Supporting Introductory Test-Driven Labs with WebIDE

Thomas Dvornik

David S. Janzen, John Clements, Olga Dekhtyar

Abstract

WebIDE is a new web-based development environment for entry-level programmers with two primary goals: minimize tool barriers to writing computer programs and introduce software engineering best practices early in a student's educational career. Currently, WebIDE focuses on Test-Driven Learning (TDL) by using small iterative examples and introducing lock-step labs, which prevent the student from moving forward until they finish the current step. However, WebIDE does not require that labs follow TDL. Instructors can write their own labs for WebIDE using any software engineering or pedagogical approach. Likewise, instructors can build custom evaluators—written in any language—to support their approach and provide detailed error messages to students. We report on a pilot study in a CS0 course where students were split into two groups, one that used WebIDE and one that didn't. 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.

1 Introduction

Students often struggle with the first few weeks of beginning computer science courses. In addition to learning programming concepts and syntax, students typically work in an unfamiliar computing environment, whether it be an integrated development environment (IDE) or a text editor with a command-line compiler. We have observed students quickly fall behind in class material due to barriers with tools (e.g. Unix and editor commands, installing IDEs on personal computers), sometimes to the point where they are unable to catch up for the remainder of the course.

1

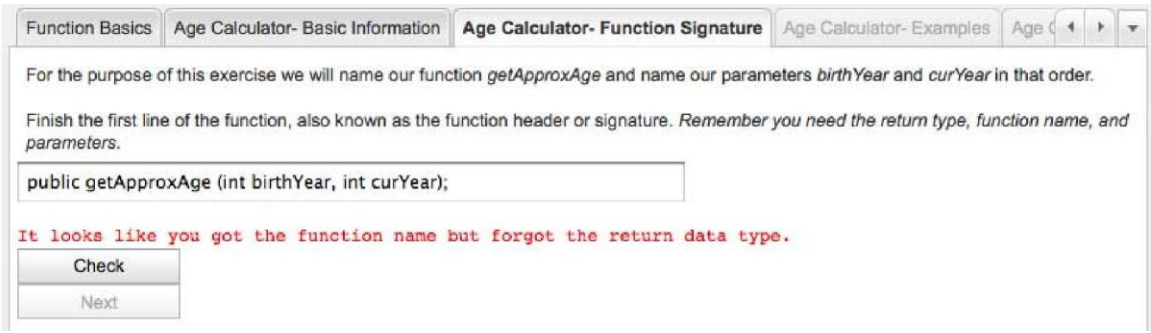

Figure 1. WebIDE lab with sample error feedback

WebIDE helps students during these difficult early weeks by offering a one-button interface in a ubiquitous and familiar context (a web browser). Figure 1 demonstrates a step in a WebIDE lab. In this case, the student is asked to type the header for a Java method, but they forgot the return type; an appropriate error message is shown in red. The student must correct the code before WebIDE allows them to move to the next step/tab in the lab.

We use WebIDE to apply test-driven learning (TDL)[1], a pedagogical approach for teaching with test-driven development (TDD). TDD is becoming a widespread software engineering (SE) 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[2]. Studies have shown that TDL can be applied in entry level classes without removing current course material, and that students who try TDD, like TDD[1]. However, finding ways to enforce a test-driven approach with beginning programmers has proven to be elusive. WebIDE solves this by moving students through labs in a lock-step fashion, requiring them to write examples and tests before implementing solutions.

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

We present related work and discuss how WebIDE differs from other environments in section 2. Section 3 discusses the role of TDL. Section 4 gives a brief overview of the WebIDE architecture and its capabilities. Next we show WebIDE's course management in section 5, followed by the initial set of labs in section 6. Section 7 discusses the pilot study and results, then we finish with conclusions and future work.

2 Related work

Most web-based coding environments support web-based scripting languages, such as PHP and Javascript. For example, W3 schools (http://www.w3schools.com/js/), Google's API Playground (http://interactivesampler.appspot.com/), JSBin (http://jsbin.com/), and Cloud9 (http://www.cloud9ide.com/) let users evaluate Javascript. A few web-based systems can compile and run code, such as Google's Go playground (http://golang.org/) and ideone (http://ideone.com/) which supports over 40 languages.

Coderun (http://www.coderun.com/) is a web-based IDE with all the features an IDE user would expect, such as syntax highlighting, code completion, and auto deployment. Unlike WebIDE, Coderun focuses on application development instead of educational labs. Users can create, run, and debug ASP.NET, Silverlight, and Facebook applications within Coderun. Additionally, Coderun supports PHP and Javascript. Additional examples include WeScheme[12], ShiftCreate (http://edit.shiftcreate.com/), Lively Wiki[13], and a system by Azalov[14] that automatically generates lab exercises.

Automated tutors exist for a variety of academic fields. Samples include Mathematics (http://www.assistments.org/), Physics (http://www.masteringphysics.com/), and Biology/Genetics (http://biologica.concord.org/). Many of these tools have been evaluated, with promising results. For instance, Warnakulasooriya et al.[3] 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 to program. Valentine[4] 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[5], Karel micro-worlds[6, 7], automated assessment tools[8], and pedagogical development environments such as DrRacket[9], Alice (http://alice.org), and Scratch (http://scratch.mit.edu).

A few systems closely relate to WebIDE. Turings Craft (http://www.turingscraft.com) is a commercial web-based system that presents interactive exercises for Python, C, C++, and Java. Truong et al.[10] created ELP (http://www.elp.fit.qut.edu.au/) which provides fill-in-the-blank style exercises. Unlike WebIDE, Turings Craft and ELP do not apply a TDL approach, and the exercises cannot be contributed or extended by individual faculty. Parlante's CodingBat (http://codingbat.net) adopts a test-based approach, although students do not write tests and the system is limited to a set of small, focused exercises. Edwards' Web-CAT[11] web-based automated grading tool assumes student creation of automated (presumably test-driven) unit tests, but it provides no support for interactive labs.

WebIDE is unique in its combination of features: a TDL approach, completely web-based delivery, and intrinsic support for community-contributed content. Unlike the systems above, WebIDE enables lab authors to create their own labs or use existing labs, and give specific student feedback on individual exercises. This functionality was explored and reported in an early prototype[15]. Although WebIDE may be seen as an alternative to some systems such as Turings Craft, it is intended to complement many systems such as Web-CAT and BlueJ (http://www.bluej.org). The primary focus of WebIDE is on the first three to five weeks of an introductory course, after which we assume students will transition to a traditional development environment. It is possible that some faculty will use WebIDE throughout a course such as a CS0 for non-majors, or as a supplement in non-introductory courses where students must learn an unfamiliar language quickly.

3 Test-Driven Learning

Proponents of Test-Driven Development[16] argue that programmers work more effectively when they focus on the results of functions—the tests—before thinking about *how* those results are computed. TDL extends this claim to the corresponding pedagogic statement: students learn more quickly when they focus first on the set of possible inputs to a

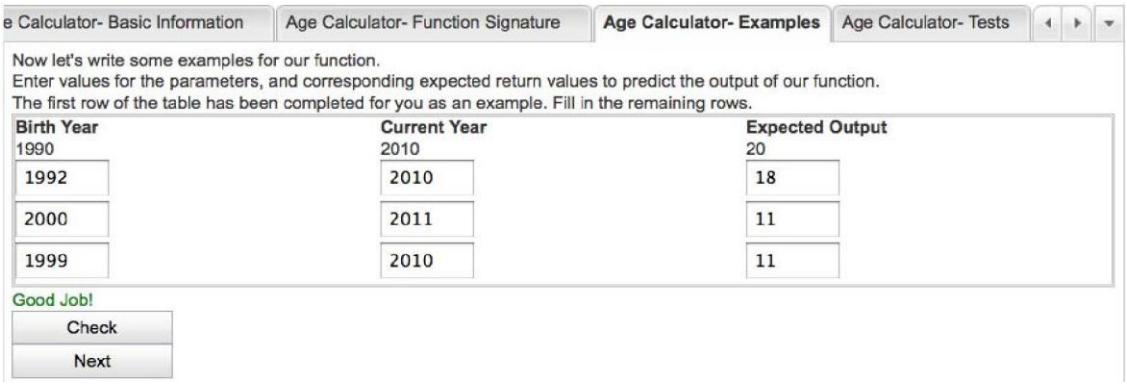

Figure 2. WebIDE step requiring student-written examples

function, and the corresponding results. Best of all, TDL has been shown to have no extra cost[1]. In other words, students can use TDL and still learn all the concepts that were originally taught, in the same amount of time.

TDL can also improve the quality of instructor feedback. In TeachScheme! [17] workshops, instructors using TDL in lab settings repeatedly report that by examining students' test cases *before* looking at their code, they can diagnose problems more quickly. Many instructors found that students discover their own problems after writing tests. Recent studies show empirical evidence that TDD makes students more productive, earn better grades, and write clean, concise, and well tested code [18, 19].

The same things that make TDL effective in one-on-one lab interactions are even more vital in an online setting. Writing a program that checks the correctness of a student's program and offers useful feedback is very difficult. By focusing on test cases, though, the task becomes vastly simpler. Additionally, correcting these errors before the students tackle the implementation of the corresponding function can save them time and stress. Figure 2 demonstrates how WebIDE can require students to create correct examples and even convert them into test cases (Figure 3) before moving on to implementing a solution.

We believe that the synergy between TDL and web-based tutors such as WebIDE is a vital step forward in making these tutors feasible and effective.

4 Architecture

WebIDE uses Google Web Toolkit (GWT) and is currently deployed on Google App Engine (GAE) with evaluators running on Amazon's EC2 cloud platform. The WebIDE architecture is focused on extensibility. Our lab specifications are written in a well-defined XML language, so that labs may be edited and contributed by third parties. Additionally, we completely decouple the presentation of the lab from the evaluator using a service-oriented architecture (SOA) where URLs identify evaluators.

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 steps:

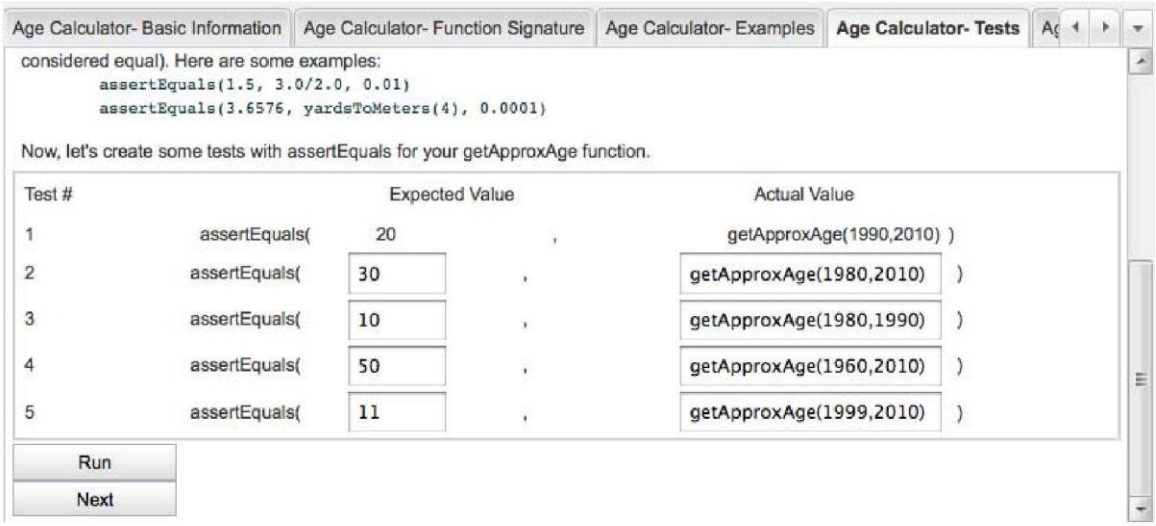

Figure 3. WebIDE step requiring student-written tests

```
start = element lab {
  attribute name { text },
  element description { text }?,
  step*
}
```

We use Jing (http://www.thaiopensource.com/relaxng/jing.html) to validate a lab's XML. The parsing phase then maps that XML to a GWT page containing user entry fields.

The evaluator associated with a given lab step is responsible for determining the correctness of student entries. Since the evaluators can be hosted on any server by any author, we supply an interface for communication between the engine and the evaluator using HTTP and encoding the request/response in JSON. Therefore, any language that can receive a HTTP request and send a HTTP response can be used to implement an evaluator.

Internal evaluators (hosted within the WebIDE engine) become faster and more reliable by removing the extra HTTP request to an external server. However, the evaluator must conform to the restrictions of the environment that WebIDE runs on. For example, GAE does not allow system calls or file I/O. Therefore, evaluators that compile a program must run on an external server such as an Amazon EC2 instance.

5 User management

WebIDE offers simple course management. First, students and professors log in using a Google account. If the user is a student, they can enroll in a course, see labs for a specific course, and see their current progress for each lab. All labs are saved automatically when a user is logged in.

If the user is a professor, they can save labs, create courses, add labs to a course, and see all students enrolled in a course. The professor can look at each student's progress,

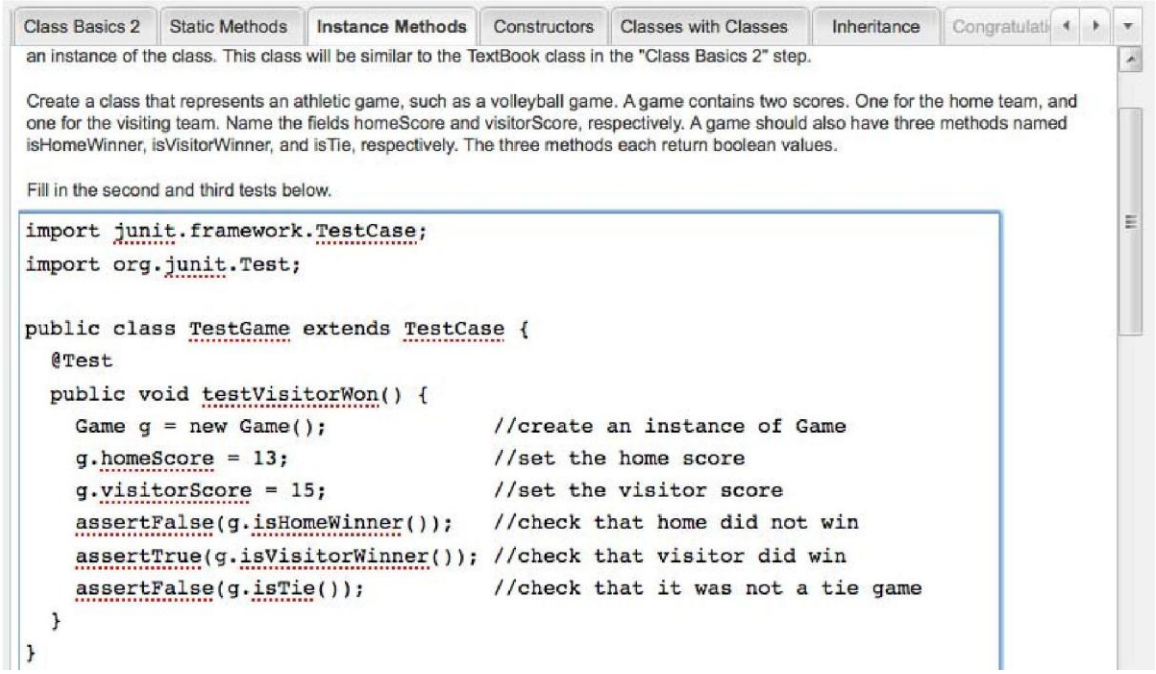

Figure 4. WebIDE step requiring students to write a full unit test class.

download student logs, or even load a student's lab within WebIDE to see the student's current step and entries.

6 Labs and evaluators

Several labs were created for use in a pilot study in a CS0 course. Lab topics included Java basics (data types and variables), if statements, functions, iterations and classes. As was seen earlier, labs consisted of steps that required students to define an interface (Figure 1), and enter correct examples (Figure 2) and tests (Figure 3) before implementing the body of a solution. As the labs progressed, students were expected to implement larger sections of code, ultimately writing entire test (Figure 4) and source classes.

The final lab developed for the pilot study does not use TDL, however, it shows the power of WebIDE by allowing students to develop a full Tic Tac Toe Android application in the browser. At the end of the lab, students are presented with a QR code that an Android phone can scan to start downloading and installing the application on the device.

An initial set of internal and external evaluators were developed for these labs, which include a regular expression evaluator, arithmetic evaluator, Java function evaluator, Java class evaluator, and a JUnit evaluator. All evaluators are generic—lab authors can control functionality via arguments—and can be used by anyone. Additional technical details and sample labs are not presented due to space limitations, but are available from the authors.

7 Pilot study

In Fall 2010, Cal Poly implemented a new CS0 course which focuses on applications of computing, such as Music, Robotics, Game development, and Android development. We

performed a pilot study with 51 students in two sections of CS0: Android Development. The primary purpose of this pilot was to gain initial feedback on WebIDE to make improvements for future studies. However, the study was designed and run so that significant differences in the two groups would be valid. We examined two hypotheses in this study: 1) students who used WebIDE perform better on programming tasks than students who used traditional static labs, and 2) students who used WebIDE spend more time on labs (because of the lock-step aspect) than students who used traditional static labs.

For each section, we randomly assigned students into group A (using WebIDE) or group B (control group using traditional static labs). We validated that there was no statistically significant difference in prior programming experience between the two groups. Students completed three weekly labs in the study beginning with Lab 4. Students programmed with Scratch (http://scratch.mit.edu/) and App Inventor (http://appinventor.googlelabs.com/) in Labs 1 through 3. Lab 4 covered Java basics, functions, and if statements. Lab 5 covered iterations and classes. Lab 6 consisted of an Android lab.

For Lab 4 and 5, group A used WebIDE with evaluation, and group B used WebIDE without evaluation. In other words, group A received immediate feedback on each lab segment, whereas group B received no feedback until they completed the entire lab, submitted it, and received feedback with a grade from the instructor. Group B students were introduced to the Eclipse development environment, and they were encouraged to check their answers by compiling and running them with unit tests in Eclipse. This was deemed equivalent to a traditional lab where the student is given lab instructions in a static HTML page and uses a development environment to complete the lab. For Lab 6, group A used WebIDE and group B used Eclipse with the Android SDK installed. Group B students were provided instructions in a static HTML page (equivalent to the WebIDE instructions) and a project stub that contained class definitions with method headers for each of the methods to be completed. Lab 6 lets us evaluate the usefulness of WebIDE as a simplified environment with no setup versus using a complex environment like Eclipse with the Android SDK.

After completing Lab 4 and 5, students were given a midterm exam with four Java programming questions. The questions asked them to 1) write a set of JUnit tests for a described method, 2) write a method that used if-then-else, 3) write a method that used nested for loops, and 4) implement two classes and two methods, where one class contained an array of instances of the second class. This last question was identical to a pre-experiment programming quiz that was given just prior to assigning Lab 4 and used to determine prior programming experience. In addition to the quantitative evaluations, qualitative surveys were administered at the end of each lab, and focus groups were conducted after Lab 6.

7.1 Analysis

There were no statistically significant differences between group A and B on their lab scores or the midterm questions, with two exceptions. First, after adjusting for previous Java experience, students who used WebIDE scored an average of 2.6 points higher on Lab 6 (Android) with a p-value of 0.006. In other words, students were more likely to successfully complete their first Android app with WebIDE, than with a traditional development environment. The focus groups seemed to reflect this as well. Several students in group B wished they had been in group A on the Android lab because it stepped them through the solution and gave feedback on correctness of interim results. Students using Eclipse

reported spending a large amount of time trying to debug small problems that were often a misunderstanding of the lab specification, whereas WebIDE would return an appropriate error message.

Secondly, among students who scored higher than zero on the pre-experiment Java quiz, students who used WebIDE scored an average of 2 points higher—with a p-value of 0.04 on the first midterm question that required them to write unit tests. In other words, among students with some prior programming experience, students who used WebIDE did better at writing automated unit tests than students who did not use WebIDE. This indicates that WebIDE may achieve one of our primary goals, which is to successfully integrate TDL into entry level courses.

It is important to note that using WebIDE did not harm students in terms of academic performance. There were no significant results showing a decrease in group A scores.

On the second hypothesis, group A (WebIDE) students reported spending 125.52 minutes on average on Lab 4, while group B reported 81.94 minutes on average. This difference was statistically significant with $p = 0.0441$. This matched observations that the lock-step aspect of WebIDE seemed to slow students down. However, on Lab 5 and Lab 6, group A students actually reported slightly lower average times than group B, although these results were not statistically significant.

7.2 Threats to validity

There are three main threats to validity. First, Cal Poly implemented CS0 for the first time in Fall 2010. The expectation is for every student to succeed in CS1 in Winter 2011. Therefore, we don't know if the overall student success will be due to CS0 or WebIDE.

Second, we ran into major errors within the environment and labs during this pilot study. At one point, the steps on classes in Lab 5 returned numerous server errors, which caused students to be extremely frustrated and, of course, halted their progress on the labs. In addition, students sometimes received vague, invalid, and incorrect error messages on the evaluation attempts. For example, one lab evaluator initially contained a race condition that occasionally caused student's code to compile over other student's code, returning either an invalid compile error or one compiler error for both students. As a result, a student who submitted correct code could potentially still receive an error message. Although most errors were resolved quickly, it was clear to students that they were working with a new product.

Finally, a few students varied from instructions without notifying the instructor. We observed a student assigned to group A join group B (this was adjusted in the evaluation). We also heard of students copying code from other students just to pass the current step. In addition, a few students reported doing everything in Eclipse then pasting their code into WebIDE. Because the labs were not completely finished in the classroom, students may have changed instructions in different ways that weren't observed.

8 Conclusion and future work

WebIDE gives professors the ability to create and track dynamic and fully customizable labs that provide rich, contextual feedback to students. We have demonstrated that WebIDE can enforce a Test-Driven Learning approach, resulting in students who are better able to write tests for their programs and will reap the TDD benefits documented in prior research.

Although our pilot study did not demonstrate that students using WebIDE perform better on programming tasks overall, we did see significant improvement in developing a beginning Android application and writing unit tests.

WebIDE is currently available to the public at http://web-ide.org. A number of additional features, additional labs, and a broader assessment in CS1 and CS2 courses are scheduled to occur in 2011. Likely improvements include infrastructure improvements (performance, data collection), lab segment options (syntax highlighting, code completion, automated evaluation), and an integrated lab authoring tool that eliminates the requirement that labs be written in XML and enables "cut and paste" of existing labs and lab segments.

Acknowledgment

We would like to thank Halli Meth for her work creating initial labs and evaluators used in the pilot study, and the CS0 students. 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.

References

[1] D. Janzen and H. Saiedian, "Test-driven learning: intrinsic integration of testing into the CS/SE curriculum," in *Proceedings of the 37th SIGCSE technical symposium on Computer science education*. ACM, 2006, p. 258.

[2] ——, "Test-driven learning in early programming courses," in *Proceedings of the 38th SIGCSE technical symposium on Computer science education*, vol. 40, no. 1. ACM, 2008, pp. 532–536.

[3] R. Warnakulasooriya, D. Palazzo, and D. E. Pritchard, "Evidence of problem-solving transfer in web-based socratic tutor," in *Physics Education Research Conference 2005*, ser. PER Conference, vol. 818, Salt Lake City, Utah, August 10-11 2005, pp. 41–44.

[4] D. W. Valentine, "Cs educational research: a meta-analysis of sigcse technical symposium proceedings," *SIGCSE Bull.*, vol. 36, no. 1, pp. 255–259, 2004.

[5] C. D. Hundhausen, S. A. Douglas, and J. T. Stasko, "A meta-study of algorithm visualization effectiveness," *Journal of Visual Languages & Computing*, vol. 13, no. 3, pp. 259 – 290, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/B6WMM-46WF0HY-2/2/60d0e87fcd1722a5205fcc56b6b15112

[6] J. Bergin, J. Roberts, R. Pattis, and M. Stehlik, *Karel++: A Gentle Introduction to the Art of Object-Oriented Programming*. New York, NY, USA: John Wiley & Sons, Inc., 1996.

[7] B. W. Becker, "Teaching cs1 with karel the robot in java," in *SIGCSE '01: Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education*. New York, NY, USA: ACM, 2001, pp. 50–54.

[8] C. Douce, D. Livingstone, and J. Orwell, "Automatic test-based assessment of programming: A review," *J. Educ. Resour. Comput.*, vol. 5, no. 3, p. 4, 2005.

[9] R. B. Findler, J. Clements, C. Flanagan, M. Flatt, S. Krishnamurthi, P. Steckler, and M. Felleisen, "Drscheme: A programming environment for Scheme," *Journal of Functional Programming*, vol. 12, no. 2, pp. 159–182, 2002.

[10] N. Truong, P. Bancroft, and P. Roe, "Learning to program through the web," in *ITiCSE '05: Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*. New York, NY, USA: ACM, 2005, pp. 9–13.

[11] S. H. Edwards, "Improving student performance by evaluating how well students test their own programs," *J. Educ. Resour. Comput.*, vol. 3, no. 3, p. 1, 2003.

[12] D. Yoo, B. Hickey, E. Schanzer, and S. Krishnamurthi. Wescheme. Online at http://www.wescheme.org/.

[13] R. Krahn, D. Ingalls, R. Hirschfeld, J. Lincke, and K. Palacz, "Lively wiki a development environment for creating and sharing active web content," in *WikiSym '09: Proceedings of the 5th International Symposium on Wikis and Open Collaboration*. New York, NY, USA: ACM, 2009, pp. 1–10.

[14] P. Azalov, "A web-based environment for introductory programming courses," *J. Comput. Small Coll.*, vol. 22, no. 4, pp. 260–267, 2007.

[15] J. Clements and D. Janzen, "Overcoming obstacles to test-driven learning on day one," in *ICSTW '10: Proceedings of the 2010 Third International Conference on Software Testing, Verification, and Validation Workshops*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 448–453.

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

[17] M. Felleisen, R. B. Findler, M. Flatt, and S. Krishnamurthi, "The teachscheme! project: Computing and programming for every student," *Computer Science Education*, vol. 14, 2004.

[18] D. Janzen and H. Saiedian, "On the influence of test-driven development on software design," in *Software Engineering Education and Training, 2006. Proceedings. 19th Conference on*, 2006, pp. 141–148.

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