Background and Related Works

Thomas Dvornik
Department of Computer Science
California Polytechnic State University
San Luis Obispo
CSC 590
June 2, 2010

1 Background

1.1 Environment Goals

The goal of the WebIDE is to make learning to program easier by addressing 4 different problems: complexity, accessibility, error addressability and flexibility.

1.1.1 Complexity

Complexity refers to the number of steps it takes for a working program. This problem directly relates to Figure 1, that shows a common student work flow. By minimizing the complexity, or the number of steps to get a working program, we hope to reduce confusion among students. This will help students focus on the concepts at hand.

![Diagram showing the workflow: Install SDK -> Editor -> Program -> Terminal -> Compile. There is a feedback loop indicating errors.]
1.1.2 Accessibility

The environment should be accessible anywhere, at anytime, without worrying about installation or system requirements. The student won’t have to worry about setting up their machine or going to the lab. They can go to ANY machine and start programming assignments, or start from where they left off.

1.1.3 Error Addressability

Students can often struggle with compile errors or runtime errors. Compile errors frequently refer to an invalid line number. For example, if a student forgets a semi-colon, it will show the error on a line below problem line. Students can look at these errors for a long time until they figure it out. Exceptions can also confuse students, as a stack trace can provide a lot of detailed and complex information. Instead, we can easily mask these errors for small labs and programs to help the students figure out there errors faster.

1.1.4 Flexibility

Flexibility concentrates on given professors an easy way to convert and create labs that can be used in WebIDE. Not only does it need to be flexible and easy so professors will use it, but so the labs will be simple for students to do while upholding the 3 previous constraints.

1.2 Test-Driven Learning

WebIDE focuses its solution to the above goals around Test-Driven Learning (TDL). TDL is an effective method to teach Test-Driven Development (TDD), an extreme programming (XP) test-first technique [7]. The idea is to create test before any code is written. TDD can often be defined by the following steps [4].

1. Create a test for an unimplemented unit of functionality.

2. Run all test and check for failures.
3. Write code for the unit of functionality.

4. Run all test and check of failures.

5. Refactor.

The result is small and efficient code. TDD is more than a testing method; it is a design method. Writing test before writing code forces the implementer to think about the functionality and design of the function and system. For example, inputs and outputs of a function must be defined in order to test it.

Stephen Edwards, from Virginia Tech, explains the lack of testing causes students to perform “trial and error.” [5] As a result, students obtain 4 misguided views that, as Edward says, some students will never lose.

1. If the compiler accepts my code, I removed all the errors.

2. My code will work all the time if it works for a couple values.

3. I’ll just switch a few things around to see if my problems will go away.

4. My program works for the instructor’s data so I am done.

TDD can resolve these issues just by its definition [4]. TDD, however, is not regularly taught in the Computer Science curriculum [12]. There is a lot of empirical evidence that TDD makes students more productive, earn better grades, and write clean and concise code [8, 11]. TDD teaches students how to write test first [7], removing the “trial and error” approach. Students who practice TDD write more test [10]. Best of all, TDL has been shown to have no extra cost [9]. In other words, students can learn TDD and all the concepts that were originally taught in the same amount of time.

1.3 Terminology

I use a few different terms throughout this paper. (I will have more terms that will be defined in greater detail the further I get into my thesis)
• **Activity:** A short programming exercise.

• **Module:** A small collection of activities.

• **Lab:** A collection of modules and activities to be done in a given amount of time. i.e. an hour, week, etc.

• **Segment:** A piece of code.

2 Related Works

There are a few different online environments for teaching computer science that are closely related to WebIDE with interesting results. (I may not need to talk about their results, but depending on my results, it may be interesting to compare. If I compare, should I have the related works section at the end?)

2.1 ELP: Environment for Learning to Program

The Environment for Learning to Program (ELP) provides "filling in the gap" exercises in an online environment as seen in Figure 2. The ELP was developed by Nghi Truong, Peter Bancroft and Paul Roe from Queensland University of Technology in Australia. The authors wrote several papers that describe the idea, implementation, and results for the ELP [18][19][17].

The ELP addresses the high failure rates of the entry level computer science classes. It allows students to program in a simplistic development environment by eliminating the "difficult" parts. One example is interacting with the compiler.

Analytic tools are provided to help determine when the student has written complex or invalid code. The analysis phase is split into two parts, a static analysis and dynamic analysis. The static analysis makes use of Software Engineering Metric Analysis and Structural Similarity Analysis. The dynamic analysis makes use of black and white box testing. The complexity of the analysis
works great for small programs but breaks down as the program or exercises get larger and more complex.

Exercises can be configured and created by any professor. The exercises are described in an easy XML structure. The professor can specify a solution, and decide what code should be displayed or filled in. Comments and hints can also be specified in the document.

Tutors can provide annotations on student programs. This increased the effectiveness of a tutor, allowing them to help many students at once. At the same time, the students feel like they are still getting individual, instant feedback. The ELP allows students to work offline and online, although the full functionality does not work offline. All exercises and student code is stored online. The results of the student programs have to be downloaded and ran locally.

There were a couple different experiments with the ELP to evaluate its effectiveness. The first study consisted of 30 students in the 5th week of Software Development 1 during part of a catch up course[18]. Only 12 students took a survey after using the system. The survey asked if the ELP helped them write and understand programs. It also asked if they would like to use the system for the remainder of the course. All of the survey results were positive.

The second experiment consisted of an evaluation on a course using the ELP[19]. This survey was also given during the 5th week and consisted of 46 students; 63% who evaluated the ELP positively. The evaluation included space for comments, which proved to be more useful. Several comments suggested that the ”fill in the gap” exercises are good for ”not so good” programmers. There were several comments that stated the anytime, anywhere characteristic of the ELP was the most useful feature. Some students said the ELP made it easy to write a program and that they could not compile a program before.
Hello

ELP - A first C# program

Hello.cs

```csharp
// file: Hello.cs
using System;

class Hello {
    public static void Main() {
        // Insert your name
        Console.WriteLine("Hello, ");
        Console.Write("Enter to continue");
        Console.ReadLine();
    }
}
```

Figure 2: ELP Environment[19]
2.2 CodeLabs

CodeLabs is a web based exercise system. The environment was previously referred to as WebToTeach, as seen in Figure 3 before it went commercial in 1999[16]. CodeLabs focuses on short examples and iterations to help the students understand concepts, which CodeLabs calls "Graduating Complexity." It uses a custom automated code-checking system that gives students instant feedback.

CodeLabs offers fill in the gap type exercises that provide instant feedback. The system automatically saves student progress. It provides a custom evaluation and analysis of a student’s progress to the professor. CodeLabs provides over 300 exercises for professors to organize into custom course plans. The exercises range from basic to advanced concepts. Each exercise consists of multiple parts, in multiple programming languages.

CodeLabs is a successful commercial product. CodeLabs claims to be used by over 100 institutions and more than 27,000 students [20].

2.3 BlueJ

BlueJ is a Java environment specifically for teaching object oriented programming to beginners. It displays classes and methods as objects in a UML diagram as seen in Figure 4. The diagram allows students to interact with the objects. Students can click on objects and change parameters. They can also bring up the source code for a class. Students can create test objects for classes by directly calling the methods within the original object. BlueJ enables a good overview of all the classes, helping students get the general picture. [14][15][13]

Bluej also provides a slew of additional features without destroying the simplistic user interface. Some additional features include a fully functional Java debugger, integration with Java Docs, and direct feedback with individual objects.
Figure 3: WebToTeach Environment[2]
The first study consisted of 26 students out of the 109 students taking a CS1 course[3]. Those 26 students were separated into two groups of 13 students each. The first group used BlueJ and the second group used the traditional method of a plain text editor with limited integration to the JDK. On average, group 1 outperformed group 2 by 9% on all quizzes and exams.

The study also got feedback from 43% of the participating students. Almost all of the feedback was positive. Students enjoyed the exercises and felt that BlueJ clearly presented the concepts associated with object oriented programming. There were also negative comments about the environment running slow.

The next study evaluated the switch from c++ to BlueJ in a first quarter programming unit[6]. The quarter’s failure rate decreased by 4% across 333 students. However, these results cannot be distinguished between the switch to BlueJ or the switch to Java. The students also evaluated BlueJ 3.5 out of 5.

The last study consisted of 40 students taking a survey using a scale of 1(high) to 7(low)[21]. On average, students rated BlueJ a 3 on its effectiveness to help them learn. However, they rated the BlueJ environment a mean of 4.3. The author got several comments that the lack of punctuation was really nice.
Figure 4: BlueJ Environment[21]
References


A new system is implemented based on WebToTeach. The new environment is used to take test on the web. The most useful part of this short paper is the comments on security.


This paper talks about the design decisions and performance of WebToTeach. It talks about the services used and the process to set up exercised within the environment.


A whole course is outlined including a schedule, reading assignments, assignments, and deadlines. The assignments use CRC-Cards and BlueJ. An evaluation is performed looking at quiz and exam scores, followed by student feedback.


This paper addressed difficulties that students encounter and how BlueJ attempts to solve those problems. Students use the environment and perform an evaluation on BlueJ. Students rated BlueJ a mean of 4.3 on a scale of 1-7, where 7 is low.


Students using test-first strategies write more test and are more productive.


Using TDD provides better code. This study shows metrics on productivity, code length, coverage, and complexity.


A method for teaching TDD is proposed. Shows that test-first gives students slightly better comprehension at no extra cost.


Shows statistics and metrics on TDD in industry. In general, programmers using test-first strategies wrote smaller units that were less complex and highly tested.


Students using test-first strategies have been shown to be more productive and average a whole letter grade higher than students using test-last strategies.

TDD has become one of the most used practices in XP. This study runs two experiments in master courses and discuss the pros and cons.


BlueJ’s website which includes downloads for the BlueJ environment.


This paper describes the features and pedagogy of BlueJ. This paper is useful because if gives specific assignment sequences; starting with steps, and then the overall picture.


This paper discusses the history of OO design and the paradigm shift to OO. It then describes the features of BlueJ. An interaction and experimentation study is done resulting in positive feedback.


Gives an overview of IDE, optimizations, and user interface of CodeLab. Gives good examples of exercises used in the environment.

The first paper published on the ELP that gives an overview of the system, including design decisions. An evaluation of the ELP is done with a survey to 12 students.


Discusses the current status of the Environment for Learning to Program (ELP) used by Queensland University of Technology. They cover the reasons behind the implementation, compare it to other tools, and explain the results from an experiment where students used the ELP.


Describes the features and framework of the ELP. The paper compares the ELP to different environments. Final, it does a case study on 39 students during the 5th week or a course using the ELP.


Website for CodeLab. Gives information on the environment and free demos to instructors.


This paper goes into deep detail of all the BlueJ features. It also gives a break down of student usages between features and an evaluation on speed and performance.