



UNIVERSITY OF CAPE TOWN  
IYUNIVESITHI YASEKAPA • UNIVERSITEIT VAN KAAPSTAD



DEPARTMENT OF COMPUTER SCIENCE

# COMPUTER SCIENCE HONOURS

## FINAL PAPER

### 2016

Title: PyTut: Evaluating the Effectiveness of E-Learning Tools in Computer Science Education

Author: Carla Kirk-Cohen

Project abbreviation: PYTUT

Supervisor: Gary Stewart

Category	Min	Max	Chosen
Requirement Analysis and Design	0	20	0
Theoretical Analysis	0	25	0
Experiment Design and Execution	0	20	20
System Development and Implementation	0	15	5
Results, Findings and Conclusion	10	20	20
Aim Formulation and Background Work	10	15	15
Quality of Paper Writing and Presentation	10		10
Quality of Deliverables	10		10
Overall General Project Evaluation	0	10	0
<b>Total marks</b>			80

# PyTut: Evaluating the Effectiveness of E-Learning Tools in Computer Science Education

Carla Kirk-Cohen  
University of Cape Town  
Rondebosch  
Cape Town  
carlakirkcohen@gmail.com

## ABSTRACT

This paper aims to identify the effectiveness of interactive code, visual debugging, automated assessment and interactive feedback in aiding beginner programmers in completing programming tasks. To test these features, a prototype providing the features was developed. This prototype was intended to be tested with UCT's CSC1010H class. However, protest action across South Africa necessitated the use of expert users who evaluated each feature's ability to aid beginners in typical tasks and its suitability for inclusion in future E-learning systems using a 1-5 likert scale and follow up open ended questions. Overall, the automated marker was found to be the most effective feature provided. The flaws of testing using a prototype were made clear by the responses of the open ended questions, many of which concern the specific instance of the feature rather than the concept of the feature in general. Further work is required in evaluating different instances of individual features to gain more insight into their effectiveness.

## CCS Concepts

•Applied computing → Interactive learning environments; E-learning; Interactive learning environments; E-learning;

## Keywords

Python; Web-Based; Tutor

## 1. INTRODUCTION

Computer Science education is facing a retention crisis [26]. Large numbers of students enroll in programming courses, but very few successfully complete them. Cognitive overload has been identified as one of the difficulties that students face when first attempting to learn to program [2]. This occurs due to difficulty in adapting to an algorithmic way of thinking and the challenge of expressing algorithms in programming languages [2]. It has also been found that complex

IDE installation and features can be a frustrating barrier to learning [12].

The University of Cape Town attempts to address this issue by streaming first year Computer Science students based on a placement test written after the first term of instruction. Students who perform adequately continue on to complete the CS1 course in one year. Under performing students are moved into an extended degree program computer science course, CSC1010H, which covers work at a slower pace and provides additional academic support [36]. This project will develop an e-learning prototype which will be used to evaluate how e-learning methods can be used to improve the learning experience for students. The prototype will focus on Python programming, as Python is the language of choice in CSC1010H.

A review of existing literature indicates that e-learning systems have been found to be equivalent to or marginally more effective than traditional instruction [41]. As the development of a comprehensive e-learning system requires a large amount of time and resources, it is important that the benefits of developing such systems are carefully weighed up against the costs. The prototype developed for this project will aim to provide insight into the effectiveness of different elements of e-learning systems developed for computer science. The research question that this project will strive to answer is: How effective are interactive code, code visualization, automated assessment and interactive feedback in aiding beginner programmers? The prototype was intended to be tested in CSC1010H lab exercise sessions, with quantitative usage analytics and qualitative survey questions taken to provide insight into the effectiveness of the various components. This was not possible due to Fees Must Fall protests that shut down the University of Cape Town during the period in which testing was intended to occur. Instead, the prototype was tested with a group of expert users recruited from UCT's Computer Science honours and masters classes.

The results obtained indicated that all of the features implemented on the prototype were highly valued by expert users. The automated marker outshone the other features. This was predominantly due to its ability to motivate learners and allow them to work at their own pace; two key features of E-learning systems. The interactive code feature that was expected to lower barriers to programming was found to be an over simplification of a vital component to completing programming tasks. The visual debugger was considered the least important feature for inclusion in future systems. From its under performance in ability to engage students, this is attributed to a failure to promote ac-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

tive learning. The interactive feedback feature placed last overall. This was attributed to the use of expert users with no real need to seek assistance through the feature. The knowledge gained from this experiments will hopefully help to provide direction in future E-learning endeavours.

## 1.1 Project Significance

Programming is a vital skill that is constantly in demand in the job market and across academic disciplines. If high failure rates cannot be addressed, shortages of skilled professionals in the Computer Science field will only perpetuate. Learning to program requires time intensive practice that cannot feasibly be completed solely in a lecture or tutorial setting [23]. All students must practice programming in their own time if they wish to succeed. This prototype developed for this project provides an interactive environment aimed at aiding them in independent completion programming exercises. Online tools to aid students in learning to program already exist. These tools were considered for use in this research, but were deemed inappropriate for our purposes as running analytics on an externally owned site would not be feasible. Additionally, the effectiveness of e-learning has been found to be conditional on the e-learning initiatives being well-focused and relatively small scale so it was decided that development of a prototype that was specifically aimed at UCT students was preferable [31]. The systems already available operate on a large scale and are not specifically tailored to an individual course's syllabus. To examine the effectiveness of an e-learning system and its components, it is important that the system being used is the most suitable for the task at hand. For this reason, a prototype was developed instead of using an already existing system.

## 1.2 Project Aims

This project will seek to isolate the effectiveness of these individual components to determine how much each contributes to the learning process. This project will examine the relationship between how much students use each component and how much students value each component. This is important for the development of E-learning platforms, as components that students do not use or do not value should not be included in future systems;; inclusion runs the risk of potentially distracting or confusing students.

## 1.3 Project Issues

A literature review of 96 research reports, with data cumulatively gathered from 19331 subjects, found that e-learning initiatives were 6% more effective than traditional classroom instruction [41]. Other studies have found that there is no significant difference between classroom instruction and e-learning's effectiveness [43]. The effectiveness of e-learning has not been definitively proven so it is important to acknowledge that the prototype introduced by this project may not significantly improve the learning experience of beginner programmers. The introduction of an unhelpful e-learning prototype will not help to address the problem, it will exacerbate it, so it is vital that research is not conducted in an unbiased manner.

It has also been observed that students that are unfamiliar with or anxious about the use of computers perform worse off in e-learning initiatives than students that are comfortable with computers [29]. This uncontrolled variable may

impact the results of this project. It was hoped that conducting tests late in the year with a class that has been in contact with computers for at least 8 months will mitigate this problem. An unexpected side-effect of testing with computer literate honours and masters students was that this issue did not need to be addressed.

# 2. LITERATURE REVIEW

## 2.1 E-Learning

E-learning is broadly defined as any instruction that is delivered electronically [13]. It has become increasingly more popular as the prevalence of Internet access and mobile devices has grown [1]. The appeal of E-Learning lies in its ability to allow students to engage actively in learning materials at their own pace, rather than merely receiving information in a typical classroom setting where they have little to no control over the speed at which information is received [30].

E-Learning has become somewhat of a hyped phenomenon over the last decade. It is important to separate the hype from reality to meaningfully assess the effectiveness of e-learning [43]. It is generally agreed that students can learn content that is delivered by technology [37, 4, 35]. However, the effectiveness of e-learning has been found to be conditional on the e-learning initiatives being well-focused and relatively small scale [31].

The benefits of e-learning include allowing students to move at their own pace, ability to track learner progress and the capacity to educate many students quickly [43]. Empirical evidence concerning the superiority of e-learning is not unanimous so it is important to understand that e-learning can be more effective than classroom instruction, but this is not always the case [43]. Educators should meaningfully compare the benefits of an e-learning initiative to the effort of implementing it and the investment in IT services and staff needed before committing to an e-learning venture.

Evidence collected suggests that the combination of e-learning and traditional classroom instruction can be highly effective [41]. Further research into this field is required before any definitive judgements can be made. Moore and Kearsley (2011) argue that instead of pitting one method against the other, it would be more productive to determine which students and topics are most suitable to each method, as the difference in effectiveness between the two appears to be limited [33].

## 2.2 Computer Science E-Learning Systems

This section describes the components that have commonly been used in E-Learning systems for introductory programming courses. The effectiveness of these components and existing implementations are discussed.

### 2.2.1 Interactive Code Examples

When beginning to learn how to program, it is important that students are able to focus on developing an understanding of computing concepts [12]. Traditional IDEs are designed with complex features to cater for experienced programmers. These features can be overwhelming for beginners and can very easily detract from the learning process [12]. In addition, installation and setup of these IDEs can be a frustrating barrier to learners; often, a large amount of time is spent on installing, configuring and understanding IDEs rather than on the actual coding. Simple, web-based

execution of code snippets has been found to be a much more beginner-friendly way of introducing a student to programming. In the context of web-based systems, interactive code comes in the form of a notepad style frame, which may contain code pre-loaded by instructors, that students can execute and, in some cases, debug. This format is more suitable because web-browsers automatically download the necessary resources to execute code which appears in a not-threatening, familiar notepad style frame [42]. Cleveland University ran an experiment in their introductory course to C programming to determine the effect of an interactive web-based learning tool learning about pointers in C [30]. This tool contained interactive code frames which students could run and edit in their own time. In the first semester, the course was taught in a traditional classroom style. The basic concept of pointers was explained in lectures and examples were discussed in class. In the second semester, the traditional classroom approach was supplemented with an interactive learning tool. Again, basic pointer concepts were explained in class but this group was told to use the interactive learning tool to go through examples. At the end of each course, each group of students wrote a test on pointers which was a variation on the examples that has been presented in class and online respectively. The results were remarkably different for the two groups. Only 10% of the class with access to the online tool failed and 80% achieved marks over 90%; in the regular class, 30% failed and only 49% achieved over 90%. These results have largely been attributed to learners being able to work through the examples in their own time, rather than being limited to the time that a lecture allows. [30] It is, however, important to note that the more successful group participated in the study in the latter half of the academic year, so may have benefited from other programming experience.

Concerns about experiment design aside, the experience at Cleveland University makes a strong case for the benefits of interactive coding examples. This is consistent with findings that interactive learning environments promote deep cognitive processing when they engage learners in active learning [22]. Active learning is when students are called on to apply information that they have learned, rather than just passively observing [7]. The use of interactivity is still a debatable issue as some studies have found that interactivity obstructs learning by unnecessarily increasing students' cognitive load [22]. Despite some debate, the emerging consensus appears to be that the degree to which students are required to engage with interactive tools determines the benefit that they confer.

## 2.2.2 Visualization of Code

Many concepts in Computer Science are inherently abstract. Visualization of these concepts is widely perceived to be effective in helping learners build conceptual models [34]. This belief cannot be uniformly supported by research, as there is still a degree of controversy surrounding the topic [17]. Clarke (1983) argues that the interactive way in which media is delivered to students produces educational benefits rather than the use of media itself [5]. Kozma (1994) disagrees, stating that Clarke's research has failed to consider the interaction between media and a learner's cognitive resources [25]. Cobb (1997) adds that Clarke's outright denial of the usefulness of media in education has stunted growth in what could have been a more thoroughly studied

field [6]. This insight is particularly relevant with the introduction of the dynamic Web 2.0, which is radically different to the Web as Clarke and Kozma experienced decades ago. Hundhausen's (2002) more recent findings have indicated that, as with interactive code, students who engage with visualizations actively obtain better results than students who passively view visualizations [18]. Building a conceptual model of how a programming language works is essential to successful programming [17]. According to constructivist learning theory, computer science students do this by actively acquiring knowledge through the merging of what they observe and the models that they've already internalized rather than by rote learning of textbooks or lecture notes [3]. It has been shown that animation and graphics can be used to help students develop this internal model [40]. There are 4 different styles of visualization used in computer science education: program visualization, algorithm visualization, visual programming environments and microworlds [17].

### 2.2.2.1 Program Visualization

Program Visualization is the low level visualization of a program's source code [17]. This kind of visualization is most successful when attempting to debug programs as it provides visualizations of the program's current state (variable values, the content of the stack etc). Students are able to interact with program visualizations by stepping through their code and observing changes in the visualization.

### 2.2.2.2 Algorithm Visualization

Algorithm Visualization provides a more conceptual visualization of the program than program visualization [17]. This high level visualization is used to explain the operation of algorithms. These systems are mostly used as teaching aids and require lecturers to add visualization code to algorithms or write animation code from scratch [17].

ALVIE, Algorithm Visualization Environment, was developed by Crescenzi and Nocentini (2007) at the University of Florence for use in a second year computer science course [9]. Algorithms are visualized after they have been executed using the interesting event approach. This means that visualization code is added to the algorithm when important steps in the process occur [9]. ALVIE was implemented in Java and is therefore highly portable. The effectiveness of ALVIE was tested in a class of 66 second year students, 43 of whom successfully completed the course [9]. In an end of course survey completed by the whole class, the material used received 8.04/10 for value [9]. In addition to the standard university survey, 20 students completed a survey specifically concerned with the use of ALVIE. When asked if ALVIE's visualizations helped them understand how the algorithms discussed in class worked, 19/20 students indicated that their experience was positive or very positive [9]. ALVIE's success has ensured its continued use at the University of Florence.

### 2.2.2.3 Visual Programming Environments

Visual Programming Environments provide users with building blocks containing code or pseudo code to create programs [17]. This kind of visualization allows students to begin to think algorithmically without having to understand complex syntax or programming environments [8].

Alice is a visual programming environment developed at Carnegie Mellon University [24]. It uses visual 3D objects such as cars or animals alongside a drop and drag set of instructions to teach students to think algorithmically [24].

When Alice programs are executed, students can watch the effects of their instructions and are provided with pseudo code to familiarize them with the way in which actual programs are structures [24]. Alice was successfully used at Ithica College in a summer course aimed at preparing high school students for programming at university, with 85% of students indicating that their confidence in programming was 9 or 10 out of 10 [8].

California Lutheran University also used Alice as a learning tool in two basic (pre-CS1) programming courses: a regular day course for students and a nightschool course for adults [24]. Students indicated that they found the interactive animated environment helpful in understanding complex problems but those who moved on to CS1 struggled to move from drop-and-drag programming to the use of proper syntax and debugging [24]. It was also found that learning in Alice was very environment specific; students with poor logical thinking skills tended to struggle to solve problems outside of Alice. The experience at CLU indicates that while Alice may be effective in courses intended to prepare students for first year programming, the simplicity of a visual programming environment can fail to prepare students for actual programming problems.

#### 2.2.2.4 Microworlds

Microworlds facilitate programming by providing a specialized teaching language which simplifies the number of instructions available to programmers [17]. Physical metaphors such a manipulating a robot are used to visualize program execution. Northwest Missouri State University tested the use of a microworld called Jeroo in their introductory Java programming course [10]. Jeroo is a kangaroo inspired animal that can move around an island in search of food while avoiding poachers' nets. Students are provided with very basic instructions (left, right, ahead, here) which are used to navigate the island map [10]. Basic methods such as `isNet()` and `isFacing()` are also provided. Over 2 seminars of use, 97 students were asked to complete a survey on the use of Jeroo [10]. On a likert scale (1-5), students indicated that Jeroo was effective in introducing them to programming, rating it 4.04/5 on average [10].

### 2.2.3 Instructor Feedback

Personal interaction in lectures and tutorials has been proven to be more effective than textbooks and example programs [7]. This is problematic because introductory Computer Science classes are often very large and must cover a substantial syllabus in a limited amount of time; lecturers often have to discourage questions in class, or limit the time they spend answering them for the sake of covering the material allocated to the time. Lecturers play a vital role in the use of e-learning in courses [28]. It has been found that web based learning is 19% more effective when students receive feedback throughout the course [41].

### 2.2.4 Automated Assessment

Before discussing automated assessment, it is important to note that no grading system can replace interaction with a real teacher [39]. This does not mean that these systems are without merit; teachers and students can benefit from automated assessment systems. Automated assessment systems can replace the marking process completely. The success of this approach is dependent on a capable plagiarism detection mechanism to prevent the students from taking

advantage of the system [38]. Alternately, automated assessment tools can be used by teacher to analyze program features for large groups of students or track the progress of a particular student.

Automated assessment allows students to receive instant feedback on their work which can be helpful when tackling problems without a teacher present. This is particularly useful for beginner programmers who often make trivial mistakes which are easy to fix when provided with basic guidance [16]. In introductory programming courses, the use of an automated assessment system has the potential to free lecturers and tutors to answer more complex issues instead of wasting time on fixing trivial errors. It is, however, important to place limits on use of automated assessment systems to prevent students from adopting a trail-and-error, brute force approach to programming [38]. Another benefit of automated assessment systems is that they offer objectivity when marking that is often unattainable when large classes' work is marked by more than one person [19].

Reinforcement theory states that students' motivation and task involvement is influenced by the promise of performance feedback. In short, if students expect to be evaluated, they become motivated to achieve good results [27]. Motivation to succeed is vitally important for beginner programmers as they will often face long periods of debugging before success is achieved. Automated assessment can aid in providing beginner programmers with goals that motivate them to keep going. An end of term survey of students in an introductory programming class conducted at MIT on student perceptions of the usefulness of their automated assessment tool CAT SOOP found that 40% of students rated it 5/5 and 32% rated it 4/5 on a likert [16].

The majority of automated assessment systems check programs for correctness using black box testing. This is a method of testing that compares the output of the submitted code to the output of a model solution for a given set of input [21]. This is an effective way to approach automated marking because a wide range of programs can be marked in this way. The major drawback of black box testing is that creating sufficient tests to check for correctness is time consuming for lecturers. The black box method is limited to checking program correctness, as tests cannot assess students programming style [11]. This can be specifically problematic in beginners' programs where students may produce the right output but do so in a clumsy way that does not incorporate the components the assignment was aimed to test (using multiple nested if statements instead of a switch statement, for example). Automated assessment tools cannot award marks for programming style or innovation, but they can provide markers with analytic information that helps them determine that the program has taken on the expected form (a count of the number of switch statements, for example, can be used to ensure that students have employed the concept). Other automated assessment tools use machine learning to produce hints for students, but these systems are beyond the scope of this paper [20].

In place of instructor feedback, which is often unattainable for logistical reasons, automated assessment systems can be used to provide users with feedback. While feedback from a lecturer or tutor is always preferable, automated feedback is the next best way to encourage learners to actively engage in debugging their programs.

### 3. PROTOTYPE DESIGN

#### 3.1 Design Choices

In browser implementation was a priority when the prototype for this project was developed. This is important because it removes the need for users to set up their environment; the relevant resources will be automatically downloaded and run on the local machine. This is particularly important to beginner programmers as they are more likely to encounter problems during programming environment installation, configuration, debugging and so on. The combination of familiar web-style navigation with a directly editable in-browser view of source code made tasks very approachable for beginners [32]. Web applications have shown great potential in online education due to web browsers' ubiquitous accessibility and cross-platform compatibility.

Initially, the e-learning prototype developed for this project was intended to be developed using the ASP.NET framework with a backend SQL server. The first iteration of this prototype was developed in C# using Visual Studio's model view controller template. The use of this traditional framework was found to be too slow for the development of multiple prototype iterations. For the second iteration of the prototype, the decision was made to move to Angular, a JavaScript MVC framework, and to use Firebase, Google's cloud based JSON database. Angular was chosen because it provided a more lightweight client side solution to the problem at hand. It was primarily chosen as the framework for this prototype because controllers are executed on the client side. Executing the majority of tasks on the client side removes the risk of server side congestion. Angular allows for direct manipulation of the DOM through two way data binding; data is immediately synchronized between the model and view components. This binding allows for faster DOM manipulation than traditional methods of editing innerHTML.

The use of Firebase logically followed the use of Angular, as both platforms are supported by the vast Google developer community. Firebase is a JSON based database; all instructions are executed client side. Executing too much client side can be a disadvantage because it places the computational load on client devices which may not be equipped to handle the load. The risk of client side overload is low in the case of this prototype because Firebase simply has to handle authentication and the occasional query for the automated marker. Google analytics were used to record the details of user interactions with the prototype.

#### 3.2 Resources Used

##### 3.2.1 Skulpt

Skulpt is a web-based Python interpreter that allows code to be executed entirely on the client side [12]. It was selected as the interpreter for this prototype because it has successfully been used in many online platforms to execute python client side [14]. Skulpt compiles the user's Python code to JavaScript at runtime and executes it in the browser's JavaScript virtual machine [32]. Skulpt consists of the typical components of a compiler: code is broken up into tokens by a lexer, a parser builds an abstract syntax tree, the semantic analyzer checks for semantic errors and a code generator produces runnable JavaScript code [42]. One of the greatest advantages of Skulpt is that it is entirely based on the client side. A server is not needed so potential prob-

lems of server-side congestion are avoided [42]. Its major drawback is that it does not provide debugging facilities of any kind. Implementing the Python debugging class PDB in Skulpt was investigated for this project. This challenge was deemed to exceed the time allocated to development of the prototype as Skulpt does not have the libraries needed to support PDB implemented and implementing PDB itself would require implementing over 13000 lines of Python code in JavaScript.

Figure 1: Interactive Code Window

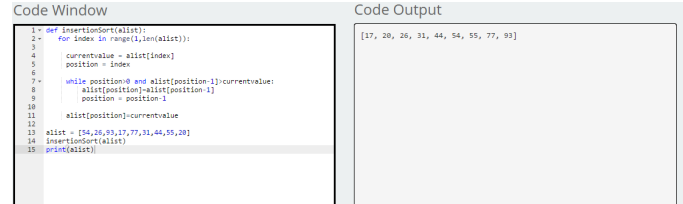


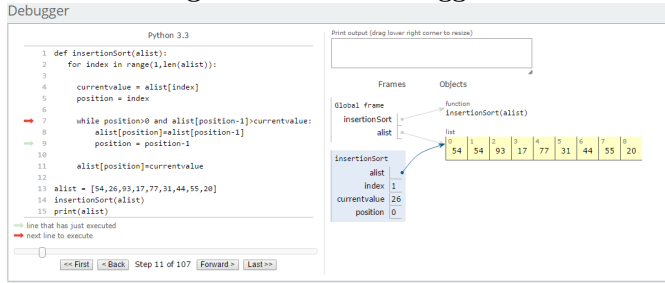
Figure 1 is a snapshot of the interactive code feature that was developed for the prototype. ACE, a web based Python syntax highlighter, was used to provide a space for users to input code. Output was contained in a simple HTML pre element. When users execute the code, it is passed to the controller managing the interactive code section and executed using Skulpt. If the code cannot compile due to syntax errors, the error is displayed in the output pre. If the code successfully executes, the output is displayed.

##### 3.2.2 Online Python Tutor

Online Python Tutor is a web-based program visualization tool that allows students to step through their code and debug programs with the aid of visualizations that represent the state of the program [12]. As it is web based, it requires no installation on the part of the learner. Online Python Tutor was chosen to compensate for Skulpt's lack of a debugger because it allows students to engage with their code by visually stepping through the debugging process. Users' code is passed to a backend server which uses the Python debugging module record the program's runtime state (the contents of the stack and heap) after each line of code is executed [42]. The program trace is the encoded in JSON format by adding additional metadata to the Python data types and serializing them into JSON types. Finally, the JSON data is visualized using the D3 and jsPlumb JavaScript libraries [15]. Online Python Tutor is limited by its use of a backend server. Programs are limited to 300 lines to prevent a bottleneck on the server side [42]. This is not problematic for the short snippets of code that beginner programmers use but it raises questions of scalability. Additionally, a considerable amount of redundant data is stored by Online Python tutor because it saves the contents of the stack and heap after executing each line [15]. The drawbacks of Online Python Tutor were deemed to be manageable given the small scale of the problems that will be addressed using this prototype.

Figure 2 provides a snapshot of the prototype's visual debugging tool. When users wish to debug their code, it is first executed using Skulpt to ensure that there are no syntax errors. If the code can successfully be compiled, a URL to the Online Python tutor page containing the correct code is generated and the debugger is embedded in the page using an iframe.

Figure 2: Visual Debugger



### 3.2.3 Saving Student Progress

A review of literature indicated that students using E-learning systems benefitted from the ability to save their work and resume working on it at a later date. This feature was implemented by adding space in the database for work to be saved to and reloaded from. Upon first login, an entry in the database is added to store user solutions to the questions posed. When users execute their code, the most recent version is saved to the database. These solutions will be retrieved from the database on subsequent logins.

## 4. EXPERIMENT DESIGN

### 4.1 Method

Testing was conducted in a closed lab environment. Upon arrival, participants were asked to read and sign UCT's informed voluntary consent to participate in research form. Participants were then emailed a link to the prototype website, login credentials and instructions for the testing session. To evaluate the prototype, users were asked to complete exercises that had been designed for the CSC1010H class. Users were asked to use typical beginner-level programming exercises to explore the prototype and examine the different features available with the aim of evaluating how effective they would be in aiding beginner programmers. Users were asked to complete at least 2 of the 3 exercises that were presented or spend 45 minutes using the system. Finally, users were asked to complete a survey about their experience which included questions about the effectiveness of the system using a 1-5 likert scale and open ended questions about improvements that could be made. Two rounds of testing were conducted, with a week between the rounds in which improvements were made to the prototype based on the suggestions given in round one of testing.

### 4.2 Participants

This prototype was initially intended to be tested with UCT's CSC1010H class. Instability across South African universities due to Fees Must Fall protest action made access to the class impossible. Instead, the prototype was tested with a group of expert users. UCT honours and masters students with experience tutoring computer science were recruited to evaluate the effectiveness of the prototype developed for this project. To recruit from the honours class, a request for students with tutoring experience was proposed to the entire class. Due to the small number of possible participants available - only a small portion of the class have tutoring experience - opportunistic sampling was used. Students with the correct experience that volunteered to par-

ticipate were chosen to participate. For the first iteration of testing eight students were recruited from the honours class to evaluate the prototype. Of the eight, six had experience tutoring university level computer science and two tutored high school computer science students privately. Recruitment from the masters class was handled by our supervisor. Masters students who have experience as teaching assistants in computer science were chosen to participate in the experiment. The teaching assistant for the CSC1010H class, the original intended participants for this prototype, and a teaching assistant for another first year module were selected as participants. These participants were specifically selected because they have worked closely with a first year computer science class and will offer good insight into the needs of a class of beginner programmers. One masters student and three honours students from the first round of testing volunteered for the second iteration of testing. The three honours students recruited all had experience tutoring university level Computer Science.

### 4.3 Exercise Design

The exercises created for testing the prototype were created by examining the 2016 CSC1010H syllabus and lab exercises that had been completed by previous years classes. Testing was intended to take the form of exam revision for the CSC1010H class so questions were designed to cover the year's work. The questions designed for the CSC1010H class were used for testing with expert users because they provide a representative set of questions that illustrate the challenges that students will face. Three exercises were designed for each of the two rounds of testing. The first round of testing contained exercises that tested string manipulation, the use of if-else statements and type casting. The second round of testing's exercises tested the use of various loops and recursion. Both sets of exercises were examined by our supervisor, the course coordinator of CSC1010H, to ensure that the level of difficulty was appropriate.

### 4.4 Evaluation

The individual features of the e-learning system were evaluated through the use of questions using a 1-5 likert scale. The survey filled in by expert users queried how well each of the features assisted in completion of typical tasks that beginner programmers will perform and the expert's opinion of the suitability of the feature for use in an E-learning system. Assessment of the features included in the prototype was evaluated by examining the individual scores that each feature obtained and by comparing the scores obtained to those of the other features. A feature was considered to be effective if it achieved a favorable score across testing iterations and outperformed the other features implemented in the prototype.

### 4.5 Ethical Issues

Before any testing took place, the appropriate ethical clearance was obtained from the faculty of science and the department of student affairs. All participants signed the appropriate informed consent documentation before proceeding with the experiment. Participants were not intended to be recruited from UCT's honours class. The use of participants who have been classmates with the researchers for the duration of the year raises concerns about bias in the results obtained.



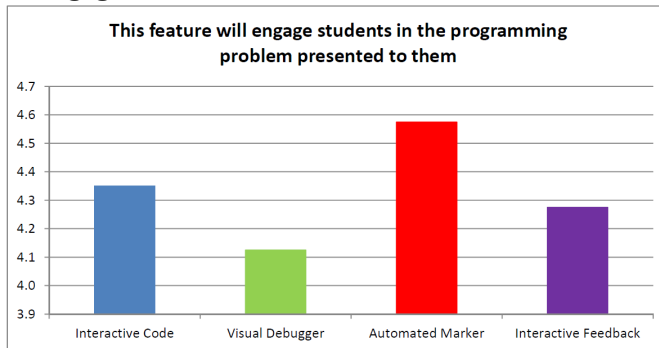
## 5. RESULTS

The expert users were asked to complete a survey evaluating the features provided on their ability to engage students in the task at hand, aid students in detection of syntax and logic errors, work independently at their own pace and suitability of their inclusion in future E-learning systems. The results obtained over two iterations of testing are discussed below.

### 5.1 Ability to Engage Students

In iteration 1, the automated marker scored 4.4/5 for its ability to engage students, with a standard deviation of 0.69. This was the highest score for this category. The interactive feedback feature was second highest, with an average of 4.3/5 and a standard deviation of 0.94. The interactive code feature scored 4.2/5 with a standard deviation of 0.42. The visual debugger obtained the lowest score, 4/5, with a standard deviation of 0.67.

**Figure 3: Expert Evaluation average result: Ability to engage students**

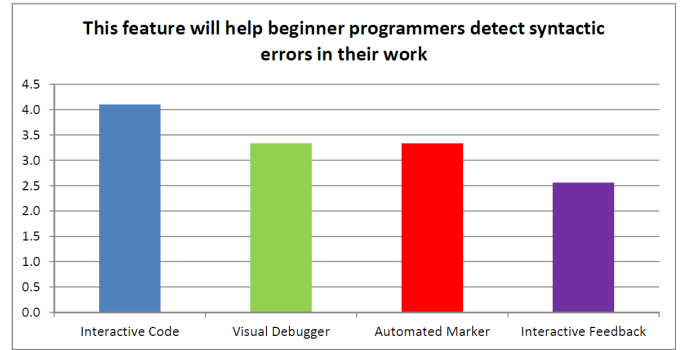


In iteration 2, the automated marker retained the highest score in this category with an average score of 4.75/5 and standard deviation of 0.5. The interactive code feature rose to second highest scoring 4.5/5 with a standard deviation of 0.57. The visual debugger and interactive feedback features tied for last position in this category, both receiving a score of 4.25/5 and standard deviations of 0.95 and 0.97 respectively. Figure 3 illustrates the average scores obtained by each feature across two iterations of testing. The automatic marker achieved the highest average score across iterations 4.6/5 followed by the interactive code 4.4/5, interactive feedback 4.3/5 and visual debugger 4.1/5.

### 5.2 Detection of Syntax Errors

In both rounds of testing, the interactive code feature scored highest in this category with 3.7/5 and a standard deviation of 1.34 in the first iteration and 4.5/5 with a standard deviation of 0.56 in the second iteration. The automated marker placed second in the first iteration of testing with 3.4/5 and a standard deviation of 1.34 and third in the second iteration with 3.25/5 and a standard deviation of 1.71. The visual debugger placed third in the first iteration of testing, scoring 2.9/5 with a standard deviation of 1.37 and second in the second iteration of testing with 3.75/5 and a standard deviation of 0.95. The interactive feedback feature achieved the lowest score in both rounds, 2.6/5 with a standard deviation of 1.2 and 2.5/5 with a standard deviation of 1.29 in the respective rounds.

**Figure 4: Expert Evaluation average result: Ability to detect syntax errors**



The average score obtained by each feature across testing iterations is illustrated in figure 4. The interactive code feature leads with 4.1/5 followed by the visual debugger and automated marker in tied second place with 3.3/5 and the interactive feedback feature achieving the lowest average score, 2.6/5.

### 5.3 Detection of Logical Errors

The order in which the features placed in this category was the same in both iterations of testing. The automatic marker placed first, scoring 4.1/5 with standard deviation of 0.73 and 4.25/5 with a standard deviation of 0.5 in the respective rounds. The visual debugger placed second with 3.9/5 with a standard deviation of 1.19 in the first iteration and 4/5 with a standard deviation of 0.81 in the second iteration. The interactive code feature scored 3.5/5 with a standard deviation of 1.27 and 3/5 with a standard deviation of 1.15 to place third in both iterations. The interactive feedback feature placed fourth with 2.9/5 and a standard deviation of 1.45 and 2.75/5 and a standard deviation of 1.25.

**Figure 5: Expert Evaluation average result: Ability to detect logical errors**

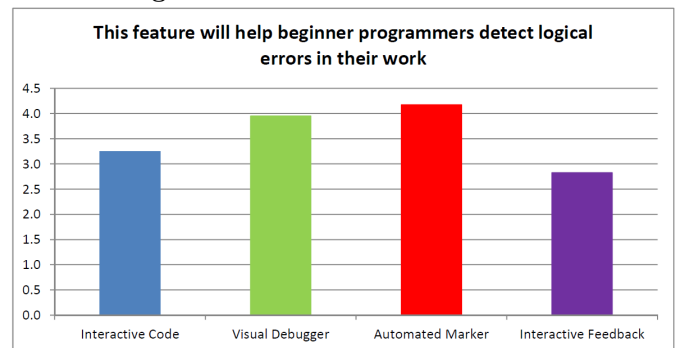


Figure 5 shows the average scores achieved by each feature across testing iterations. It follows that the automated marker placed first with 4.2/5, the visual debugger second with 4/5, interactive code third with 3.3/5 and interactive feedback last with 2.8/5.

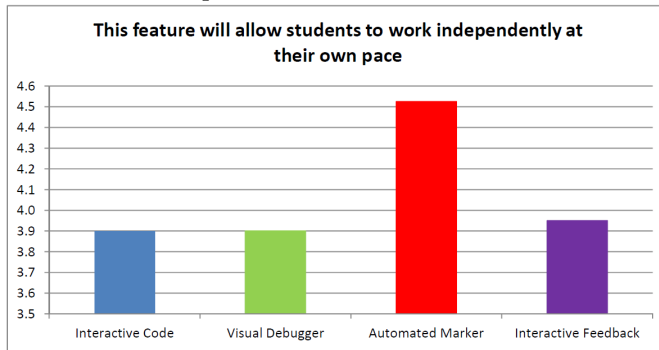
### 5.4 Ability to Work at Own Pace

In iteration 1 of testing, the automated marker scored the highest with 4.3/5 and a standard deviation of 1.1. The



interactive feedback feature placed second with 3.9/5 and a standard deviation of 0.87. The visual debugger and interactive code feature placed joint third with 3.8/5 and a standard deviation of 0.79. The automated marker maintained first placed in the second iteration of testing, scoring 4.3/5 with a standard deviation of 1.05. The visual debugger, interactive feedback and interactive code features scored joint second with 4/5 and a standard deviation of 0.81.

**Figure 6: Expert Evaluation average result: Ability to work at own pace**



The average scored obtained across iterations is illustrated by figure 6. The automated marker scored 4.5/5, the highest average across iterations. The interactive feedback feature placed second with 4/5. The visual debugger and interactive code placed joint third with 3.9/5.

## 5.5 Inclusion in Future Systems

In iteration 1 of testing the automated marker and interactive feedback features placed joint first with 4.7/5 and a standard deviation of 0.48. The interactive code and visual debugger scored joint second with 4.1/5 and a standard deviation of 0.57. In the second iteration of testing the automated marker and interactive code features received the highest score, 4.75/5 with standard deviation of 0.5. The interactive feedback feature placed second with 4.5/5 and a standard deviation of 0.58. The visual debugger placed last with 4/5 and a standard deviation of 0.81.

**Figure 7: Expert Evaluation average result: Inclusion in future systems**

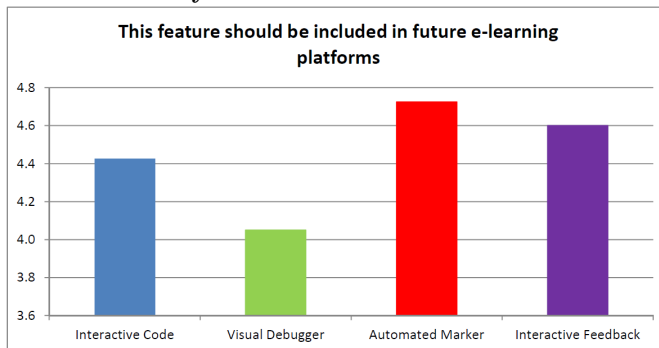
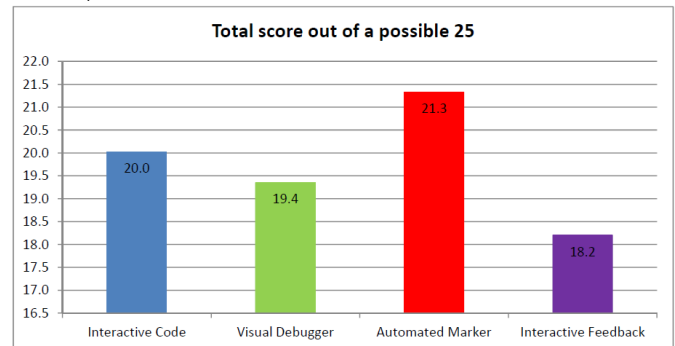


Figure 7 illustrates the average scores achieved by the features over two iterations of testing. The automated marker placed first with 4.7/5, closely followed by the interactive

feedback feature with 4.6/5. The interactive code feature placed third with 4.4/5. Finally, the visual debugger placed fourth with 4.1/5.

**Figure 8: Expert Evaluation results: Total Possible Score /25**



## 5.6 Overall Score

Figure 8 displays the total score that each feature obtained out of a possible 25 points. The automated marker leads with 21.3/25, followed by the interactive code feature with 20/25. The visual debugger closely follows with 19.4/25. Finally, the interactive feedback features placed last with 18.2/25.

## 5.7 Open Ended Questions

Users were also asked to provide feedback on the features provided by the prototype. The main concerns raised about the interactive code feature was that it did not provide many of the beneficial features that an IDE offers - detailed error reporting on and highlighting of syntax errors were commonly mentioned. There were very few concerns about the visual debugger, except for one user who raised concerns that beginner programmers would not understand the visual representation. It was suggested by two participants that the automated marker features should point out the part of the student's code where it is likely that mistakes have occurred. There were no criticisms of the interactive feedback feature.

## 6. DISCUSSION

Across categories, the features provided by the prototype achieved high scores. This is partially to be expected because the features that were included in the prototype were chosen because of their common use in Computer Science E-learning systems. The few exceptions to this statement were to be expected; for example, the interactive feedback feature did scored below 3 for the ability for detect logical and syntax errors. The high scores obtained indicate that evaluating each feature by merely examining the score achieved out of 5 for each category will not provide particularly informative insight into the effectiveness of each feature. It is not sufficient to say that a feature is effective because it achieved a high score out of 5 because a high score is not remarkable in this set of results. Features must therefore be evaluated relative to each other to determine which are most valuable to beginner programmers.

The automated marker out-performed the other features in four of the five categories tested, making it the clear front

runner among the features tested. Of particular interest is its high score for the ability to engage learners, 4.6/5, which confirms the expectation laid out by reinforcement theory that the promise of assessment motivates students to engage with the task at hand [27]. The automated marker scored 0.5 higher than the next best feature in the ability to let students work at their own pace. This is one of the key advantages of using E-learning to deliver content [30]. Suggestions that the automated marker feature should point out areas in the code where students are likely to have made errors are beyond the scope of a simple black box based automated marker. These suggestions emphasize the limitations mentioned in this paper’s literature review of an automated marker’s ability to produce the same quality of feedback that interaction with an instructor can provide.

The use of a simplified notepad-like frame for users to code in was intended to remove the confusion of the unnecessarily complex features that IDEs provide beginner programmers, a known barrier to programming for beginners [12]. Despite obtaining the second highest overall score across categories, the interactive code feature did not perform as well as expected. The open ended questions revealed that the simplicity of the feature detracted from its score rather than adding value, with multiple users raising concerns about the simplistic way in which syntax errors were highlighted and reported. The main concern raised was that this simple feature was not superior to service-rich IDEs. Simple notepad-like frames for executing code have been found to significantly aid learning by Tang (2015), but the frames were only used for execution of pre-loaded code to illustrate a programming concept [42]. The results obtained indicate that a simple code execution environment is insufficient for writing and debugging of code. The ability to execute code is essential to an E-learning system, but a more feature-rich environment appears to be required.

The visual debugger received the lowest score of all the features for the ability to engage learners. This results was not expected; the ability to visually step through code to detect logical errors was expected to engage learners in active learning. It is clear that the experience of this feature is not one in which one actively applies knowledge, but rather one of passive observation. This observation is emphasized by the fact that the visual debugger placed after the automated marker in ability to detect logical errors, despite the fact that the marker only compares expected and actual output and the debugger provides the ability to work through the program in detail. The debate around the effectiveness of visualizations in Computer Science education has not yet reached a definitive conclusion [17]. Failure of this specific visualization to engage learners does not universally exclude visualization as a significant aid to beginner programmers. Constructivist learning theory states that students can create an internal model by actively acquiring knowledge from observing visualizations, the results obtained in this experiment merely indicate that this specific visualization failed to engage users in active learning [3].

The vital role that instructor feedback is known to have in E-learning systems is emphasized by the interactive feedback placing second by only 0.1 in the inclusion in future systems category. This high score is unexpected considering the low score that it obtained in the ability to detect syntax and logical errors categories. It is possible that this feature has been a casualty of the method that this prototype was

tested. Expert users have no need to ask questions about beginner programming tasks. This tool’s potential to aid students having trouble with syntactic or logical errors cannot be genuinely explored by users who are not experiencing said errors.

The results obtained for the interactive code and visual debugger reveal a key flaw in testing a prototype to determine which features are most effective. Users were asked to interact with the specific features implemented, rather than to provide an opinion of that kind of feature in general. For example, users assessed the ability of this specific visual debugger to engage students rather than visual debugging in general. This is in part unavoidable as users must have some tool to test, but must be taken into consideration when reviewing the results of this experiment. This flaw also provides the opportunity for future work to investigate the effectiveness of different implementations of the same feature.

## 7. FUTURE WORK

Future work in determining which components of E-learning systems are most helpful to beginner programmers should aim to test these features with a first year class as was the intention for this prototype. Obtaining usage analytics from a class of students grappling with programming exercises that challenge them will provide great insight into which one of these features assists the learning process.

This prototype focused largely on browser based execution of features. As was seen with the interactive code feature, this objective placed a limitation of what could be provided. Future work in developing and testing a web-based tool which lies between the two extremes a basic code execution frame and a complex IDE could provide interesting insight into the barriers to programming that beginners face in their programming environment. The development of such a tool would provide an interesting investigation into the benefits of the ease of using web based learning environments versus the advantages of using a fully fledged IDE.

The opportunity to test different versions of the same feature also presents itself for visualization of code. The visualization used in this prototype was not found to be effective, but that does not render all visualizations useless. There is still a great amount of controversy around the effectiveness of visualizations in education. Further investigation into the different methods of visualizations available has the potential to provide great insight in this ongoing debate.

## 8. CONCLUSIONS

Of the features tested, it is evident that the automated marker was found to be the most effective in aiding beginner programmers. The interactive code and visual debugging features placed a close second and third respectively, but further work is required to investigate different implementations of these features to truly establish the benefits that they can provide. The interactive feedback feature was highly valued by expert users despite its low score in key categories. This inconsistency is attributed to the method of testing employed. All of the features included in the prototype achieved favourable scores in this experiment. Under performance of features in areas where they were expected to perform well provides insight into areas in which future work is needed.

## 9. REFERENCES

- [1] B. Alshaigy, S. Kamal, F. Mitchell, C. Martin, and A. Aldea. Pilet: an interactive learning tool to teach python. In *Proceedings of the Workshop in Primary and Secondary Computing Education*, pages 76–79. ACM, London, UK, 2015.
- [2] C. Alvarado, B. Morrison, B. Ericson, M. Guzdial, B. Miller, and D. L. Ranum. Performance and use evaluation of an electronic book for introductory python programming. 2012.
- [3] M. Ben-Ari. Constructivism in computer science education. In *Proceedings of the Twenty-ninth SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '98, pages 257–261, New York, NY, USA, 1998. ACM, New York, NY, USA.
- [4] K. G. Brown. Using computers to deliver training: Which employees learn and why? *Personnel Psychology*, 54(2):271–296, 2001.
- [5] R. E. Clark. Reconsidering research on learning from media. *Review of educational research*, 53(4):445–459, 1983.
- [6] T. Cobb. Cognitive efficiency: Toward a revised theory of media. *Educational Technology Research and Development*, 45(4):21–35, 1997.
- [7] D. A. Cook and D. M. Dupras. A practical guide to developing effective web-based learning. *Journal of General Internal Medicine*, 19(6):698–707, 2004.
- [8] S. Cooper, W. Dann, and R. Pausch. Alice: a 3-d tool for introductory programming concepts. In *Journal of Computing Sciences in Colleges*, volume 15, pages 107–116. Consortium for Computing Sciences in Colleges, 2000.
- [9] P. Crescenzi and C. Nocentini. Fully integrating algorithm visualization into a cs2 course.: A two-year experience. In *Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITiCSE '07, pages 296–300, New York, NY, USA, 2007. ACM, New York, NY, USA.
- [10] B. Dorn and D. Sanders. Using jeroo to introduce object-oriented programming. In *Frontiers in Education, 2003. FIE 2003 33rd Annual*, volume 1, pages T4C–22. IEEE, 2003.
- [11] C. Douce, D. Livingstone, and J. Orwell. Automatic test-based assessment of programming: A review. *J. Educ. Resour. Comput.*, 5(3), Sept. 2005.
- [12] S. H. Edwards, D. S. Tilden, and A. Allevato. Pythy: Improving the introductory python programming experience. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 641–646, New York, NY, USA, 2014. ACM, Blacksburg USA.
- [13] T. Govindasamy. Successful implementation of e-learning: Pedagogical considerations. *The internet and higher education*, 4(3):287–299, 2001.
- [14] S. Graham. Skulpt online python interpreter.
- [15] P. J. Guo. Online python tutor: Embeddable web-based program visualization for cs education. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 579–584, New York, NY, USA, 2013. ACM.
- [16] A. J. Hartz. *CAT-SOOP: A tool for automatic collection and assessment of homework exercises*. PhD thesis, Massachusetts Institute of Technology, 2012.
- [17] J. Helminen and L. Malmi. Jype-a program visualization and programming exercise tool for python. In *Proceedings of the 5th international symposium on Software visualization*, pages 153–162. ACM, New York, NY, USA, 2010.
- [18] C. D. Hundhausen, S. A. Douglas, and J. T. Stasko. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages & Computing*, 13(3):259–290, 2002.
- [19] D. Jackson and M. Usher. Grading student programs using assyst. In *ACM SIGCSE Bulletin*, volume 29, pages 335–339. ACM, California, USA, 1997.
- [20] W. Jin, L. Lehmann, M. Johnson, M. Eagle, B. Mostafavi, T. Barnes, and J. Stamper. Towards automatic hint generation for a data-driven novice programming tutor. In *Workshop on Knowledge Discovery in Educational Data, 17th ACM Conference on Knowledge Discovery and Data Mining*. Citeseer, 2011.
- [21] M. Joy, N. Griffiths, and R. Boyatt. The boss online submission and assessment system. *J. Educ. Resour. Comput.*, 5(3), Sept. 2005.
- [22] S. Kalyuga. Enhancing instructional efficiency of interactive e-learning environments: A cognitive load perspective. *Educational Psychology Review*, 19(3):387–399, 2007.
- [23] C. Kelleher and R. Pausch. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys (CSUR)*, 37(2):83–137, 2005.
- [24] M. Klassen. Visual approach for teaching programming concepts. In *Proceedings of the 9th International Conference on Engineering Education (ICEE 2006)*, pages 23–28, 2006.
- [25] R. B. Kozma. Will media influence learning? reframing the debate. *Educational technology research and development*, 42(2):7–19, 1994.
- [26] E. Lahtinen, K. Ala-Mutka, and H. M. Järvinen. A study of the difficulties of novice programmers. In *ACM SIGCSE Bulletin*, volume 37, pages 14–18. ACM, 2005.
- [27] K. M. Y. Law, V. C. S. Lee, and Y. T. Yu. Learning motivation in e-learning facilitated computer programming courses. *Computers Education*, 55(1):218–228, 8 2010.
- [28] S. Liu. Faculty use of technologies in online courses. 2005.
- [29] J. J. Martocchio. Effects of conceptions of ability on anxiety, self-efficacy, and learning in training. *Journal of applied psychology*, 79(6):819, 1994.
- [30] D. R. McIntyre and F. G. Wolff. An experiment with www interactive learning in university education. *Computers Education*, 31(3):255–264, 11 1998.
- [31] R. Michaelson. Does e-learning work? 2002.
- [32] B. Miller and D. Ranum. Beyond pdf and epub: Toward an interactive textbook. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '12, pages 150–155, New York, NY, USA, 2012. ACM,

New York, NY, USA.

- [33] M. G. Moore and G. Kearsley. *Distance education: A systems view of online learning*. Cengage Learning, 2011.
- [34] T. L. Naps, G. Rößling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S. Rodger, et al. Exploring the role of visualization and engagement in computer science education. In *ACM Sigcse Bulletin*, volume 35, pages 131–152. ACM, New York, NY, USA, 2002.
- [35] R. F. J. North, D. M. Strain, and L. Abbott. Training teachers in computer-based management information systems. *Journal of Computer Assisted Learning*, 16(1):27–40, 2000.
- [36] U. of Cape Town Science Faculty. Extended degree programme, 2016.
- [37] J. M. O’Hara. The retention of skills acquired through simulator-based training. *Ergonomics*, 33(9):1143–1153, 1990.
- [38] A. Pears, S. Seidman, L. Malmi, L. Mannila, E. Adams, J. Bennedsen, M. Devlin, and J. Paterson. A survey of literature on the teaching of introductory programming. In *Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education*, ITiCSE-WGR ’07, pages 204–223, New York, NY, USA, 2007. ACM, New York, NY, USA.
- [39] D. Pritchard and T. Vasiga. Cs circles: An in-browser python course for beginners. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE ’13, pages 591–596, New York, NY, USA, 2013. ACM, New York, NY, USA.
- [40] S. H. Rodger. Introducing computer science through animation and virtual worlds. In *ACM SIGCSE Bulletin*, volume 34, pages 186–190. ACM, New York, NY, USA, 2002.
- [41] T. Sitzmann, K. Kraiger, D. Stewart, and R. Wisher. The comparative effectiveness of web-based and classroom instruction: A meta-analysis. *Personnel psychology*, 59(3):623–664, 2006.
- [42] L. Tang. A browser-based program execution visualizer for learning interactive programming in python, 2015.
- [43] E. T. Welsh, C. R. Wanberg, K. G. Brown, and M. J. Simmering. E-learning: emerging uses, empirical results and future directions. *International Journal of Training and Development*, 7(4):245–258, 2003.