1 Introduction

Software is currently integrated into almost all aspects of life. Businesses, oil mines, and cars are just a few examples where software is integrated to make them more efficient. Soon a shopping cart will be able to receive your shopping list wirelessly. Then, guide you around the store after calculating the shortest path in a matter of nanoseconds. Just wait!

As a result, the need for Computer Scientist is continuously growing. This brings up an important question. How can we increase the number of computer science students? Getting students interested and enrolled in computer science is only half the battle. Computer science (CS) is known to have one of the highest failure rates of all majors. Back in 1997, the average failure rate in the first CS course at colleges ranged between 30% and 60% [1]. Today, failure rates are just as high. Colleges across the world address high failure rates [10, 9, 13].

Currently, Cal Poly is introducing new CS classes for freshman to take before the 101, 102, and 103 series. These classes include robotics, music, android development, and game design. These classes intend to do two things. First, get more students
interested in computer science. Second, decrease failure rates within the department. This is just one example of how academia is getting students prepared for CS. Many colleges currently offer "CS0" classes as an introduction before the 100 series [14, 12].

Another solution to ease students into CS is to make the methods and tools used to teach computer science even better. For decades, researchers have been developing tools to make programming easier to learn [8]. Many have been very successful [2, 11, 14]. Most environments are specific to certain, languages, concepts, or different learning methodologies. Other environments leave some of those options up to the instructor, and instead focus on simplifying or addressing problems with the steps to create a program [8].

An example of a common flow graph for entry level programmers can be look like Figure 1. Here, the student has to install a SDK. Open an editor, type a program, and save. Open a terminal to compile their program. Read and understand compile errors. Then jump back and forth to fix compile errors and logic errors until they
have a working program. Most of these steps distract the student from concentrating on the concept at hand. Instead the flow should be more like Figure 2. This is one of the problems WebIDE tries to solve.

WebIDE tries to solve four problems: complexity, accessibility, error addressability, and flexibility. Complexity tries to reduce the number of steps it takes get a working program. Accessibility focuses on making the environment accessible from anywhere at anytime, regardless of installation of the environment. Error addressability simplifies convoluted error massage, hopefully enabling students to find errors faster. Lastly, flexibility lets instructors create and modify labs using different teaching methodologies, languages, and constructs while keeping the other three problems in mind.

Another contribution of WebIDE allows instructors to focus on Test Driven Development (TDD), or other testing strategies, such as Test Driven Learning (TDL) [5]. Testing is not regularly taught in entry level courses [10, 3]. Students using TDL are more productive, write more test, and get better grades [7]. As a result, students write small, less complex code that is highly tested [4, 6]. The effect of teaching testing strategies early on results in more written test later on. (Forget which paper says this, will cite when I find it.)

WebIDE focuses on entry level programmers in the first few weeks of intro level classes. It is during these weeks that students are bombarded with programming concepts as well as the tools to implement those concepts. Of course students will need to interact with the terminal and compiler at some point, but that can be done at a later point after the students have a foundation.

WebIDE was used in 4 sections of CPE 123 courses at Cal Poly. We also performed a pilot study in a high school class at San Luis High. WebIDE turned out to be an effective learning tool, accelerating the first few weeks of intro level courses for
beginner programmers. All the feedback from students were positive, indicating that WebIDE helped them learn programming concepts efficiently.

References


High failure rate in computer science are a big problem. One of the biggest problems is syntax and syntax related errors. A solution is to introduce students to problem-solving skills before applying those skills in a programming environment.


Students grade 7 and younger learn better in a graphical environment while senior programmers learn better in a textual environment. Students between these grade levels find graphical environments too easy and textual environments too difficult. Thus, the BrickLayer environment is recommended to ”fill the gap” so these students can have a more efficient learning environment.


This paper summarizing 5 novice programming environments based on anecdotal, analytical, and empirical techniques. A rubric is created for evaluating these systems.

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

CS/SE curriculum. In *Proceedings of the 37th SIGCSE technical symposium on Computer

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 av-
erage a whole letter grade higher than students using test-last strategies.


A HUGE taxonomy of CS environments ranging from super simple systems to very
complex. Also looks at systems for all grade levels, as well as old and new systems.
This paper provides a great compilation of the environments with corresponding
features and categories.

programming. In *Proceedings of the twenty-fourth SIGCSE technical symposium on Computer
Identifies and addresses problems with languages and tools in undergraduate classes. An environment is created to address broad language support, integrated tools, consistent user interface, and efficient and portable implementations. The environment is geared for all students of all levels in an undergraduate degree.


An in depth study is performed to determine the skills of first-year CS students. This paper provides useful methodologies, assessment techniques, and guidelines for environments. It also provides great charts of their results.


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


This paper discusses their experience with teaching Alice in CS0. Student’s understood the material, but encountered trouble when moving to a text based environment. The paper has a lot of anecdotal observations, but not much empirical data.


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.

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.