scissors game, Android project structure, manifest file, xml layout files, xml string files, Android Activity class, application lifecycle, onCreate method, attaching a layout to an Activity, Launching another Activity using intents, passing parameters to another Activity using Bundle.</td>
</tr>
<tr>
<td>8*</td>
<td>Java arrays Declaring and initializing arrays, referencing array elements, out of bounds exception, Looping through arrays, passing arrays to methods as parameters, Arrays of objects, calling methods on objects in an array.</td>
</tr>
<tr>
<td>9*</td>
<td>More Android, creating a Tic Tac Toe game, Designing a user interface, using a GridView layout, Designing an app with non-UI classes, keeping track of cell state and game state, implementing BaseAdapter.</td>
</tr>
</tbody>
</table>

### TABLE II
PRE-STUDY QUIZ QUESTIONS

<table>
<thead>
<tr>
<th>Question #</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Write a method/function that receives six parameters representing student scores in a course, and returns the student’s weighted grade as a real number between 0 and 100. Each input is a real number between 0 and 100. The inputs/parameters are: labs, project1, project2, participation, midterm, final The weightings are: labs: 25%, project1: 15%, project2: 25%, participation: 10%, midterm: 10%, final: 15%</td>
</tr>
<tr>
<td>2</td>
<td>Write a Java class that represents a mobile device. Each device has three attributes: a name, price, and screen width in inches. Include a constructor that accepts all three attributes. Include a function/method that returns the cost per inch (price/width).</td>
</tr>
<tr>
<td>3</td>
<td>Write a JUnit test that constructs one instance of the class defined above, and tests that the cost per inch method works correctly.</td>
</tr>
</tbody>
</table>

within the class), and differences in the meeting times of the two sections. One particular threat to validity should be noted. In the final Android Tic Tac Toe lab (Lab 9), a large number of students in the control group discovered and used the WebIDE lab when they ran into challenges with completing the lab in Eclipse. When questioned, all students reported that this was their first use of WebIDE.

**D. Anecdote**

In addition to the evaluated courses, WebIDE is currently being used in an on-line commercial course at Udemy. Students from the Fall 2011 CS0 course were invited to enroll in the Udemy course for free while they were taking a Spring CS2 course which also used Java. Although the percentage of female students in the Fall 2011 CS0 course was only 19% (14 out of 72), 64% (9 out of 14) of them signed up for the Udemy course when it was offered as a free supplement to their CS2 course. This compares to only 38% (22 out of 58) of the male CS0 students who signed up for the Udemy course. It would appear that female students are more likely to take advantage of online resources like WebIDE.

### VII. RELATED WORK

A variety of online systems support development and evaluation of programs in a variety of languages [13], [19], [2], [10], [1], [27], [18], [39], [32], [26]. These systems are intended for developers, and do not operate as automated tutors.

Separately, automated tutors exist for a variety of academic fields. Samples include Biology/Genetics [33], Mathematics [23], and Physics [38]. Many of these tools have been evaluated, with promising results. For instance, Warnakulasooriya et al. [37] reports that their web-based automated Physics tutor improves student time to completion, reduces the need for hints, and improves the number of correct answers all by approximately 15%.

Not surprisingly, computing faculty and researchers have also built many software tools to support students as they learn
TABLE III
STUDENT PERFORMANCE ON PROGRAMMING ASSESSMENTS

<table>
<thead>
<tr>
<th>Assessment</th>
<th>Total Possible</th>
<th>WebIDE Mean</th>
<th>Control Mean</th>
<th>Significant?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Pre-Study Quiz</td>
<td>25</td>
<td>2.72</td>
<td>3.50</td>
<td>No</td>
</tr>
<tr>
<td>Midterm Exam: App Inventor</td>
<td>20</td>
<td>14.56</td>
<td>14.92</td>
<td>No</td>
</tr>
<tr>
<td>Lab 4 Quiz: Java Basics, If, Methods</td>
<td>20</td>
<td>17.89</td>
<td>17.14</td>
<td>Yes</td>
</tr>
<tr>
<td>Midterm Exam: Java if</td>
<td>15</td>
<td>13.47</td>
<td>13.31</td>
<td>No</td>
</tr>
<tr>
<td>Midterm Exam: Java methods</td>
<td>20</td>
<td>17.89</td>
<td>16.50</td>
<td>Yes</td>
</tr>
<tr>
<td>Lab 5 Quiz: Java Methods</td>
<td>26</td>
<td>18.43</td>
<td>17.78</td>
<td>No</td>
</tr>
<tr>
<td>Lab 6 Quiz: Java Loops</td>
<td>12</td>
<td>8.11</td>
<td>7.29</td>
<td>Yes</td>
</tr>
<tr>
<td>Lab 7 Quiz: Android Basics</td>
<td>10</td>
<td>7.89</td>
<td>6.67</td>
<td>Yes</td>
</tr>
<tr>
<td>Lab 8 Quiz: Java Arrays</td>
<td>22</td>
<td>13.75</td>
<td>11.44</td>
<td>Yes</td>
</tr>
<tr>
<td>Comp. Lab Quiz: Java Arrays/Loops</td>
<td>12</td>
<td>10.17</td>
<td>8.75</td>
<td>Yes</td>
</tr>
<tr>
<td>Lab 5 Quiz: Android</td>
<td>14</td>
<td>6.47</td>
<td>5.08</td>
<td>Yes</td>
</tr>
<tr>
<td>Final Exam</td>
<td>210</td>
<td>180.81</td>
<td>173.17</td>
<td>Yes</td>
</tr>
<tr>
<td>Final Exam: Java Class</td>
<td>12</td>
<td>11.08</td>
<td>10.08</td>
<td>Yes</td>
</tr>
<tr>
<td>Final Exam: JUnit Test</td>
<td>6</td>
<td>4.72</td>
<td>4.50</td>
<td>No</td>
</tr>
<tr>
<td>Final Exam: Android XML</td>
<td>8</td>
<td>7.47</td>
<td>7.36</td>
<td>No</td>
</tr>
<tr>
<td>Final Exam: Android Activity</td>
<td>12</td>
<td>8.83</td>
<td>7.97</td>
<td>No</td>
</tr>
</tbody>
</table>

TABLE IV
STUDENT OPINIONS

<table>
<thead>
<tr>
<th>Question</th>
<th>WebIDE Mean</th>
<th>Control Mean</th>
<th>Significant?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confident will complete major</td>
<td>1.28</td>
<td>1.61</td>
<td>Yes</td>
</tr>
<tr>
<td>Course stimulated interest</td>
<td>1.66</td>
<td>1.92</td>
<td>No</td>
</tr>
<tr>
<td>Computing is exciting</td>
<td>1.59</td>
<td>2.00</td>
<td>Yes</td>
</tr>
<tr>
<td>I liked this course</td>
<td>1.59</td>
<td>2.03</td>
<td>Yes</td>
</tr>
<tr>
<td>I am comfortable with Java</td>
<td>2.38</td>
<td>2.47</td>
<td>No</td>
</tr>
<tr>
<td>I am comfortable with Android</td>
<td>3.03</td>
<td>3.25</td>
<td>No</td>
</tr>
<tr>
<td>Labs helped learn Java</td>
<td>2.21</td>
<td>2.64</td>
<td>Yes</td>
</tr>
<tr>
<td>Labs helped learn Android</td>
<td>2.09</td>
<td>2.72</td>
<td>Yes</td>
</tr>
<tr>
<td>Liked Lab 4</td>
<td>2.03</td>
<td>2.31</td>
<td>No</td>
</tr>
<tr>
<td>Liked Lab 5</td>
<td>2.34</td>
<td>2.56</td>
<td>No</td>
</tr>
<tr>
<td>Liked Lab 6</td>
<td>2.22</td>
<td>2.36</td>
<td>No</td>
</tr>
<tr>
<td>Liked Lab 7</td>
<td>2.66</td>
<td>3.44</td>
<td>Yes</td>
</tr>
<tr>
<td>Liked Lab 8</td>
<td>2.34</td>
<td>2.81</td>
<td>Yes</td>
</tr>
<tr>
<td>Liked Lab 9</td>
<td>1.44</td>
<td>1.72</td>
<td>No</td>
</tr>
</tbody>
</table>

TABLE V
STUDENT OPINIONS OF IDE

<table>
<thead>
<tr>
<th>Question</th>
<th>WebIDE Percent Positive</th>
<th>Control Percent Positive</th>
</tr>
</thead>
<tbody>
<tr>
<td>I liked my IDE</td>
<td>65.7%</td>
<td>69.4%</td>
</tr>
<tr>
<td>My IDE helped me learn Java</td>
<td>84.4%</td>
<td>69.5%</td>
</tr>
<tr>
<td>My IDE helped me learn Android</td>
<td>68.8%</td>
<td>58.3%</td>
</tr>
<tr>
<td>My IDE's failure messages were helpful</td>
<td>59.4%</td>
<td>44.4%</td>
</tr>
<tr>
<td>My IDE's failure messages were confusing</td>
<td>62.5%</td>
<td>66.7%</td>
</tr>
</tbody>
</table>

to program. Valentine[36] reports that 22% of the CS1/CS2-related SIGCSE conference papers from 1984 to 2003 included software tools to aid learning. Some of the more popular tools include visualizations[22], Karel micro-worlds[8], [7], automated assessment tools[14], and pedagogical development environments such as DrRacket[17], Alice [12], and Scratch [31].


Aleven et al.’s work with CTAT [3] highlights the common struggle with reducing the effort to author labs for intelligent tutoring systems. Parlante’s CodingBat [30] adopts a test-based approach, although students do not write tests and the system is limited to a set of small, focused exercises. Edwards’ WebCAT[16] web-based automated grading tool assumes student
creation of automated (presumably test-driven) unit tests, but it provides no support for interactive labs.

VIII. Future Work

We are continuing to improve and use WebIDE. Current efforts are underway to improve the WebIDE user interface and to add some gamification aspects. The user base for WebIDE is expanding, and we look forward to seeing community contributions of labs and automated evaluators.

IX. Conclusions

WebIDE is unique in its combination of features: a TDL approach, completely web-based delivery, and intrinsic support for community-contributed content. WebIDE enables lab authors to create their own labs or use existing labs, and to give specific student feedback on individual exercises. WebIDE is flexible, scalable, and robust.

Classroom experiments indicate that students learn early programming concepts better with WebIDE than using traditional static labs with traditional IDE’s. Furthermore, students report more positive experiences in learning to program with WebIDE over more traditional development environments.

Acknowledgments

The authors are grateful to Aaron Keen and Gene Fisher for their comments on an earlier version of this paper, and to Olga Dekhtyar for her assistance with the statistical analysis.

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.

We are also grateful to Amazon Web Services and Google for web services and lab development support.

References