

CodeWorkout: Design and Implementation of an Online Drill-and-Practice System for Introductory Programming

Krishnan Panamalai Murali

*Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of*

Master of Science
in
Computer Science and Applications

Stephen H. Edwards, Chair
Manuel A. Pérez-Quinones
Clifford A. Shaffer

April 20, 2016
Blacksburg, Virginia

Keywords: computer science education, programming tool, web application,
drill-and-practice

Copyright 2016, Krishnan Panamalai Murali

CodeWorkout: Design and Implementation of an Online Drill-and-Practice System for Introductory Programming

Krishnan Panamalai Murali

(ABSTRACT)

The massive rise in Computer Science enrollments in both traditional classroom courses and in Massively Open Online Courses (MOOCs) shows the enormous opportunities in engaging students to learn programming. While the number of students in CS courses continues to increase, there has been no concomitant increase in the number of instructors for such courses. This leads to a completely lopsided learning environment where the already-stretched instructor is pressed to spend more time on ancillary tasks like grading and course bookkeeping.

CodeWorkout is an online drill-and-practice system with course management features that aims to address these issues. CodeWorkout hosts an online repository of programming questions that instructors can incorporate into their courses. It also provides instructors with a facility to create their own programming questions so that exercises can be tailored according to the needs of the class. CodeWorkout has an open gradual engagement model that allows students who are not enrolled in a course to use it. CodeWorkout also creates an open environment for instructors to collaborate by sharing exercises that they create.

CodeWorkout has been used in four courses at Virginia Tech. It has been shown to significantly improve the student's skills in introductory programming through providing a number of online practice questions.

CodeWorkout: Design and Implementation of an Online Drill-and-Practice System for Introductory Programming

Krishnan Panamalai Murali

(PUBLIC ABSTRACT)

The massive rise in Computer Science enrollments in both traditional classroom courses and in Massively Open Online Courses (MOOCs) shows the enormous opportunities in engaging students to learn programming. While the number of students in CS courses continues to increase, there has been no concomitant increase in the number of instructors for such courses. This leads to a completely lopsided learning environment where the already-stretched instructor is pressed to spend more time on ancillary tasks like grading and course bookkeeping.

CodeWorkout is an online drill-and-practice system with course management features that aims to address these issues. CodeWorkout hosts an online repository of programming questions that instructors can incorporate into their courses. It also provides instructors with a facility to create their own programming questions so that exercises can be tailored according to the needs of the class. CodeWorkout has an open gradual engagement model that allows students who are not enrolled in a course to use it. CodeWorkout also creates an open environment for instructors to collaborate by sharing exercises that they create.

CodeWorkout has been used in four courses at Virginia Tech. It has been shown to significantly improve the student's skills in introductory programming through providing a number of online practice questions.

Acknowledgments

Our lives are not our own and the works of our hands much less so. Similarly, this work is the result of contributions from various individuals. First and foremost, I would like to acknowledge my advisor, Dr. Stephen Edwards, who is the primary force behind CodeWorkout. He has been the most important figure in my two years of graduate studies and without him I could not have reached where I am. He is also the most technically skilled person I have met at Virginia Tech and I have never ceased learning from him. Finally, I would like to thank him for all the guidance and financial support that he has shown towards me throughout my studies. I would like to acknowledge Dr. Manuel Pérez-Quñones for all his insights and words of support. I would like to thank him for reiterating to me the importance of having the end user in mind while designing this system. I would like to acknowledge Dr. Clifford Shaffer for his incisive comments regarding CodeWorkout.

I would like to thank my family members in India who are the blood of my own blood.

I would like to thank my schoolmates, Hem Vignesh, Manoj, Manichandar, Narayanan, and Vignesh, for reminding this noob that the most important aspect of life is *living*.

I would like to thank Soumik Ghosh and Subhodip Biswas (Leo), my companions at Virginia Tech through many trials and tribulations, without whom Blacksburg would have been greyer than it already is.

I would like to thank the Virginia Tech Writing Center for helping me fix the many errors in earlier drafts of this document.

This work is supported in part by the National Science Foundation under grant DUE-1245589. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

Last but certainly not the least, I would like to thank my roommates cum caretakers, Bharat, Pavan, and Siddhu, without whom I would not have survived the transition to Blacksburg from Chennai.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Background	3
1.3	Enter CodeWorkout	5
1.4	Problem Statement	6
1.5	Terminology	7
1.6	Organization of Content	7
2	Literature Review	8
2.1	Related Work	8
2.1.1	Forerunners	8
2.1.2	CodeWorkout’s inspirations	9
2.1.3	Current Drill-and-Practice Systems	10
2.1.4	Path to extensive use	12
2.2	Summary of key differentiators	13
3	CodeWorkout Design Goals	15
3.1	Target: Gradual engagement	16
3.2	Target: A flexible yet secure permissions model	17
3.3	Target: Enhanced drill-and-practice features	18
3.4	Target: Course versus Gym separation	20
3.5	Target: Exercise organization	20

3.6	Target: Collaborative and active learning	21
3.7	Additional considerations	22
4	Implementation	25
4.1	Architecture	25
4.2	Facilitating Gradual Engagement	28
4.3	Facilitating a Flexible Permission Model	31
4.4	Facilitating Enhanced Drill-and-Practice Features	33
4.5	Facilitating Gym versus Course Separation	33
4.6	Facilitating Exercise Organization	36
4.7	Facilitating Collaborative and Active Learning	38
4.8	Towards a more complete system	39
5	Evaluation and Analysis	43
5.1	Analysis of attempts	44
5.1.1	Number of attempts vs. Final score	45
5.1.2	Number of completed exercises vs. Exam score	45
5.2	Analysis of a practice workout	47
5.2.1	Relationship between practice workout performance and exam performance	49
5.3	Analysis of MCQs vs. Coding exercises	52
6	Survey Results	55
6.1	Survey method	55
6.2	Likert-Scale Question Results	56
6.2.1	Results from Likert-Scale Questions	56
6.3	Free-form Text Question Results	59
6.3.1	Impressions	59
6.3.2	Impediments	60
6.3.3	Overall usefulness	62

7 Conclusion	64
7.1 Contributions	64
7.2 Future Work	65
Bibliography	67
Appendix A Text of Survey	71
Appendix B Survey Report	75
Appendix C Virginia Tech IRB Approval	86

List of Figures

3.1	Support for IRT statistics	24
4.1	CodeWorkout’s Architecture	26
4.2	CodeWorkout’s Gym—An area of open practice exercises1	28
4.3	Student search for exercises on conditionals	29
4.4	User authentication using CAS	30
4.5	Student enrolling in an open course	30
4.6	Authentication options	31
4.7	Course Enrollments	32
4.8	Student review of past exercises	34
4.9	Autograder Feedback	34
4.10	CodeWorkout data model support for scoring	35
4.11	A CodeWorkout MCQ exercise	36
4.12	A CodeWorkout multi-part question	37
4.13	Workout and exercise thumbnails	40
4.14	Instructor accessing course gradebook	41
4.15	Instructor reviewing student performance on a workout	41
5.1	Number of attempts on an exercise versus final score	45
5.2	Statistics for the analysis of final exercise score against number of overall attempts and number of completed exercises	46
5.3	Regression between the CodeWorkout Test 2 score and the number of perfectly completed exercises	47

5.4	Regression between the scores on CodeWorkout exercises in the two tests for Practice Group 1	48
5.5	Regression between the scores on CodeWorkout exercises in the two tests for Practice Group 2	49
5.6	Attempts on an exercises and perfectly completed exercises versus final score	50
5.7	Performance of the two practice groups on the multiple-choice questions in the two tests	51
5.8	Relation between the number of perfectly completed practice exercises and Test 2 score	51
5.9	Relation between the number of perfectly completed practice exercises and Test 1 score	52
5.10	Regression between the scores on CodeWorkout exercises and Canvas MCQs in the two tests	53
6.1	Prior programming experience of students	56

List of Tables

2.1	Comparison of CodeWorkout with other related systems	13
5.1	CodeWorkout usage in CS@VT	43
5.2	CodeWorkout’s use in Virginia Tech’s Spring 2016 CS 1114 course for different workouts	44
6.1	Likert-survey questions means and variances	58

Chapter 1

Introduction

Herein we give the motivation for CodeWorkout, as well as background information about automated drill-and-practice systems in Computer Science Education. We later introduce CodeWorkout by mentioning its key features and mention the exact problem statement that gave rise to its development. We also delineate some of the terminology used in this document.

1.1 Motivation

Many students find it hard to acquire programming skills. There are various concepts involved in programming and many of these have to be employed in tandem to solve a particular problem. Traditional paper-based assignments require programs to be written down on paper by students. This is inadequate for two reasons: the environment on which the code written is not an actual programming environment and students receive feedback only once (after all the assignments are turned in). Unlike paper-based coding assignments, real-life programming is done on computers where the compilers point out syntax and semantic errors. Programmers also use tools like Integrated Development Environments (IDEs) that help in syntax highlighting and provide warnings to fix some basic coding errors. Also, quick feedback enables students to identify the mistakes they have made and correct these mistakes themselves. Novice programmers need extensive practice and feedback on each and every topic. Extensive practice enables them to gain a mastery of fundamental topics like the writing of loops and conditionals. Multiple attempts, especially in homework assignments, enables them to practice on an exercise as much as they want to. This also enables different students to learn at their own pace since not all students will take the same number of attempts to solve an exercise. Such features cannot be easily provided in traditional paper-based systems but are feasible in digital autograder systems, especially drill-and-practice systems. These systems allow students to practice programming extensively by offering

them a large number of small exercises on various concepts.

Another problem of paper-based systems from the perspective of instructors with regard to drill-and-practice is the burden that it imposes on them. Drill-and-practice systems, by their very nature, involve multiple exercises to be solved by students on a single topic. Since there are many concepts in programming, there will be a large number of exercises on each of these topics for an instructor to grade and to provide feedback on. This detracts instructors and any associated teaching assistants from the actual task of teaching students. This also imposes bookkeeping overheads on instructors since they need to keep track of the many assignments for every student and how these count towards the final grade of the student. Automated systems can take care of all these tasks for the instructor and allow them to focus more on their actual task of teaching. Thus, instructors also stand to reap many benefits through the use of automated drill-and-practice systems despite the initial effort that is necessary on behalf of instructors to set up such systems for their course. The increasing enrollments in Computer Science indicate that drill-and-practice systems will continue to be useful to instructors.

In her keynote speech at SIGCSE 2016, Dr. Jan Cuny, the NSF Program Director of Computing Education, spoke about ‘Catching the Wave’ [8]. The wave in this context is the massive surge in the number of enrollments seen currently in introductory Computer Science courses across the United States both at undergraduate and high school level. Enrollments in CS have seen tremendous fluctuations over the past thirty years. More specifically there was an upswing in CS enrollments in the 1980s that was immediately followed by a precipitous reduction. The same trend again was observed at the end of the previous millennium with an even more drastic fall in enrollments coming after the bursting of the dot-com bubble. Then in 2007 came the two great American comebacks of the current century: the Ford renaissance under its CEO Alan Mulally and the ‘Third Wave’ in the growing enrollments in CS courses across the country. Skeptics initially wrote off this surge countering that this is just another ‘bubble’ in CS enrollments. Yet, it is now almost a decade since the wave got going and the numbers are still strong in favour of its continuance.

Dr. Cuny also spoke about how the trend of increasing enrollments is especially visible in undergraduate institutions. Then she asked a thought-provoking question that is highly relevant to the topic of this thesis: how can instructors in these universities provide the same quality of computer science education to an increased quantity of students? It is an interesting question that requires an immediate answer lest we risk ‘missing the wave’. The comprehensive Taulbee 2014 Report [47] expounds particularly on enrollment data from universities. The overall enrollment in undergraduate universities for computing degrees across numerous CS departments in the country was shown to have continuously increased for the past half a dozen years, with an increase of about 22 percent just over the past two years. These increased enrollments have led to more bachelor degrees in computing. However, the trouble here is that the increase in the number of bachelor degrees awarded has not kept up with the increase in enrollments and this increase hovering around ten percent. This means that there is a lacuna in keeping students engaged in CS courses at the

university level.

Another factor is that even though more doctoral degrees have been awarded in Computer Science than ever before [47], the increase in computing Ph.Ds (3.2%) is small compared to the rise in undergraduate CS enrollments (13.4%). What this effectively means is that even though more and more students are signing up for CS courses at the undergraduate level, there are simply not enough new CS faculty members with terminal degrees to teach them. This shortage is aggravated by the fact that most of the new doctoral graduates enter into the industry after their graduation instead of going back into academia. The figures from the CRA report [47] highlight this scenario: Only 12.8 percent of those graduating with a doctoral degree in computer science go into any sort of career where they can teach it and less than 15 percent of new graduates go on to do a postdoc. Of course, not all of these new teachers will become instructors in introductory programming courses. Similarly, not all of these postdocs will get a teaching position suited to them.

Thus, the trends in CS undergraduate and graduate education indicate there will not be enough qualified instructors to teach these incoming students. The problem is that even ‘purely teaching’ faculty cannot spend an overwhelming percentage of their time in teaching because they have extensive administrative and grading commitments. The bulk of the grading work in CS undergraduate courses involves the grading of student code or MCQs because most of the coursework therein deals with the learning or application of programming concepts, which are time-consuming to evaluate manually. Thus, automated drill-and-practice systems are becoming more and more relevant in introductory programming courses.

Such systems are useful to high schools and not just universities alone. High schools that offer introductory programming courses generally cannot afford (or sometimes lack the expertise and administrative wherewithal) to install and run autograders on their own servers. High school instructors then resort to testing the programming ability of students by using paper-based exams and then manually going over their code. Code writing on paper also poses a significant difficulty for instructors because high schools generally lack the aid provided by grading assistants and hence will have to spend a significant amount of their teaching time in not teaching. Thus in both universities and high schools, there is a need for online drill-and-practice systems with automatic feedback where students can be offered a large number of shorter programming exercises that allow for multiple attempts.

1.2 Background

The role and use of automation in undergraduate level CS courses have been studied by Wilcox [45]. He states that there are many benefits to using autograding in such courses, primarily resource savings. He shows that even for a modest class size of 250 students with very few, small assignments that will take about five to ten minutes to manually grade per student, would cost \$4000 just for TAs to grade them. This is not accounting for the other

administrative tasks of re-grading and grade bookkeeping. Of course, all this manual grading only detracts both the instructors and TAs from the actual task of teaching. Students who approach TAs during their office hours spend most of the time asking about their assignment grades rather than trying to gain better grasp of programming concepts.

Even more illuminatingly, autograders were found to enable the assigning of multiple smaller questions that would otherwise be untenable with manual graders. He has stated that autograders impede neither the interest nor the academic performance of the students but were rather found to increase them if used appropriately. Online autograding with fast feedback was also shown to have alleviated the problem of lab overcrowding, i.e. students who are able to solve the programming questions on their own need not crowd the lab. Instead, the students who are having real difficulties with the programming questions can attend the lab and avail the assistance offered by the TAs. Another way that online autograders were found to ameliorate lab-overcrowding was its 24x7 availability since it enables students to learn at their own pace because students now have the option of working on their exercises whenever they want.

Autograding alone might not keep this large influx of students into computer science engaged. Per Bloom's taxonomy of learning [43], autograding of multiple-choice questions can lead only to the evaluation of lower-order thinking skills of remembering and understanding. Similarly, the autograding of programming exercises can lead only to the evaluation of the 'Apply' skills per Bloom's Taxonomy. On the other hand, if a student after mastering a particular concept can create his or her own programming question for that concept then it shows that that student has gained a higher-order thinking skill. CodeWorkout aims to provide such a facility to students since it enables them to create their own questions. It has the feature of tagging each question/assignment with all the concepts involved in them. Since the system can track the scores of students on a concept-by-concept basis, once it identifies a student who has gained a sufficient mastery of the concept it can grant him or her the option of creating a question on the same concept that can be shared with the rest of the class for practice after it has been curated by the instructor or teaching assistants.

To explore the viability of a 'student-sourced' approach to creating practice question banks, the 2015 study by Denny, Cukierman and Bhaskar [10] performed an empirical validation of the efficacy of student-created programming exercises in boosting learning. The study involved a set of randomly selected students who had to create a set of programming questions before a summative exam while the rest of the class did not have to. The study found that students who partake in the activity of creating viable practice programming questions achieve better grades. It was also shown that students were capable of producing exercises across a wide range of topics and difficulties.

1.3 Enter CodeWorkout

CodeWorkout is an online drill-and-practice system that is designed to implement the ideas initially presented in [5]. It is a completely online and open system for all those who are interested in teaching programming to their students. Thus, CodeWorkout is not limited to any particular university or any other educational body. It aims to provide an universal platform on which students from various backgrounds can practice programming and instructors can offer courses. CodeWorkout, is of course, not limited to short single-method programming questions but is capable of supporting different kinds of questions like full class body implementation or multiple-choice questions. CodeWorkout is completely open-source and its source code with installation instructions can be found on GitHub [16].

CodeWorkout facilitates active learning through both its drill-and-practice approach and the facility for students to create their own questions which they share for the benefit of their peers. CodeWorkout also provides a facility for instructors to share their programming exercises with other instructors if they wish. Most of the introductory CS courses involving programming cover a similar set of topics and hence the exercises involved therein would be similar too. Instead of the same kind of exercises being reinvented over and over by different instructors, an instructor using CodeWorkout can utilize a pre-existing exercise that is publicly available in the system, which is different from simply sharing the question text.

As mentioned earlier, CodeWorkout is an open system that is free to use by everyone everywhere. Thus anyone with a browser and internet connection on their computer can declare themselves to be an instructor, create their course on CodeWorkout and have their students enrolled in it. CodeWorkout can be used as a completely online solution and so there is no installation overhead for instructors to incorporate it into their traditional courses. Neither students nor instructors necessarily need to be tied to the infrastructure of their institutions or be shackled to a particular Learning Management System (LMS). Thus, any instructor who wants to use CodeWorkout can easily set up a course and get his or her students started on it.

CodeWorkout is also suited to be utilized on a massive scale similar to how courseware is offered in MOOCs. For example, MOOCs that teach Introduction to Java can potentially use CodeWorkout to evaluate the programming assignments of thousands of students per offering. In order for MOOCs focused on programming to flourish, it is imperative to have an autograding system that can operate at a similar scale and enable its students to learn collaboratively. There exists no other programming drill-and-practice system that offers similar features and flexibility. CodeWorkout can also be used in various settings like exams, assignments and general programming practice. Hence, it can collect information about student performance in various scenarios and this offers tremendous possibilities in CS Education research through data mining. Perhaps the most important facet of CodeWorkout is that it hosts under one roof the best features of various other drill-and-practice systems while

adding its own unique features to offer a comprehensive platform for programming practice.

1.4 Problem Statement

This work aims to address some of the problems faced by introductory programming courses in various settings. These include non-teaching tasks like grading and assignment management. We aim to do this through the design and development of CodeWorkout, a new online drill-and-practice system for enhancing introductory programming. Its key design aspects towards this goal are:

- **Providing a gradual engagement model:** The gradual engagement model lets both students and instructors use the system at various levels of engagement. This enables *all potential* students and instructors of programming to use this at their required level of engagement. Thus, students and instructors, unhindered by institutional affiliation or lack thereof can benefit from the system.
- **A flexible permission model:** To facilitate its usage in various educational settings for a large and diverse set of users, CodeWorkout provides a robust but flexible permission model to regulate the actions of users on the system.
- **Enhanced drill-and-practice features:** CodeWorkout provides enhanced features over conventional drill-and-practice systems like a growing public repository of practice exercises and compartmentalized attempts.
- **Simultaneously supporting course and Gym exercises:** CodeWorkout is designed to support classroom for graded assignments. Additionally, CodeWorkout provides an open “free practice” arena where anyone, whether enrolled in a course or not, can practice publicly available exercises. The same exercise can be used simultaneously in both contexts.
- **Exercise Organization:** CodeWorkout provides various features and facilities around exercises like versioning, sharing and families that are lacking in current drill-and-practice systems.
- **Collaborative and active learning:** To increase the engagement of students in courses, CodeWorkout’s framework facilitates features such as student-created course content like programming exercises. Its framework also provides a facility for instructors to share assignments across courses.

1.5 Terminology

We present the terminology that is specific to CodeWorkout. Since CodeWorkout is fundamentally a drill-and-practice system for coding questions, we have gone with a catchy naming convention. In CodeWorkout, coding questions are called exercises and a collection of related exercises are called a ‘workout’. An instructor also has a form to create a new exercise or workout if they deem necessary. A workout is the equivalent of a programming assignment that is given over the space of a week or so. Instructors who want to share a particular workout with everyone else using CodeWorkout simply need to mark it as a public workout. This makes the workout available in the common repository that enables it to be practiced by everyone. This area is appropriately called the ‘Gym’. The workouts in the Gym can also be used in any course offering that needs it. The exercises in CodeWorkout can be of different types. The most commonly supported types are multiple-choice questions (MCQs) and coding questions.

1.6 Organization of Content

Chapter 2 introduces the different approaches to autograding and student-contributed content undertaken by other CS Educators in the past and how CodeWorkout differs from them. Chapter 3 expounds more on the objectives of CodeWorkout and how it is designed to address those objectives. Chapter 4 goes into some of the implementation details of CodeWorkout. Chapter 5 explains the evaluation of the system and provides the various data-oriented evaluations that were carried out with the data collected by the system, showcasing its felicity for analysis of student work. Chapter 6 discusses the result of the survey given to students to elicit their opinions of CodeWorkout and its efficacy. Chapter 7 concludes this thesis and discusses the possible future extensions to this work.

Chapter 2

Literature Review

In this chapter, we introduce other that are similar to CodeWorkout in spirit, including early pioneers and CodeWorkout’s own inspirations. In the end of this chapter, we summarize the main difference between CodeWorkout and related extant systems.

2.1 Related Work

Throughout the past several decades there have been multiple systems for the autograding of student programs, each for either a different set of students or a different set of institutions. Hence they have had different objectives and their capabilities were dictated by the technologies of their time. Here we explain previous drill-and-practice systems and some Automatic Assessment (AA) systems for program grading, beginning with early forerunners, then mentioning some of CodeWorkout’s inspirations and current competitors. This section ends with a discussion on how CodeWorkout can be widely adopted and used at scale.

2.1.1 Forerunners

The forerunner of all autograding systems is the work done by Hollingsworth in 1960 [24]. He explains how his course benefited from using an autograder because of the significant time savings accrued from the use of it compared to its development costs. Of course, being first used in the late 1950s, its grading was done on a now obsolete IBM 650, but this paper remains remarkable for the kind of autograder issues it discusses which are still relevant today. This prescient paper even mentions how “the finest teaching machine of all” in the future would be computers. One of the other seminal works in this area has been ASSYST [28]. This early autograder provides features like code checking not only based on correctness but on aspects like efficiency (based on the number of statement executions by

a student's code as compared to the instructor's), complexity (based on the controversial McCabe's metric) and code coverage of student provided tests.

2.1.2 CodeWorkout's inspirations

One cannot talk about autograding systems without mentioning the most widely used one that is publicly available: Web-CAT [14]. The Web-based Center for Automated Testing is still the most widely used open-source autograding system available and has been used in more than a hundred institutions around the world. Web-CAT has been used in a wide variety of programming courses and is not limited to introductory programming courses involving conventional programming languages like Java or Python. Web-CAT [15] can either be downloaded, installed and run on a local server or can be used as a completely hosted solution from Virginia Tech's instance. Web-CAT works utilizes student-written tests to grade their submissions. It is a free and open-source system that can support most forms of programming like functional programming or logic programming. It allows for the submission of code in different languages like Java, Python, Haskell, and Prolog. Web-CAT can be modified according to the requirements of a course and has a plug-in style architecture that enables instructors to add in the features that they want.

CodeWorkout's architecture is based on Web-CAT's architecture: submit code through a browser to a central server that evaluates each student code separately in its own sandboxed environment. Both Web-CAT and CodeWorkout perform the grading of student code through referential testing against the instructor provided test cases. However, there are some significant differences between these two systems. The most obvious difference is that CodeWorkout is a drill-and-practice system while Web-CAT is not. Also, Web-CAT requires student-written tests in its main goal to teach students test-driven development. CodeWorkout is for smaller programming exercises, sometimes involving just the writing of a single method or completing the definition of a class. Hence, CodeWorkout is more fine-grained and is easier for instructors to set up their assignments and novice programmers to get started with. Web-CAT is aimed more towards larger assignments that require a few weeks to finish. CodeWorkout can provide hints and feedback at the granular level of test cases. CodeWorkout is also aimed at providing exercises at the scale of MOOCs which Web-CAT is not. CodeWorkout's gradual engagement model and social collaboration features for both students and instructors are not available in any other AA system, including Web-CAT.

One of the other systems that CodeWorkout is based on is CodingBat [36]. Both are online code drill-and-practice systems that offer an open repository of programming exercises. It was developed by Nick Parlante at Stanford and expects students to complete short methods based on the logic required for that question. CodingBat, just like CodeWorkout, is also both available on its parent website as a cloud offering or can be installed and run on an institute's server like Virginia Tech's CodingBat instance [7]. Unlike CodeWorkout, CodingBat does not support exercises involving non-primitive data types. For example, CodeWorkout allows

the testing of exercises involving lists and maps while CodingBat does not. CodingBat also does not provide for exercises involving more than one method or involving the writing of an entire class. Though CodingBat is free to use, it is not an open-source system and hence cannot be extended to meet institutional requirements or research purposes. While CodingBat does provide an open repository of practice exercise, it does not offer a gradual engagement model since users who log in do not get access to courses or additional material. CodingBat also has the drawback of having a very basic coding interface for students to use to submit their code and does not actually record the various student attempts.

CodeWorkout's facility of enabling students to author their own exercises so that it can be used for practice by their peers is an idea derived from PeerWise [11]. PeerWise is a web-system that provides a repository of MCQs where the questions and associated explanations can be written by students in a course. These questions are given for practice to their peers to assist their learning. Students can give ratings for the exercises created by their peers. It was observed that the questions that were given higher ratings were either conceptually different from the rest or had very detailed explanations associated with them. This suggests that students who can come up with such questions have developed higher-order skills in their course. PeerWise is inherently not a code autograding system and deals only with MCQs.

2.1.3 Current Drill-and-Practice Systems

Denny and Luxton-Reilly, two of the authors of PeerWise, have created another system called CodeWrite [12]. It combines both the facility of letting students in a course create their own programming exercises to be shared with their peers, and the concept of drill-and-practice. Exercise creators specify a set of test cases for their questions and must provide a working implementation for it in Java. It also enables peer review in the sense that once a student has perfectly solved a programming exercise he or she is given a chance to critique the solutions of peers. While interesting, CodeWrite suffers from some significant drawbacks that CodeWorkout does not. CodeWrite is heavily based on the CodingBat model and hence is restrictive when it comes to what kind of questions can be authored. It only allows for single method questions like CodingBat does. Worse still, it only allows for a maximum of five parameters to be specified in the method and only ten test cases per method. The data types of variables that can be used in the method signature are limited to primitives, Strings and arrays. CodeWrite only allows for exercises written in Java unlike CodeWorkout which can facilitate all major programming languages including Java, Python and Ruby. CodeWrite is a completely in-house system in the University of Auckland and hence currently cannot be installed or used outside of it.

Marmoset [40] is a drill-and-practice system with the objective of encouraging students to start working on their programming exercises early and to test their own code. It achieves this by the simple concept of release tests, which is one of the four categories of test cases

that student code is evaluated on. Once a student passes all the public test cases, they are given the option of running their code against the release test cases. These release test cases are available for only a few attempts. Once a student has exhausted his or her available attempts, then he or she must wait for some specified amount of time until they are re-generated. This gimmick encourages students to start their exercises early so that they will have more attempts overall. Marmoset makes students test their own code before submitting so as to not waste precious attempts. It is also a web-based system where students submit their code online and is flexible enough to work with assignments in diverse languages such as C and Ruby. Just like CodeWorkout, Marmoset also makes available to the instructor snapshots of the student code at various stages of completion through a CVS repository.

CodeWorkout differs from Marmoset because of the collaborative features it offers. It enables students to create their own programming exercises that can be shared with their peers. Instructors can also share exercises across courses and institutions. Marmoset is not a completely online system in the sense that it can only be used if it is installed on a university's server. Its setup is cumbersome because it requires the setup of two different servers, a build server and the submit server, as well as a csv repository. For these reasons, Marmoset poses a significant barrier to adoption for a lot of universities and most high schools.

A related system, that is also heavily influenced by CodingBat, is CloudCoder [35], which is also an open-source web-based system that is free to use for institutions. While there is not much to differentiate CloudCoder from other existing systems (other than the fact that it is open-source), it does allow the sharing of questions between instructors of different institutions. Apart from this, CloudCoder is essentially the same as CodingBat in that student code is submitted through a web-interface to the institution's server. The server runs the student code against the instructor-defined test cases and the results are displayed as feedback to the student.

CloudCoder has many limitations that CodeWorkout hopes to improve upon. CloudCoder only supports two kinds of exercises, viz., function-based exercises (where the student completes a single method that returns a specific value) and whole-program exercises (where the student code merely reads from the standard input and writes the computation result back into the standard output stream). CodeWorkout on the other hand supports more than just these type of questions. Indeed, its 'wrapper code' logic and the multi-prompt approach (to be described in the later parts of this thesis) enable CodeWorkout to offer a diverse range of coding questions. Despite its name, CloudCoder is not completely cloud based in the sense that it still requires participating institutions to install and maintain their own server instance. In fact, CloudCoder requires not one but two Linux servers to be set up for the institutions that use it, and the authors have admitted that self-hosting is one of the biggest impediments to its adoption. Studies of its usage by its authors indicate that CloudCoder has scalability and concurrency issues that will impair its usage in large traditional courses, let alone at the level of MOOCs.

Infandango [25] is a simple drill-and-practice system with a testing infrastructure that is

similar to that of CodeWorkout. In this system, the final Java programs for an exercise are submitted by students through the web front-end after authentication. Then Infandango's JUnit daemon compiles the code, executes it in a sandboxed environment and runs the test cases against it. Students are then given feedback based on which cases passed and failed. The authors point out that the modularity of the system in the form of its authentication, web front-end and database are its unique selling points. Of course, most of the autograding systems nowadays are extremely modular in nature. CodeWorkout is also modular in nature by virtue of being developed with Ruby on Rails and its concomitant MVC design. CodeWorkout is far more flexible in terms of its authentication as compared to Infandango because the latter only allows for the CoSign authentication system. Most educational institutions do not use this authentication system. It would not be feasible to use Infandango in MOOCs because they generally only accept their own authentication systems. It is also constrained by the fact that it works only for Java and not other programming languages. Infandango is symptomatic of how current autograding systems can fall into total disuse: there has been no update to its source code for the past three years. This is all the more telling because it has been in existence only for five years [25].

2.1.4 Path to extensive use

In their survey paper on automatic assessment (AA) systems, Ihantola et al. [26] have given some stinging reproaches to extant autograding systems, including drill-and-practice systems. They decry that most autograding systems have not been widely adopted because they were created solely for the sake of a thesis or a paper publication. They do not focus on building features that would make them a viable system for continuous use outside their parent institution. As the authors state: "there are far less systems that are widely adopted than there are papers about new tools" [26]. They point out some significant barriers to wide-scale adoption, beginning with systems that are constrained to work with only a single language like Java. Some systems are tightly coupled to their institution's LMS (Learning Management System) and so preclude use elsewhere. Ihantola et al. also argue that the flexibility and ease of specifying tests for the questions is of greater importance than previously perceived. Therefore, the autograding system must not limit instructors on what kind of programming exercises can be provided in introductory courses.

Thus, systems like CloudCoder and CodeWrite are their own barriers for adoption because of the limited kind of exercises that they can deal with. Another important aspect to widespread adoption is that an AA system must be truly open-source so that institutions can extend it if they deem necessary. Very few autograding systems that are in use outside their original institutions are open-source. Web-CAT has none of the above drawbacks, which explains much of its longevity. CodeWorkout aims to be the second such system but, as explained previously, it has other goals and features that will set it in a league of its own, particularly its operation at the scale of MOOCs.

Table 2.1: Comparison of CodeWorkout with other related systems

System	Gradual engagement	Open repository of exercises	Cloud-based courses	Student-created content	Flexible permission model
CodeWorkout	✓	✓	✓	✓	✓
CloudCoder	✗	✗	✗	✗	✗
CodingBat	Partially	✓	✗	✗	✗
Marmoset	✗	✗	✗	✗	✗
Web-CAT	✗	✗	✗	✗	✗
CodeWrite	✗	✗	✗	✓	✗
Infandango	✗	✗	✗	✗	✗

In her paper on the possible use of automated assessment systems in MOOCs [38], Vreda Pieterse re-affirms that designing test cases is quite difficult but is one of the key determinants to the success of AA systems in MOOCs. In general, we see that for traditional AA systems, setting up courses is problematic because they are either not free, require extensive work for deployment on an institution's server or suffer from significant usability issues. These problems, according to Pieterse, mean that these systems are not viable for courses like MOOCs because the difficulties exerted on students and instructors do not outweigh the benefits. Also, some of these systems do not have a proper sandboxing environment for executing student code. While this may be acceptable in traditional courses, Pieterse points out that, at the scale of MOOCs, this can lead to crippling consequences from malicious code. She goes on to give out specific recommendations for an AA system to be used successfully at the level of MOOCs. CodeWorkout is already in a position to satisfy most of her recommendations and is well on its way to satisfy the remainder. She concludes by saying AA systems have the potential to provide significant facilities for both MOOC students and instructors.

2.2 Summary of key differentiators

CodeWorkout offers a unique paradigm where anyone can declare themselves to be an instructor and register their course on it regardless of their institutional affiliation. In fact, instructors can create courses even if they are not part of any educational institution. It also provides an open model of enrollment for students through its different channels of authentication. Students are not limited to using the authentication system of their institution (if they have one) but can login through various social platforms like Google or Facebook. Table 2.1 gives the key differences between CodeWorkout and related systems.

If needed, CodeWorkout can also support the authentication system used by the particular institution. It is designed to support a gradual engagement model where students can prac-

tice problems even without logging in. In many other systems, especially on social media like Quora which allows for browsing even without authentication, this model has been shown to be effective in attracting and retaining users. This will serve well to attract non-institutional instructors and students.

As was seen previously, there are many key differences between CodeWorkout and its related systems. A lot of the autograders in the past decades were either not free or not open-source and thus were prohibitive to widespread adoption whereas CodeWorkout is both free and open-source. Some of the other systems that we have seen were not available to be used as a true cloud offering in the sense that they needed to be installed on dedicated servers within an institution. Naturally, a significant barrier that prevents institutions from using such autograding systems because of the expense of procuring and maintaining such an infrastructure. The overhead involved for instructors further detracts them from teaching. For higher-education institutions of limited means, like small colleges and high schools, this would be completely infeasible. CodeWorkout, on the other hand, is available completely online for various kinds of instructors and students to start using. Furthermore, CodeWorkout offers a completely open model of usage when compared to other systems:

There are other aspects to the flexibility of CodeWorkout. It is not limited to offering programming exercises in a particular language like Java or Python. It currently supports assignments in Java, Python, and Ruby, but its extremely modular grading framework enables it to support any programming language. Indeed, one of its salient features is the conversion of exercises across languages. A programming exercise written in Python that is conceptually possible in Ruby can be easily converted into its Ruby version. For example, a Python exercise involving the writing of a method to compute the factorial of a number can be easily converted into an exercise in Ruby, instead of having to recreate the entire question. Instructors can also define sharing policies for the programming exercises that they create; if an instructor makes an exercise sharable then it becomes available in the common repository of public exercises ('the gym') and so other instructors can easily incorporate it into their own courses.

Another aspect to CodeWorkout is that it provides a way for students to create their own programming questions and make it available to their peers. This increases the involvement of students in the course and showcases the higher-order skills that they have acquired in a programming language. CodeWrite is the only system that has remotely similar capabilities but is a restrictive system in the sense that there are only a limited type of exercises that students can create and a similarly limited sharing environment. As CodeWorkout says in its home page [41], it has been inspired by various educational systems and it aims to be the best-in-class system by incorporating the positive aspects of these systems while providing its own unique features.

Chapter 3

CodeWorkout Design Goals

As stated in the problem statement in Chapter 1, CodeWorkout aims to be more than just another of the plethora of autograding systems available. It is striving to be the most comprehensive and open drill-and-practice system for anyone who wants to learn the basics of a programming language. With that in mind, CodeWorkout’s design strategies have been consistently subordinated to this grand objective. We believe that this grand objective can be achieved through the completion of the following sub-goals:

- **Gradual engagement:** Anyone, regardless of institutional affiliation, should be able to start a course on CodeWorkout, set up their assignment and start having their students enroll in it. Similarly any student with just a browser should be able to access and practice programming exercises. Students must be provided access to significant features of CodeWorkout’s drill-and-practice system even if they have not logged in to the system.
- **A flexible yet secure permission model:** This goal is tightly linked to the previous one. CodeWorkout must be easily accessible to whoever wants to use it.
- **Enhanced drill-and-practice features:** CodeWorkout provides enhanced features over conventional drill-and-practice systems like feedback at the level of test cases and weighting of test cases in scoring of programming exercises.
- **Gym versus course separation:** Gradual engagement leads to the situation where the same exercise or workout can be practiced by a student in the Gym and then later as a part of his or her course. Only the scores for the latter should count towards a course grade. Ensuring this correctness has been one of the design challenges for CodeWorkout.
- **Exercise organization:** Exercises are the main focus of any drill-and-practice system and CodeWorkout aims to provide significant ‘helper’ features and facilities organized

around it. Exercises are available to be either directly used in courses or to be organized into workouts. The exercise model is truly ‘polymorphic’ in CodeWorkout: an exercise can be of different types like multiple-choice or coding; it can also be comprised of multiple parts, allowing for a richer variety of questions. CodeWorkout also provides ancillary capabilities like exercise families and exercise versions.

- **Collaborative and active learning:** CodeWorkout provides for instructors to share exercises with one another. Its framework facilitates students who have mastered a particular topic to be able to create their own programming questions that they can share for the practice of their peers. Finally, it also tries to provide students with a facility to share hints for exercises and to participate in a discussion forum associated with each question.

3.1 Target: Gradual engagement

CodeWorkout supports a gradual engagement design [1] for its clients. Gradual engagement is a way of attracting and enhancing the engagement of users with a stage-wise approach to systems, especially online systems. A crude but appropriate description of gradual engagement is that it aims to make all first-time visitors of a system to be its users. There are different kinds of systems on the web that follow different engagement strategies. These range from completely open systems like Google’s search engine to systems like YouTube that allows watching of episodes to be done without authentication but requires registration for upload of content. Since the Gym in CodeWorkout is meant to be a completely open public area where anyone can practice exercises of their choice, it must support users who have not logged in to the system. This is the most open and basic level of user engagement in CodeWorkout.

Next, users who have logged in have the additional capabilities like registering themselves as instructors, creating courses, or enrolling as students in available courses. Instructors get a chance to evaluate the system before adopting CodeWorkout in their course and can begin by first trying out a few practice exercises in the Gym area to gain an understanding of it and then try to create their own course. Then there is the more restrictive form of engagement where only a few specific students are allowed to enroll for a private course or only designated students can enroll themselves as a grader for a course. Such kind of gradual engagement would be beneficial in attracting both instructors and students.

CodeWorkout aims to provide an active practice area of public questions that anyone who wants to learn a programming language can access even without logging in. Once such users understand the usefulness of CodeWorkout, they can authenticate/register themselves on the system. This is where another of CodeWorkout’s facilities for gradual engagement comes in: its multi-system authentication feature. As seen on numerous successful Silicon Valley start-up websites, first-time users are more likely to authenticate themselves if they are given the

possibility of logging in using their existing credentials from popular platforms like Google or Facebook. This simple change, as compared to users registering themselves on the site's own authentication system, has proved to be successful in web sites like Quora [27]. For traditionally inclined brick-and-mortar institutions, CodeWorkout allows for authentication using CAS [46] which is used by a large number of universities.

We aim for our system to be adopted by people of various background intending to learn the basics of any programming language, regardless of institutional affiliation or lack thereof. The first basic step that we have taken in this regard is to make the system free and open-source. This minimizes the cost barriers involved with the adoption and gives institutions the opportunity to inspect the system beforehand. Of course, this also gives other CSE researchers the opportunity to extend the system with new features. There is no system installation, server setup overhead or maintenance involved for those do not wish to get entangled with it.

3.2 Target: A flexible yet secure permissions model

One of the main issues that CodeWorkout needs to take care of is authorization and permissions. At first glance, this might not seem to be as big an issue in educational systems as compared to enterprise systems. But the difference is the scale at which CodeWorkout is going to operate, hundreds of active courses. Since anyone can be an instructor or a student, there is no fixed role for anyone using CodeWorkout except for administrators. For example, a student who is in an introductory Java course might self-declare himself or herself to be an instructor of an ad hoc course to teach Python because of his or her previous expertise. That is to say that the same person at the same time can have roles both as an instructor and a student on CodeWorkout. The same situation is also possible with graders. For example, a graduate student who is a student in the CS 5314 Programming Languages course at Virginia Tech can also be a Teaching Assistant in an undergraduate course, where both courses use CodeWorkout. Very few AA systems have even a coarse-grained permission system and none deal with a role-based permissions requirement like CodeWorkout does. The data model of CodeWorkout is flexible enough to capture the role of each and every individual at the level of a course and not at the level of an institution.

The permission system of CodeWorkout is thus based on the enrollment of instructors, graders and students at the course level rather than any 'official role' at an organization level. This is one of the key features that enables it to be a viable product for MOOCs since it works for students who are enrolled in courses from both different MOOC participating institutions (like Stanford, University of Pennsylvania, etc.) and different MOOC systems like Coursera [33]. The easy registration also makes it simple for MOOC providers and other non-institutional educators to set up their courses and enroll their students. Since a different set of courses needs to be created on MOOCs every year, ease of course setup and student enrollment is important in MOOCs and CodeWorkout's easy-to-use course management

features go a long way in facilitating this.

The flexibility of such a permission system posed definite challenges in the design of CodeWorkout. How can the system remain flexible enough to accommodate the various roles that a single person might simultaneously take while offering the necessary authorization guarantees that are absolutely crucial in online systems that deal with education? For example, it must be ensured that the FERPA (Family Educational Rights and Privacy Act) protected information like the gradebook and student answer code must remain protected. Moreover, they should be accessible only to those who have a need for it. For example, it would be a gross violation if a person who is a grader in one course that uses CodeWorkout gains access to the gradebook of another course where he or she is just a student. Thus, course participants who use CodeWorkout need to be not only authenticated but authorized for their different actions based on their roles.

The modular design of CodeWorkout that makes it extensible hampers the specification of allowed and prohibited actions for different users. In terms of Aspect-oriented software design [29], authorization is a cross-cutting concern and is a single aspect logic that needs to be implemented for the various controllers that handle it. It becomes difficult to extend and maintain such a system where logic of permissions needs to be re-defined repeatedly for each controller, resulting in entanglement of code and inconsistencies of implementation. Fortunately, CodeWorkout has implemented a solution for this problem as will be explained in Chapter 4.

3.3 Target: Enhanced drill-and-practice features

From the paradigm-shifting arguments put forth by Geoff Colvin [6], we understand that mastery in a field or topic comes not with some mystical talent but through deliberate practice in tandem with coaching (i.e., feedback). Thus, the best way for students to gain mastery over the core concepts of a programming language is through solving a large number of small exercises that deal with those concepts. CodeWorkout supports this by providing students with multiple attempts for a single coding exercise. This gives them the opportunity of working on the code as much as possible. Thus, for a single question in CodeWorkout, most students can provide multiple submissions in order to get a perfect score. CodeWorkout stores the student code for every attempt along with the results of the test cases and any associated error messages. This allows the system to store snapshots of every student submission and opens a huge avenue to pose a lot of research questions on student behaviors, understandings and misconceptions.

There are many requirements for an AA system to claim itself as a drill-and-practice system: the first is that it needs to be available on a continuous basis for the students to practice. Online systems are perfect for this since they are available to students all the time, unlike lab sessions. Second, such systems need to provide multiple attempts for the same exercises

to accommodate students of varying learning curves within the same course. For example, a student who has a better grasp of the concepts needed to complete an exercise will be able to solve the question in fewer attempts as compared to a student who does not yet have a strong grasp of the required material. There is a deeper aspect to CodeWorkout—it allows for a student to work on an exercise until they have received a perfect score and thus are not limited to a single submission.

Third, a drill-and-practice system must have a large collection of these small programming exercises. Both the ‘large’ and the ‘small’ is important here. There needs to be a large number of exercises so to give students a substantial practice on the same concepts in different exercises. The small is important because each exercise imparting an improved grasp of a few core concepts is better suited for a drill-and-practice system as compared to larger exercise where the conceptual focus is diverted. Last but certainly not the least, the system must provide quick and appropriate feedback to students when they make a code submission (“an attempt”). This ensures an active learning environment and a chance for themselves to figure out what is wrong with their code, facilitating an introspection that is necessary in budding programmers.

One cannot talk about a large number of students in an drill-and-practice system without mentioning concurrency. When dealing with a large number of students, it is conceivable that a lot of them will submit their code for grading to the system at roughly the same time. The autograding system should be capable of grading these simultaneous submissions in a timely manner. Most drill-and-practice systems do not address this; it might either be because they have not been used substantially out of their parent institution or because they were never really meant to deal with a lot of students using their systems simultaneously. Hundreds of students from many institutions using a traditional drill-and-practice system at the same time could easily bring it to its knees. This is not a problem in CodeWorkout, because multithreading was one of the key tenets of its autograding framework design. The Puma web server, on which CodeWorkout is currently operating at Virginia Tech, has supported timed exams when over a hundred students have simultaneously submitted numerous code attempts and has performed reasonably.

While most systems give customized feedback only at the level of questions or assignments, CodeWorkout is the only drill-and-practice system that does scoring of weighted test cases and provide feedback at that level. This means that instructors can tailor-design the test cases of their exercises such that the system can give feedback that specific to failing test cases. For example, in an exercise that requires the computation of the factorial of a number, most students might not understand that zero factorial is one instead of zero or their code might not even work for such a case. The feedback given by CodeWorkout in this case, called negative feedback, informs them that zero factorial is one. Such tailor-made feedback can go a long way in disabusing students of their misconceptions. The onus is on the instructors to specify the negative feedback for a test case. However, it can clearly be seen that the time spent in designing such feedback is certainly worth it in terms of benefits at such scales.

Also, we remind here that the specification of the negative feedback is optional and hence instructors need only provide them when necessary. Test case level feedback synergises nicely with test case level weighting. Some test cases are more important than others. For example, when coming up with test cases, instructors can inherently determine which test cases are more difficult to pass than others. Furthermore, test case design for student code usually consist of a lot of trivial test cases. These repeatedly test the most obvious part of the question and only relatively few test cases deal with edge cases. This can be clearly seen in the case of factorial computation. Thus, test cases sometimes need to be weighted according to their importance in the exercise. This is optional and in exercises where it is not applicable, all test cases have the same default weight of 1.0. Thus when a student code is submitted, the score that the student receives is not simply a reflection of the ratio of test cases passed but rather the weighted average of the test cases. This simple but elegant change provides for better feedback and test specification, one of the main prerequisites for the use of AA systems at MOOC scale [38].

3.4 Target: Course versus Gym separation

Another design challenge that arose out of our gradual engagement strategy was how to separate the practice of the same exercise at different levels of engagement and use. For example, consider an authenticated student who practices an exercise or workout on the open Gym at one point in the Fall semester and is only partially able to solve it. Then the same exercise or workout could be part of a course that he or she takes up in the Spring semester. In this case, the system should not take the previous attempts of the student in the Gym into consideration for the course. This is because only the work that a student has done during the duration of a course should count towards his or her course grade.

There is another scenario to be taken into account wherein instructors can share workouts between courses and students can repeat courses leading to the situation where the same exercise or workout can be attempted by the *same student in two different courses*. The same principle applies here: only the work that the student has put in for the current course should count towards the course grade. In effect, the system should not take previous attempts on that exercise or workout into consideration.

3.5 Target: Exercise organization

CodeWorkout also provides for the concept of exercise families: exercises that are basically minor variations of one of another. For example, a question involving the determination of the median of an integer array is basically the same as that of an Integer ArrayList. It would make no sense for additional exercises from the same exercise family to be incorporated into the same workout. Exercise families also form a manageable layer of abstraction over

exercises in the sense that when instructors search for questions, they can simply search for it by exercise families. When dealing with a large repository of exercises, it is easy to incorporate exercises that are the same and such mechanisms help instructors from repeating questions. Exercise family is a notion that has not been tackled in any other drill-and-practice system.

CodeWorkout also provides for a feature called exercise version. An exercise may be substantially modified by an instructor after students in a class have already attempted it, like when an instructor changes to suit his or her next course. If the students simply continue to use the previous version, then those students who have already worked on that question are at a disadvantage because their solution might no longer be valid and may have already lost a number of attempts on it. On the other hand, it is also problematic for the teacher to create a whole new exercise because it is an unnecessary overhead for them. The simple solution that CodeWorkout provides is to create a new version of the exercise. When students try to access the same question again, they will get the current version of the exercise as opposed to the previous version and they will get to start with fresh attempts, i.e., the scoring is done at the level of an exercise version rather than an exercise. Aside from offering multiple versions on exercises, CodeWorkout offers multiple types of questions, mainly coding exercises and multiple-choice exercises. There are also multi-prompt/multi-part exercises that allow multiple concepts/topics to be test in the same question. More will be elaborated on these in Chapter 4.

3.6 Target: Collaborative and active learning

Per Bloom's taxonomy, there are seven levels of students demonstration of skills that they have acquired out of a long learning experience such as a course. The lowest level of skill is just remembering the facts peddled out in class. The highest level skills are called high-order learning skills and involves students coming up with their own content based on their learning. In order for any learning experience to be truly effective, it must give students an opportunity to demonstrate higher-order thinking Skills. While drill-and-practice systems like CodingBat can help test the 'Apply' part of Bloom's taxonomy they are not sophisticated enough to do more. CodeWorkout tries to address this by providing students the facility to create their own content: students who have displayed mastery of a particular topic are given the opportunity to create programming exercises on their own for the benefit of their peers. Student contributed content can also manifest itself in the form of hints, where students who have obtained a perfect score on an exercise can write hints for that exercise that can made available to the other students in the class.

Another way of promoting both collaborative and active learning within CodeWorkout is through anonymous discussion forums. Just like comments are available to be posted at the bottom of most blog articles, we believe there is great value to be had from making a per-exercise discussion thread available on CodeWorkout. Instead of lost emails and repeated

broadcasts, it would be beneficial if the discussions about an exercise are directly attached to it instead of being posted elsewhere. This is especially relevant in drill-and-practice systems where there are numerous questions assigned to students and they can easily miss the information given for a question. In this way, students can alert an instructor if there is something wrong with an exercise or clarify any ambiguities in the question that can be shared with the rest of the class. The viability of this feature in CodeWorkout has been discussed in [31] and it emphasizes that discussion forums works only if students have the option of posting anonymously.

3.7 Additional considerations

An instructor can use a form interface to create a workout by adding the relevant exercises to it using a drop-down list. The same form can be used to add points to the individual exercises, set deadlines and add these exercises to a course offering. The flexibility in the creation and sharing of exercises and workouts is one of CodeWorkout's unique selling points. For instructors who do not wish to share their workouts publicly but rather share it confidentially with a few trusted instructors, they need only specify the emails of the other instructors with whom they want to share it. We are currently working on implementing a model of sharing of workouts based on the different license given for them.

The simplicity of its interface to students also helps. Regardless of the type of question, CodeWorkout gives the same kind of experience to the students when they are practicing. That is, CodeWorkout uses the same views to generate the practice page for all types of questions which vary only a little depending on whether to display choices or a coding area. CodeWorkout's responsive design [32] also helps in making the overall system more usable on mobile devices for its different aspects. CodeWorkout's responsive design makes its interface render well on mobile devices, enabling use from such devices. The availability of educational content on mobile devices has been shown to be beneficial to students and has made Smart Learning Content (SLCs) [4] available to students even on the move. Students would be able to check their grades on CodeWorkout, answer MCQs and participate in discussion forums using their tablets. Such kind of ubiquitous availability of CodeWorkout on mobile devices will make it more popular, especially amongst teenagers. Some web-site providers tackle this with a brute-force approach by offering only apps for a mobile experience. Such a solution is not acceptable in educational systems because it necessitates students to install an additional mobile app which might be deemed invasive and has been shown to pose significant barriers to adoption[19].

An aspect of CodeWorkout that is beneficial to instructors is that it allows them the option of specifying policies for workouts. The rationale for this feature is that different kinds of scenarios call for different behaviors from the system regarding workouts. An obvious scenario is the inherently vast differences between assignments and exams. How and when feedback is to be displayed is different in tests as compared to homework assignments. Also,

since CodeWorkout allows for student-written hints in exercises and exercises are portable between courses, different instructors can decide whether hints are allowed or not in their workouts. As mentioned earlier, CodeWorkout also allows for a forum to exist at the level of workouts and exercises. While this is certainly a useful feature, some instructors might wish to disable this for specific workouts. They can do all this and more using workout policies. These help constrain the display and practice of the exercises in a workout for students according to policies set forth by instructors. A policy is specified through a set of boolean fields that specify what is allowed and what is not allowed in a workout. A workout policy can be attached separately for *each workout offering*, i.e., different course offerings using the same workout can specify different policies for it.

We have also put in mechanisms to implement ideas from Item Response Theory (IRT) [30] to refine the student-contributed exercises and hints. When an exercise has been practiced by hundreds of students, we can find the difficulty and discrimination associated with the question to automatically assess its quality as broadly depicted in Figure 3.1. Questions that are too difficult or are too easy can be quickly found out from the IRT data gathered for these exercises. Similarly, the discrimination enforced by these questions can also be automatically gauged by CodeWorkout. These two together can automatically help determine how good a student-contributed exercise is. This information can be leveraged by the system to automatically weed out bad student-written exercises and help come up with a repository of high quality exercises. In a similar vein, hints can also be tracked to see how well they help students in fixing their missing test cases and good hints can thus be determined.

As was previously mentioned, CodeWorkout's design is modular in nature and this modularity is apparent at all levels. Because it was developed using Ruby on Rails, CodeWorkout follows the MVC design pattern and shows a clear separation of code concerns. The flexibility of Rails' Object-Relational Mapper (ORM), ActiveRecord, makes working with the databases completely transparent. The only database specific information in the entire system is present in the database configuration files. Thus, CodeWorkout is agnostic to the underlying database and can shift to a different database almost at will. Similarly the front end design is decoupled from the computation logic or the storage system, which allows the design of flexible views for CodeWorkout's clients (instructors, graders and students). This modularity makes it portable when CodeWorkout needs to be installed on the servers of different institutions.

- The discrimination model systematically gives precedence to exercises that differentiate between students with adequate mastery and those without (high discrimination)
- Hint helpfulness model automatically ranks hints based on correctness of responses after it is received
- Most helpful hints given precedence

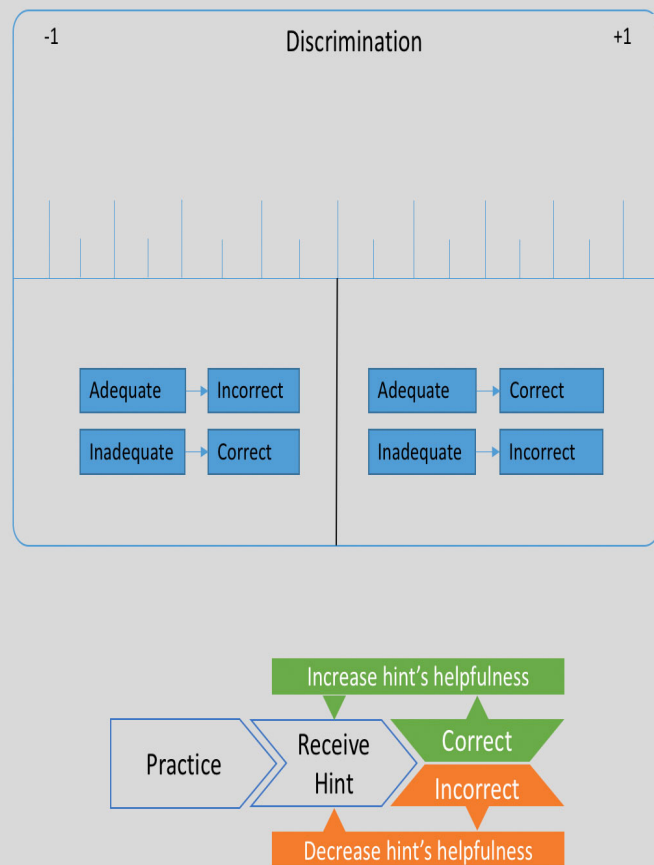


Figure 3.1: Support for IRT statistics

Chapter 4

Implementation

In this section, we give some of the details about CodeWorkout’s implementation. We start with a description of its architecture, then move on to detail how its design goals are implemented. Finally, we describe how other useful features of CodeWorkout were implemented.

4.1 Architecture

At the highest level, CodeWorkout is based on the traditional three-tier architecture as shown in Figure 4.1. Since it is a completely hosted solution, the clients of the system (instructors, graders and students) only need a web browser to interact with the system. All the code submissions of the student are sent as a HTTP request to the application server. The same goes for any form submissions made by the instructor to create a course or a workout. The application server contains a CodeWorkout instance developed using Ruby on Rails [21]. The application server takes care of authentication, authorization, object creation and HTTP response generation. The student submitted code is also processed in this layer and is evaluated in a sandboxed environment. There is also a separate MySQL server that acts as the database for the entire system. The data model of the CodeWorkout database is rich enough to support the various model objects involved in it and will be discussed shortly. The CodeWorkout Rails application is hosted on Puma, a Ruby server that is specialized for concurrency [37]. The whole system is run on top of the high-performance NGNIX web server [42].

CodeWorkout is a system that is intended to be available to hundreds of institutions and for online courses that act at the scale of MOOCs. Hence, unlike traditional AA systems, concurrency becomes a real issue for its routine operation. With an active user base in the thousands, it would be a common scenario to see hundreds of students accessing the system at the same time. Autograding at these scales becomes a difficult matter and CodeWorkout uses multithreading to deal with its concurrency issues. As shown in Figure 4.1, the grading

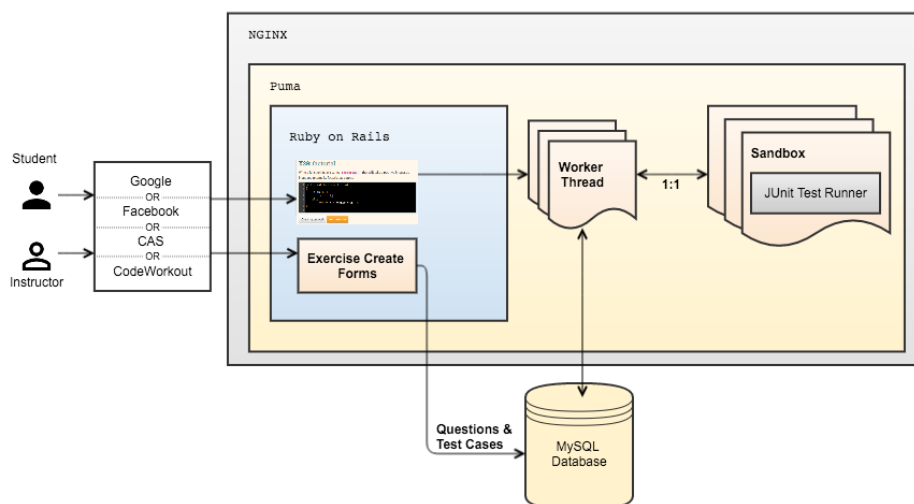


Figure 4.1: CodeWorkout's Architecture

of each student code is done parallelly and independently. When a student code is submitted, the Exercise Controller in CodeWorkout creates a new attempt for it and passes on the code to the 'perform' method of the autograding module, CodeWorker. CodeWorker is the part of CodeWorkout that does multithreaded grading using SuckerPunch [23], an asynchronous processing library. SuckerPunch is now a part of ActiveJob, the background job framework for Ruby on Rails. The default number of grading threads, called workers, needs to be specified beforehand and these worker threads are started when the CodeWorkout application is started on the server. These worker threads are present as a pool of threads available to simultaneously grade submissions. Each worker thread individually takes about two seconds on average to grade a typical programming exercise.

When a student solution is submitted to the server and the autograding code is called for it, a job call is put in the job queue. One of the many worker threads running in the system dequeues the grading job from this queue, renders the student code into its final form, calls the unit testing code on it and stores the result in the database. Thus, when there are a lot of submissions simultaneously, instead of student code getting 'lost in grading', CodeWorkout adds these submissions into a queue where they are all guaranteed to be taken up by one of the worker threads. Since each code is graded in its own sandboxed environment by a separate worker thread and the results for it are stored separately, there is no possibility for one student's code to interfere with another student's code even if they were submitted around the same time. The stored student code is evaluated independently in its own sandboxed environment, mitigating the harm that could arise from potentially malicious programs.

We now return to another of Pieterse's recommendations [38] in order for AA systems to be incorporated into MOOCs: to provide a secure environment where student code can be executed. The system must be robust enough to withstand student code that can, adver-

tently or inadvertently, bring it to a halt upon execution. CodeWorkout incorporates the secure sandboxing environment of Web-CAT to safeguard against this. Every time a student submits his or her code, a new attempt is recorded in the database with its own unique identifying number. The grading thread spawned from CodeWorker helps create a sandboxing environment to safely execute the code for that attempt. This sandboxing environment consists of a unique sub-directory in CodeWorkout's file system, distinguished from that of other attempts using the ID of the attempt.

An Ant script (a modified version of the one that Web-CAT uses) is used to build this environment, run the tests on the student code, collect the results and performs clean-up. The main grading thread spawned from CodeWorker runs the ant script for grading code written in a particular language, for example Java. The build script tries to compile the completed student code in its attempt directory. If the compilation is unsuccessful, it sends back an appropriate error message back to student as feedback. If the compilation succeeds, then the JVM is set up again and the JUnit test runner is called the student code. The JVM is run under the aegis of Java's Security Manager [34] for safely executing the Java code. Of course, the Security Manager only works as per the policy specified by the application. For Java, CodeWorkout provides this separate policy file that specifies the allowed actions by the student code. This policy is what helps enforce the sandbox since it restricts the student code to run commands only within its attempt directory. The policy file contains a lot of 'grant' statements: statements that allow the code to perform certain action and access certain files. Anything that is not granted is disallowed.

Novice programmers frequently write loops that lead to time-out errors because of infinite looping. Since CodeWorkout deals with a larger number of students, this is a problem that needs to be tackled. The time-out strategy of JUnit for test cases is not enough on its own because it is usually either too pessimistic or optimistic. The JUnit invocation that that our Ant script makes uses the adaptive timeout capabilities that Edwards et al. implemented for Web-CAT's test cases [17]. It dynamically adjusts the timeout limit for each test case in the test suite. This reduces the overall grading time for student code where at least one test case gives an exception due to timeout. Features like these in the sandboxing environment go a long way in providing secure platform for a large-scale drill-and-practice system.

As explained previously, CodeWorkout can be used completely from the web or can be installed on internal servers by institutions that would like to keep their data sequestered. This allows for widespread adoption of CodeWorkout. Since its source code is open-source [16], it provides for more participation with the rest of the CSE research community and for institutions to add their own desired features. CodeWorkout also has a dedicated wiki site [13], inside the Web-CAT community, where user groups can post questions and share information.

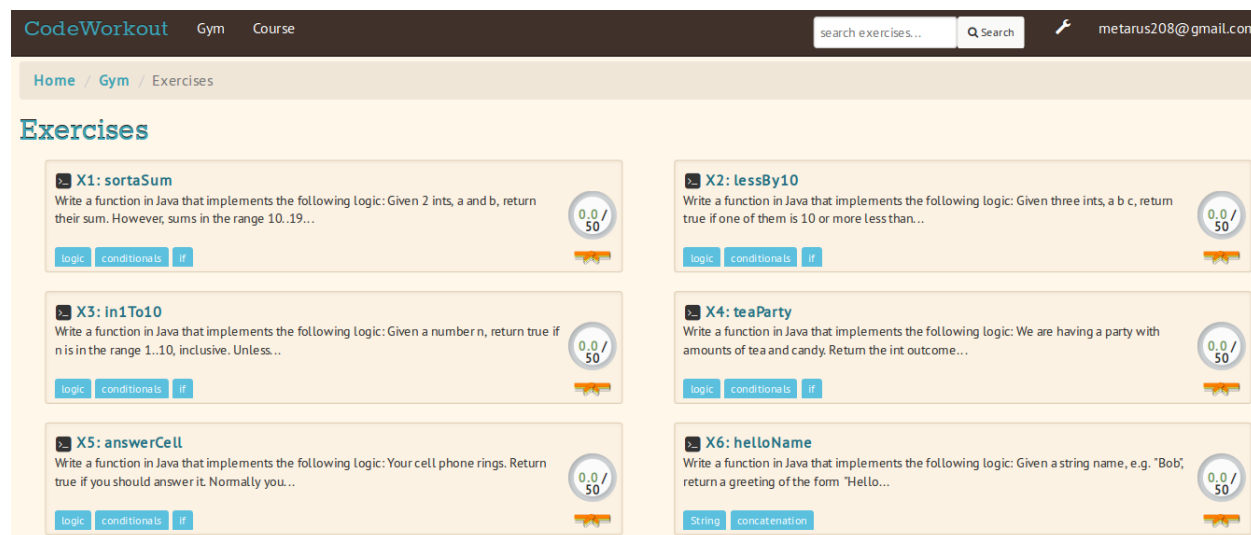


Figure 4.2: CodeWorkout’s Gym—An area of open practice exercises¹

4.2 Facilitating Gradual Engagement

CodeWorkout provides for a gradual engagement strategy where users practice exercises on the system even without logging in to the system. The Gym, as shown in Figure 4.2, provides a repository of open workouts whose exercises can be freely practiced regardless of whether a student is in a course that uses CodeWorkout. For example, students who just want to practice Ruby programming, but are not part of any ‘official’ course that offers it on CodeWorkout, can still access the Gym to work on such problems. The Gym also provides for a facility students can search for topics and languages of interest to them for practice as shown in Figure 4.3.

Students can also choose to authenticate themselves into the system through their existing accounts on popular systems, like Google or Facebook. In case their institution use CAS authentication, they can use it to authenticate themselves on CodeWorkout. Instructors also have the option to make their courses public to allow self-enrollment. This means that students who are logged in to the system but are not part of the course can still enroll themselves in it. These are called open courses and students get the benefit of accessing high-quality exercises that instructors have created. Figure 4.5 shows a student self-enrolling into an open course. This feature goes a long way in engaging students who have just drifted into the system.

Teachers can register themselves on CodeWorkout as instructors regardless of whether they are in an actual education system. Similarly, they can get students enrolled in their course. These are the features that make CodeWorkout extremely viable for use in MOOCs. On the other hand, institutions that do not want to use the centrally hosted CodeWorkout instance at Virginia Tech can setup their own instance on their servers with just a few commands.

Search Results

Searching for: conditionals

Individual Exercises

The screenshot displays a search results page for the keyword 'conditionals'. Under the heading 'Individual Exercises', there are six exercise cards arranged in a 3x2 grid. Each card has a title, a description, a progress indicator (0.0/50), and a set of tags (logic, conditionals, if). The exercises are: X1: sortaSum, X2: lessBy10, X3: in1To10, X4: teaParty, X5: answerCell, and X7: endsLy.

Exercise ID	Exercise Name	Description	Progress	Tags
X1	sortaSum	Write a function in Java that implements the following logic: Given 2 ints, a and b, return their sum. However, sums in the range 10..19...	0.0 / 50	logic, conditionals, if
X2	lessBy10	Write a function in Java that implements the following logic: Given three ints, a b c, return true if one of them is 10 or more less than...	0.0 / 50	logic, conditionals, if
X3	in1To10	Write a function in Java that implements the following logic: Given a number n, return true if n is in the range 1..10, inclusive. Unless...	0.0 / 50	logic, conditionals, if
X4	teaParty	Write a function in Java that implements the following logic: We are having a party with amounts of tea and candy. Return the int outcome...	0.0 / 50	logic, conditionals, if
X5	answerCell	Write a function in Java that implements the following logic: Your cell phone rings. Return true if you should answer it. Normally you...	0.0 / 50	logic, conditionals, if
X7	endsLy	Write a function in Java that implements the following logic: Given a string, return true if it ends in "ly".	0.0 / 50	String, conditionals, if

Figure 4.3: Student search for exercises on conditionals

For the traditionally inclined brick-and-mortar institutions, CodeWorkout can interface with Learning Management Systems (LMS) that use OAuth2-based (RFC 6749) authentication [18]. This in effect means that students on CodeWorkout can use the same Single Sign-on (SSO) authentication services that a lot of institutions already use. The LTI 2.0 standards are also making a push for SSO services and hence soon in the future CodeWorkout can interface with the authentication systems of most learning management systems.

CodeWorkout already provides for authentication via CAS [46] (Central Authentication Service) that Virginia Tech currently uses. Figure 4.4 shows an example of a user authenticating himself into CodeWorkout using his account on VT CAS. CAS is one of the most widely used authorization services that is used in universities worldwide after it originated from Yale. More and more universities have started adopting it and this is expected to continue. Also, it would be a simple matter to extend CodeWorkout to similar SSO services used at other institutions. We believe that such facilities will bring in a large number of students into the systems, either as a part of MOOC scale courses or students from various institutions that will use CodeWorkout or just curious self-motivated student drifting on the web.

OAuth support for CodeWorkout comes through the OmniAuth gem [2] along with Devise [39] in Rails. Through OmniAuth, CodeWorkout is able to support the login of users through multiple authentication pathways. Thus, users have alternative options of logging into the system through their Google or Facebook accounts as shown in Figure 4.6. CodeWorkout only requires that a user provide an email to be logged in to the system and thus when users log into the system using these external accounts, it takes their email address from their accounts. The login links on CodeWorkout for these external accounts are provided by the OmniAuth callback controller that uses the server's DNS to contact the appropriate authenticator and receive confirmation.



Figure 4.4: User authentication using CAS

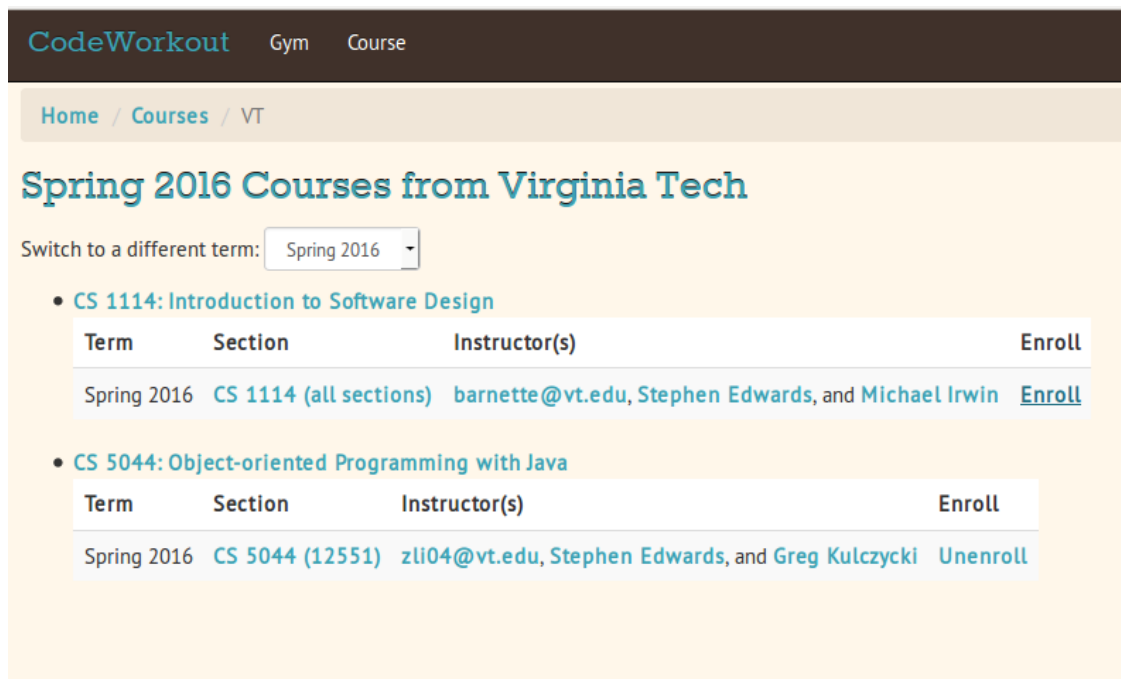


Figure 4.5: Student enrolling in an open course

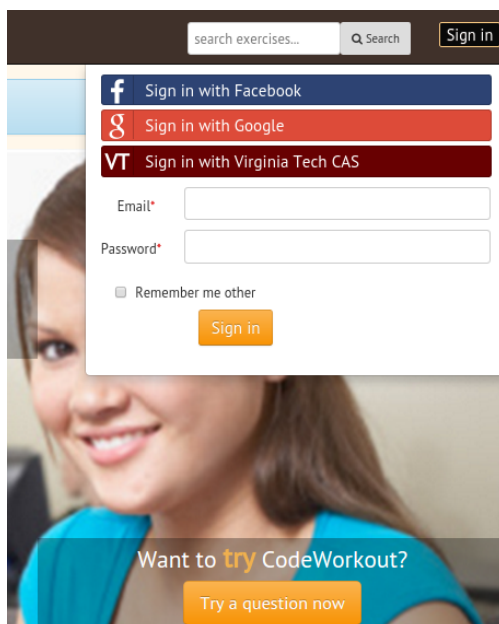


Figure 4.6: Authentication options

While the gradual engagement strategy has been implemented and the code for this is in master branch of CodeWorkout’s source code, this feature has not yet been evaluated in a real course. The reason for this is that CodeWorkout has been undergoing extensive changes even while it was being used in different courses at Virginia Tech. The gradual engagement non-authenticated users was implement only in Spring 2016 and so was not made available on our production server.

4.3 Facilitating a Flexible Permission Model

CodeWorkout’s permissive authentication model is one of the primary facilitators of its gradual engagement strategy and enables it to be used in a variety of settings. Students and instructors are not limited by the global role they have on the system but instead can ‘shapeshift’ their roles according to the course at hand. This is possible because there are fundamentally two categories of roles in CodeWorkout. The first is the traditional global role; here one is either a system administrator or simply a regular user. Each user has a global role associated with him or her. Roles like graders and students are not considered as a global role but are part of Course Role model. These roles can be separately specified for each user on a course level. This role information for a user in a course offering is captured through the course enrollments table; in effect it captures the ternary relationship between the regular user, course offering and the possible roles as shown in Figure 4.7. Also, users are not inherently affiliated with organizations, i.e., the concept of organizations is only

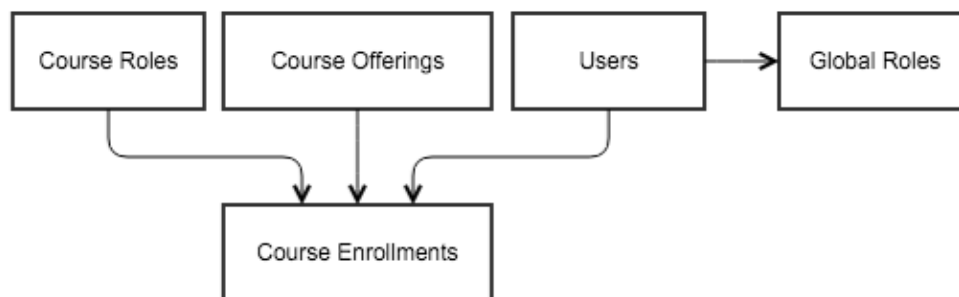


Figure 4.7: Course Enrollments

applicable for courses and not users. Thus, in CodeWorkout, it is a course that can be affiliated with an organization and not the other way around. Hence, organizations are no barrier to an instructor offering a course.

Of course, this seemingly completely open permissions gives the impression that it could lead to a lot of potential security breaches. CanCan is the authorization library for Ruby on Rails that CodeWorkout uses to prevent this from happening. Authorization (as a subset of security) is a cross-cutting concern that affects almost all the major controllers in our Rails application. While CanCan is not, *per se*, an aspect-oriented gem, the problem that it solves and the way that it does it almost makes it one. It mainly works through an ability file: this is the centralized file containing all the allowed user actions within CodeWorkout. The controllers are configured with this gem such that any user action that is not specified in the ability file cannot be executed. This might sound a bit restrictive at first glance but has proven useful in designing a well guarded system. This restriction is applied by calling the `load_and_authorize` method on the controller so that no action for the controller can be performed without statement in the ability model file. In Aspect-oriented Software Design terminology, the `load_and_authorize` makes a ‘catch all’ pointcut making all controller method calls a join point. The permission checking is made into a ‘before’ advice by looking up the ability file to see whether an action is allowed. The ability file generally consists of `can` statements: statements that specify that actions are valid for an individual user. Almost all such `can` statements are conditionally executed. For example, a `can` statement can be specified such that it only allows users who are system administrators to view the list of all exercises in the system or only instructors for a course can access its gradebook.

This flexible centralized permission model has been available for CodeWorkout since its first use at Virginia Tech in Fall 2015. It has performed well in permitting users only those actions for which they are authorized.

4.4 Facilitating Enhanced Drill-and-Practice Features

The main functionality that CodeWorkout provides is drill-and-practice of small introductory-level programming exercises, even though its framework is flexible enough to support other kinds of exercises like full class implementation. CodeWorkout is designed to be an online system and this takes care of the most fundamental requirements of a drill-and-practice system because they need to be always available to a student. However, what ultimately makes or breaks a drill-and-practice system is its facility for multiple attempts and feedback. Hence, the cornerstone of our design is around implementing these. Next, the system also allows for multiple attempts: each student can potentially attempt an exercise an unlimited amount of times and each attempt is recorded separately in the database. Instructors can also choose to set a stricter policy for the exercises in their course workouts by limiting the number of attempts that a student gets over a period of time. This helps ensure that students review their own code before submitting and not just rely on the system to find out its flaws as they have been observed to do [3].

CodeWorkout does not penalize students for the number of attempts that students need for an exercise and this encourages students to work on problems till they get it right. Since each attempt is stored and evaluated separately, it allows for both students and instructors to review their progress on a particular exercise over a period of time. Also, CodeWorkout provides the facility for students to review their past work as shown in Figure 4.8. For example, just before the final cumulative test in a course students have the facility of looking at their submissions for earlier assignments as well as the feedback that they got for them. This helps students tremendously in rectifying their own mistakes over the duration of a course, a feature noted for its absence in most other drill-and-practice systems.

CodeWorkout has a reasonably quick feedback system where once the student code is submitted, it is quickly evaluated on the server and the results with feedback is sent to the student. The feedback ranges from compiler errors to missed test cases. CodeWorkout is particularly helpful because it gives feedback at the level of test cases. Thus students can use this granular feedback to self-improve their code. An example of feedback to student code is shown in Figure 4.9.

4.5 Facilitating Gym versus Course Separation

CodeWorkout provides for the separation of exercises and workouts in the Gym versus those in courses through two extensions in its data model. These are the workout offerings table and the course offerings table. These two together, in conjunction with the attempts table and workout scores table, has provided a clean solution to the problem of entanglement of grades of the same workout being practiced by the same user in different contexts. The workout offering captures the relationship between a workout and an offering of a course,

Home / Courses / VT / CS 1114 / Test 2 Practice / X167

X167: countHi

Given a string, compute recursively (no loops) the number of times lowercase "hi" appears in the string.

```

1 public int countHi(String str)
2 {
3     return 1;
4 }

```

Next exercise

Feedback

0.23 / 1.0

Behavior	Result
countHi("xxhixx") -> 1	✓
countHi("xhixhix") Expected:<2> but was:<1>	✗
countHi("hi") -> 1	✓
countHi("hihih") Expected:<2> but was:<1>	✗
countHi("h") Expected:<0> but was:<1>	✗
countHi("") Expected:<0> but was:<1>	✗

Figure 4.8: Student review of past exercises

Home / Gym

X1: Factorial 1

Write a function in Java called `factorial()` that will take a positive integer as input and returns its factorial as output.

```

1 public int factorial(int n)
2 {
3     if (n == 0)
4         return 0;
5     if (n == 1)
6         return 1;
7     return n * factorial(n - 1);
8 }
9

```

Check my answer!

Feedback

Behavior	Result
factorial(0) Remember that zero factorial is one	✗
factorial(1) -> 1	✓
factorial(2) -> 2	✓
factorial(3) -> 6	✓
hidden	✓
hidden	✓

Figure 4.9: Autograder Feedback

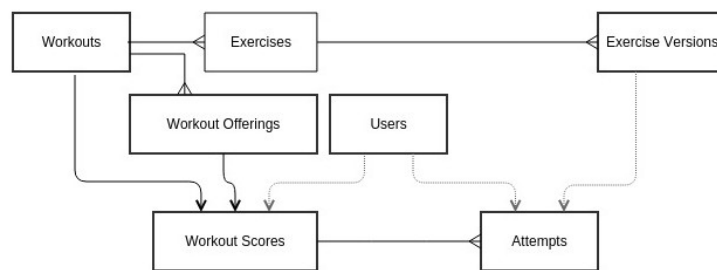


Figure 4.10: CodeWorkout data model support for scoring

i.e., it provides a many-to-many mapping between workouts and the course offerings that use them. This table also captures the data that exists only with this association. For example, the opening date for that workout in a course, the submission deadlines associated with that assignment for that course or policies to be enforced for that workout in a course. Similarly, course offerings can include stand-alone exercises, exercises which have not been bundled into a workout. The relationship between stand-alone exercises and courses is captured through the course exercises table. This table provides a many-to-many mapping between exercises and the course offerings that use them. This table also captures the data that is associated only with this association. Thus the workout offering model and the course exercises model offer richer usage of workouts and exercises in courses compared to a static model where they cannot be used outside their parent course.

In order to keep separate the scores for the same workout in different courses, the workout offering information is used in the **workout scores** table. This table keeps tracks the ternary relationship between users, workouts and the particular workout offering in consideration. This table keeps track of the information associated with this relationship like the user's score in the workout and how many exercises in it have been completed so far. Thus, this table helps provide a clear separation of scores for the same workout-user pair across different offerings. The relationships are depicted in Figure 4.10. A **workout score** where the workout offering is empty indicates that the workout was practiced outside of any course, that is, from the Gym. Thus CodeWorkout's data model allows provides for a clean separation of workouts in the Gym and workouts in courses.

The same applies for **attempts** and **workout scores**; a user's score in a workout is the sum of all the final scores secured by the user in its component exercises. The final score in an exercise for a user is the score in the last attempt by that user on the exercise. The score for an user's code submission for a particular exercise version is stored in the **attempts** table. Suppose a workout has three exercises, then a user's score in that workout is the sum of the final scores that he or she has obtained for all three exercises. Each entry in the **attempts** table can also be associated with a workout. An entry in the **attempts** table without an associated workout or offering indicates that the exercise was practiced independently from the Gym. Thus CodeWorkout's data model allows provides for a clean separation of exercises in the Gym and in courses.

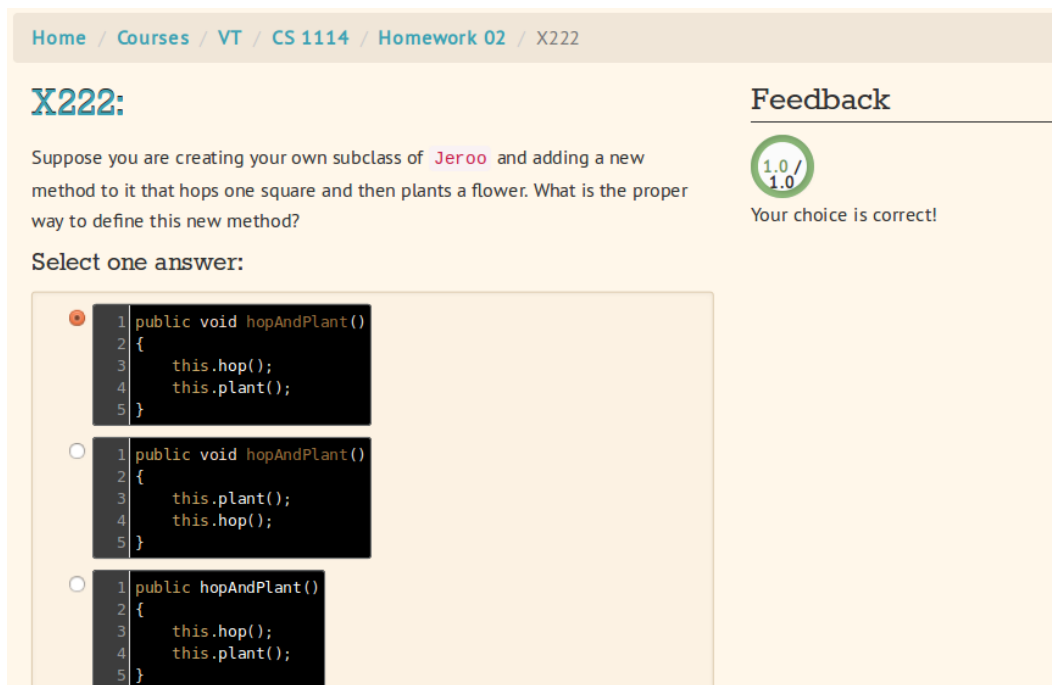


Figure 4.11: A CodeWorkout MCQ exercise

4.6 Facilitating Exercise Organization

As mentioned, CodeWorkout’s exercises show ‘polymorphic’ behavior: exercises can be of different types like MCQs or coding questions. Instead of creating separate classes for each type of exercise, CodeWorkout’s data model uses an inheritance-like approach to this. This has resulted in cleaner organization of exercises since objects that aggregate exercises (like workouts and exercise offerings) need not concern themselves with the *type* of exercises that they are aggregating. For example, since workouts are just a collection of generic exercises, a workout can consist of a mix of MCQs and coding questions. The logic and rules for each type of exercise is specified in its own class but it inherits the common properties like creator from the parent. An example of a CodeWorkout multiple-choice question is shown in Figure 4.11.

Exercises can also be either multi-prompt (multi-part) or single prompt exercises. This allows for creating richer types of exercise, instead of just plain-old single-method questions like CodingBat or CloudCoder. Multi-part questions, for example, can help in testing how well students have understood concepts like functional decomposition. The first prompt can be used for the top-level method while the second prompt can be used for the helper method. An example of this kind of CodeWorkout multi-part question is shown in Figure 4.12.

Exercise families and exercise versions are useful features that help instructors organize their course content. Exercise families help ensure that no two exercises from the same family get



The screenshot shows a web interface for a programming exercise. At the top, a breadcrumb trail reads: Home / Courses / VT / CS 214 / Ventress / X181. Below this, the title 'X181: kicsortaSum' is displayed in a teal font. The main text explains the task: 'Use the functional decomposition principles you've learnt in Chapter 3 for this rounding question.' It provides a detailed rule for rounding: 'In this case, round an int value up to the next multiple of 10 if its rightmost digit is 5 or more, so 15 rounds up to 20. Alternately, round down to the previous multiple of 10 if its rightmost digit is less than 5, so 12 rounds down to 10.' The task is then defined: 'Given 3 ints, a b c, write a roundSum() method return the sum of their rounded values.'

Below the text, there are two code editors. The first editor contains the following code:

```
1 public int roundSum(int a, int b, int c)
2 {
3
4 }
5
```

The second editor contains the following code:

```
1 public int round10(int num)
2 {
3
4 }
5
```

Between the two code editors, there is a text instruction: 'To avoid code repetition, write a separate helper public int round10(int num) { and call it 3 times. Write the helper entirely below and at the same indent level as roundSum().'

At the bottom of the interface, there are two buttons: 'Check my answer!' (highlighted in orange) and 'Next exercise'.

Figure 4.12: A CodeWorkout multi-part question

added to the same workout because that would be redundant. When a workout is about to be created from a set of exercises, CodeWorkout can check whether the two exercises are from the same workout and if so can raise an error during the exercise creation. The exercise families table is used to keep track of the exercises that are from the same family and so provides another way of searching for exercises in the repository.

Similarly when there are various versions to an exercise, the information that is specific to each version is stored separately in the exercise version table whereas the generic exercise information is stored in the exercise table. Each exercise in the latter table keeps track of its current version by simply updating the current exercise version number when a new version is created. The key design decision facilitating exercise version is that they are associated with attempts and not exercises, i.e., students attempts are recorded with respect to the exercise version and not the exercises directly. This gives a significant amount of latitude when creating different versions for the same exercise so that scores for an exercise will accurately reflect the latest version.

4.7 Facilitating Collaborative and Active Learning

Instead of just providing passive learning through student practice of exercises, CodeWorkout also aims to provide an environment for active learning. This is done through the creation of student-created content and discussion forums. The former manifests itself on CodeWorkout mainly through the creation of programming exercises authored by students. Consider the case when a student, through assiduous practice on CodeWorkout, has gained perfect scores for all exercises on the topic of conditionals. Then the instructor has the option of allowing the student to create their own question on the topic of conditionals. After the instructor curates this question, it can be shared with the rest of the class for practice. This achieves a wonderful synergy effect. First, for the students have authored an accepted exercise they gain the higher-order skills of 'Creation' per Bloom's taxonomy. As already explained [10], students who put in the effort for the Higher Order Skills of being able to create their own programming exercises have been shown empirically to perform better in introductory programming courses. Next, it promotes collaborative learning: students getting together to help each other accomplish a learning objective. This helps foster a basic adaptive learning environment where students who are lagging in the course have the extra exercises to practice on and improve their programming skills.

Then there are hints. Similar to student-authored programming exercises, students who have shown their mastery of a topic can come up with hints that they can give to the rest of the class. After this hint has been created and curated, it can be affixed to the exercise for other students to avail it if needed. Just like programming exercises, a good hint is difficult to come up with. If it's too obscure it will not help students and if it is too directly based on the solution, the instructor will disallow it. As such, student-created hints are also another way that CodeWorkout facilitates active learning, collaborative learning and showcasing of higher-order skills.

It is also important for drill-and-practice systems to have a large number of small exercises. It is difficult and time-consuming for instructors to come up with such a large number of questions on their own. Even re-using questions from question banks like the Canterbury question bank [20] is not enough because instructors still have to spend time in data-entry for these questions in traditional AA systems. CodeWorkout helps out in two ways for this. First, the sharing of workouts between instructors greatly reduces the time spent in re-inventing the wheel. Workouts that have been made public by an instructor will be available in the Gym and another instructor can easily add it into his or her course.

For private workouts, instructors can choose to confidentially share it with other instructors as they deem fit. The data model of CodeWorkout, through its workout owners table, records who are the allowed proprietors of a workout and allows for a private workout to be include in a course only if the instructor has been allowed to do so. The other way that CodeWorkout facilitates the creation of a large number of programming exercises for a course is enabling students to create their own questions. This is especially useful in large courses

where student contribution can significantly expand the pool of practice exercises.

As mentioned earlier, discussion forums are available on per exercise and this can be used as a place where students can voice their concerns. Since CodeWorkout aims to be a completely online solution, the discussion forum attached to every exercise ensure that an exercise's question specification page is a one-stop place for student to get information about solving it. Students can ask general doubts about the exercise as well as hear instructor clarifications regarding it or hear their fellow students' views on it. All these features make for a more collaborative and active learning environment.

The instructors have an additional level of control over what kind of additional content appears on an exercise's question page. When a student create his or her own hint or exercise or posts in the discussion forum, it is visible only to the instructor and not to the rest of the class. When the instructor goes to the exercise page, the instructor can approve the new student-contributed content by simply approving it, thereby making it visible to the rest of the class. Else they can choose to decline it. Just like student-contributed content, discussion forums are also an optional part of courses on CodeWorkout. It is up to an instructor to decided whether such features should be allowed in their workouts.

4.8 Towards a more complete system

An aspect to CodeWorkout that we believe will significantly aid its usability is the strategy of responsive design of its user interface. Responsive design [32] is a way of designing the front-end of a system so that its interfaces adapt to the device at hand. For example, it is possible to set 'breakpoints' in the user interface so that it can gracefully meld into different screen sizes. The difference that Bootstrap provides is the way in which our interface is written to set up breakpoints. This has helped provide students a more elegant and usable front-end experience to students not only on their laptops but on their tablets and smart phones. Responsive design is also beneficial to the ever-busy instructor who gets to monitor the progress of his or her students by simple opening CodeWorkout inside his or her mobile browser.

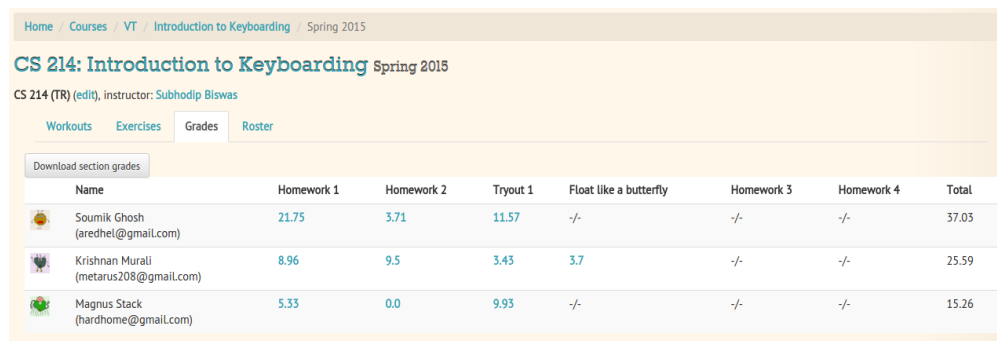
CodeWorkout has taken some definitive steps to make the system easier to use with help from the Bootstrap framework [44] to implement its responsive design. The thumbnail display of exercises, as shown in Figure 4.13, in a workout also gives a good sneak peek into them and is particularly useful for students who trying to finish a workout. CodeWorkout provides an IDE-like interface through CodeMirror [22] for students to practice the programming exercises. CodeWorkout's coding interface provides useful features like indentation and syntax highlighting that goes a long way towards the scaffolding of novice programmers.

CodeWorkout also provides useful auxiliary features to instructors like the course gradebook as shown in Figure 4.14. Instructors can also review the work of a student in the exercises of a course workout as shown in Figure 4.15 or even drill-down to see the code that a student

The screenshot displays the CodeWorkout application interface. At the top, there is a dark header with the 'CodeWorkout' logo and navigation links for 'Gym' and 'Course'. Below the header, a 'Test 2 Practice' assignment is shown with a lock icon, a '10' badge, and a date of '2016-04-06'. The assignment description states it contains practice coding questions for Test 2 involving loops over lists or strings. A progress indicator shows 12% completion. Below this, five exercise thumbnails are listed, each with a title, description, tags, and a progress gauge.

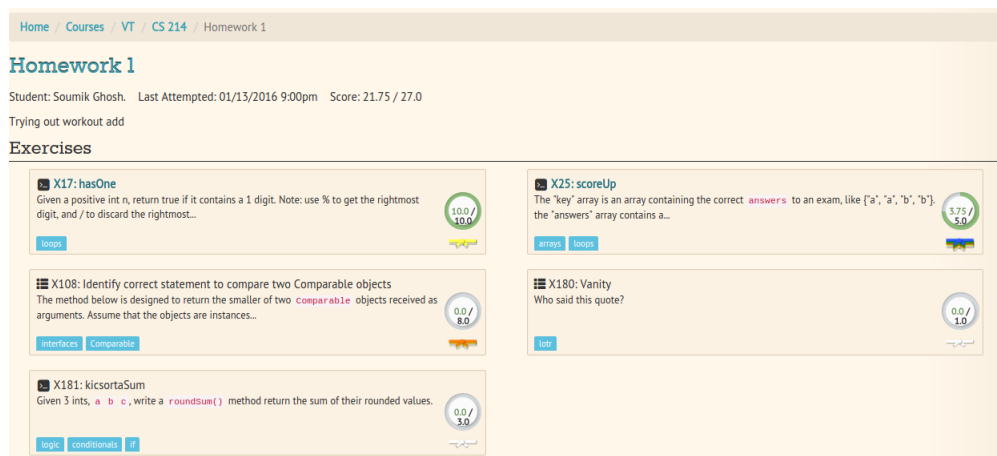
Exercise ID	Exercise Name	Description	Tags	Progress
X237	listUpper	Write a method called <code>listUpper()</code> that takes in a list of strings, and returns a list of the same length containing the same strings...	loops, logic, conditional, List, String	0.0 / 1.0
X167	countHi	Given a string, compute recursively (no loops) the number of times lowercase "hi" appears in the string.	recursion, logic, conditionals, String	0.23 / 1.0
X29	factorial	Write a function in Java called <code>factorial()</code> that will take a positive integer as input and returns its factorial as output.	factorial, multiplication, math, numeric, re...	1.0 / 1.0
X240	doubleChar	Given a string, return a string where for every char in the original, there are two chars.	String, loops	0.0 / 1.0

Figure 4.13: Workout and exercise thumbnails



Name	Homework 1	Homework 2	Tryout 1	Float like a butterfly	Homework 3	Homework 4	Total
Soumik Ghosh (aredhel@gmail.com)	21.75	3.71	11.57	-/-	-/-	-/-	37.03
Krishnan Murali (metarus208@gmail.com)	8.96	9.5	3.43	3.7	-/-	-/-	25.59
Magnus Stack (hardhome@gmail.com)	5.33	0.0	9.93	-/-	-/-	-/-	15.26

Figure 4.14: Instructor accessing course gradebook



Homework 1

Student: Soumik Ghosh. Last Attempted: 01/13/2016 9:00pm Score: 21.75 / 27.0

Trying out workout add

Exercises

- X17: hasOne**
Given a positive int n, return true if it contains a 1 digit. Note: use % to get the rightmost digit, and / to discard the rightmost... 10.0 / 10.0
- X18: Identify correct statement to compare two Comparable objects**
The method below is designed to return the smaller of two `comparable` objects received as arguments. Assume that the objects are instances... 0.0 / 8.0
- X181: kicSortaSum**
Given 3 ints, a b c, write a `roundSum()` method return the sum of their rounded values. 0.0 / 3.0
- X25: scoreUp**
The "key" array is an array containing the correct `answers` to an exam, like ["a", "a", "b", "b", "b"]. the "answers" array contains a... 3.75 / 5.0
- X180: Vanity**
Who said this quote? 0.0 / 1.0

Figure 4.15: Instructor reviewing student performance on a workout

had written for a particular exercise.

CodeWorkout's workout policy model is used to record the workout specific rules that an instructor has for a particular workout. For example, workouts which are homeworks ought to be treated differently from ones which are exams. In CodeWorkout terms, the latter are simply workouts that have much shorter duration of availability. There are other significant differences between them in how the feedback ought to be rendered for students. Exams involving coding questions are thought to emulate paper-based tests and so should not provide extensive feedback to students on their test case results for their submitted code. That is, while the students in an exam sees that the system has graded the code they should not be given any further feedback beyond code compilation errors, if any. There are also various shades of gray between these two extreme policies. Instructors might decide that students will not receive feedback from the system until the workout is completely closed for practice, i.e. the last possible deadline for it has passed. Instructors might also restrict the access of students to the review of their own code after the workout is completely closed for practice. All this can be specified while adding the workout to a course offering.

For handling students who need more time for certain assignments and exams on CodeWorkout, the student extension model object is used. Appropriate extended deadlines can be given for different students by the instructors after determining the necessity for it on a case-by-case basis. This student extension overrides any other deadline for that student in a particular workout and hence it will be available for an increased amount of time even though that workout is nominally ‘closed’. Such kind of flexibility is what makes CodeWorkout such a great addition to be used in introductory programming courses.

Chapter 5

Evaluation and Analysis

At the end of the day, CodeWorkout is meant to be a real-world system and one of the best ways to test out a real-world system is on the field. CodeWorkout was first deployed for two courses, CS 1114 and CS 5044 (both using Java), in Fall 2015 at Virginia Tech. Informal feedback was collected from the students and bugs were fixed. Two of the main features that was added after our initial release was the instructor review of past student exercises and the reloading of a student’s past attempt in an exercise. The former means that CodeWorkout now allows instructors to drill down to the level of an exercise submission to see the code students submitted for each and every exercise, along with the feedback received. The latter means that if a student opens an exercise, his or her last attempt is automatically loaded to facilitate continuity.

Table 5.1: CodeWorkout usage in CS@VT

Course	Term	No. of students	No. of workouts	No. of exercises
CS 1114	Spring 2016	378	8	48
CS 5044	Spring 2016	38	3	25
CS 1114	Fall 2015	372	6	35
CS 5044	Fall 2015	24	12	196

CodeWorkout was re-deployed in Spring 2016 for the next offering of the above two courses at Virginia Tech as shown in Table 5.1. The analysis of student performance was done for CS 1114. Consent forms were distributed to the students of this course and 200 of the 378 students enrolled consented for their data to be used for research purposes. Each workout in course had multiple exercises, and these were predominantly programming exercises but Homework 2 consisted solely of simple MCQs. Two mandatory homework workouts followed it. During graded homework workouts, students had unlimited attempts and unlimited time to practice, and were shown the maximum amount of feedback on each exercise—that is, they saw the results of all software tests applied to their answers, and for nearly all software

tests, they also saw the full details of test values and expected results. Only a small number of software tests did not expose the details of what was being tested.

After these, a mid-term test (Test 1) was delivered in March on CodeWorkout with two programming exercises along with MCQs through Canvas. A small sample test was administered on CodeWorkout before that. Then a practice workout was released in the beginning of April to help the students prepare for the second test. After this, the second test (Test 2) on CodeWorkout was deployed to students. Table 5.2 presents the statistics of CodeWorkout’s usage in this course. Note that the abnormally high average time taken for Test 1 is because of the many extensions granted to students for it, which has skewed that metric. Overall, CodeWorkout has evaluated 177637 student code submissions in the seven months of its deployment. Of this, 56084 attempts were made by consenting students from this course. A part of the two mid-term exams for the CS 1114 course was conducted using Canvas MCQs and the coding questions were conducted on CodeWorkout.

Workout Name	Exercises	Students attempted	Avg. no. of exercise attempts	Avg. time taken per exercise (sec)	Avg. normalized workout score
<i>Homework assignments</i>					
Homework 2	10	197	1.4	120.72	1.0
Homework 5c	7	176	4.7	4634.98	0.986
Homework 6	5	195	9.9	12616.84	0.96
Homework 7	10	195	9.0	13643.43	0.933
<i>Exams</i>					
Test 1	2	198	7.7	29243.34	0.848
Test 2	2	190	11.3	913.07	0.637
<i>Practice</i>					
Test 2 Practice	10	155	7.0	14584.98	0.649
Homework 5d	2	175	7.8	541.38	0.846

Table 5.2: CodeWorkout’s use in Virginia Tech’s Spring 2016 CS 1114 course for different workouts

5.1 Analysis of attempts

Herein we present the results of the analysis that we conducted on the performance of students on the exercise attempts and how it is correlates with their final scores on exercises.

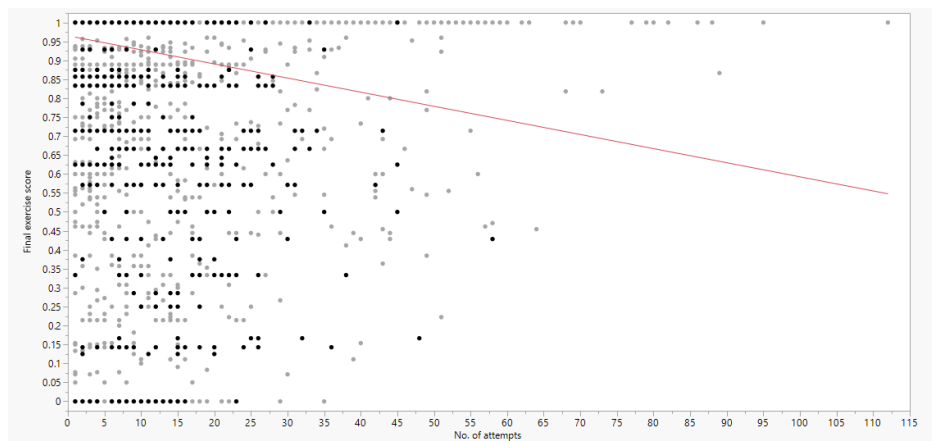


Figure 5.1: Number of attempts on an exercise versus final score

5.1.1 Number of attempts vs. Final score

We studied the relationship between the number of exercise attempts by the student on an exercise with their final score on it. The Analysis of Variance (ANOVA) was performed on this and the regression is shown Figure 5.1. Even though the p-value of the analysis is less than 0.0001, the distribution shown in the graph clearly indicates that there is no non-trivial relationship between these two measures. This is understandable because these are mostly drill-and-practice programming assignments and thus most students will keep working at an exercise till they get it right. This is indicated by the horizontal line for the maximum score at the top of the graph. The statistics from Figure 5.2 further backs this up.

5.1.2 Number of completed exercises vs. Exam score

We studied the relationship between the overall number of exercise attempts by the student with their exam scores on CodeWorkout. We also studied the relationship between the number of perfectly completed exercises by the student and their exam scores on CodeWorkout. An exercise is considered to be 'perfectly completed' if the user has scored the maximum score possible for that exercise in his or her last attempt. For every student, we considered the attempts and completion status for all the exercises in CodeWorkout in CS 1114, regardless of whether they were assigned, practice or exam exercises. This subsection also examines which of the two is more important. By exam score, we mean the average of the two exam scores for a student, separately for the CodeWorkout coding questions and Canvas MCQs.

Students were under different constraints during tests as compared to homework workouts. Students had hard time limits, and were expected to complete their code writing exercises along with all of the non-coding questions that were also on the exam. In addition, CodeWorkout did *not* give full feedback to students. Instead, exercises were answered

Summary of Fit					
RSquare		0.155258			
RSquare Adj		0.142195			
Root Mean Square Error		0.205372			
Mean of Response		0.743684			
Observations (or Sum Wgts)		198			
Analysis of Variance					
Source	DF	Sum of Squares	Mean Square	F Ratio	
Model	3	1.5038837	0.501295	11.8853	
Error	194	8.1824760	0.042178	Prob > F	
C. Total	197	9.6863597		<.0001*	
Lack Of Fit					
Source	DF	Sum of Squares	Mean Square	F Ratio	
Lack Of Fit	102	5.8232562	0.057091	2.2263	
Pure Error	92	2.3592198	0.025644	Prob > F	
Total Error	194	8.1824760		<.0001*	
				Max RSq	
				0.7564	
Parameter Estimates					
Term		Estimate	Std Error	t Ratio	Prob> t
Intercept		0.3921743	0.130569	3.00	0.0030*
No. of attempted exercises		-0.008633	0.005091	-1.70	0.0915
No. of fully correct exercises		0.0190442	0.004116	4.63	<.0001*
(No. of attempted exercises-41.9545)*(No. of fully correct exercises-36.4596)		0.0004684	0.000231	2.02	0.0443*
Effect Tests					
Source	Nparm	DF	Sum of Squares	F Ratio	Prob > F
No. of attempted exercises	1	1	0.12130080	2.8759	0.0915
No. of fully correct exercises	1	1	0.90285784	21.4060	<.0001*
No. of attempted exercises*No. of fully correct exercises	1	1	0.17282492	4.0975	0.0443*

Figure 5.2: Statistics for the analysis of final exercise score against number of overall attempts and number of completed exercises

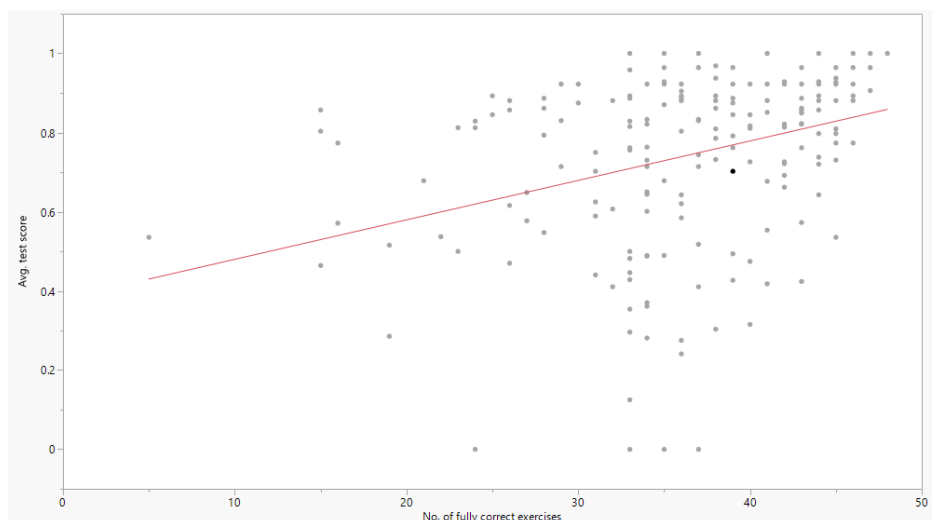


Figure 5.3: Regression between the CodeWorkout Test 2 score and the number of perfectly completed exercises

under a policy that showed compilation errors, and results on three provided examples that were part of the question prompt only. Although exercises included a much more extensive set of tests to assess the correctness of student answers, they could not see the results for these tests, or the numeric scores for individual exercises or the entire workout, during the exam. Since student work is automatically saved on CodeWorkout each time they check their work, students were free to work on other parts of the exam and come back to review their work, complete with the most recent results on the limited set of examples, whenever necessary until the exam ended.

In this analysis, we first consider how the average CodeWorkout exam score of a student was affected by the number of fully completed exercises by that user in *all* previous exercises. This is further illustrated in Figure 5.3. The p-value for the relation between the number of perfectly completed exercises and the average exam is less than 0.0001 and the F-ratio for the same 25.2225. This is further reinforced by the low F ratio of 0.9425 in the Lack of Fit analysis. This indicates that there is a strong positive relation between the two, i.e., the more thorough the coding practice of a student, the better his or her exam performance. However, the spread of the score in Figure 5.3 show that these two measures are more correlated rather than one causing the other. There are possibly other factors influencing this relation, like the course specifics and possible differences amongst students themselves.

5.2 Analysis of a practice workout

In this section we consider the effect of the practice workout that was released on CodeWorkout. This was an optional workout that was made available beforehand to the students

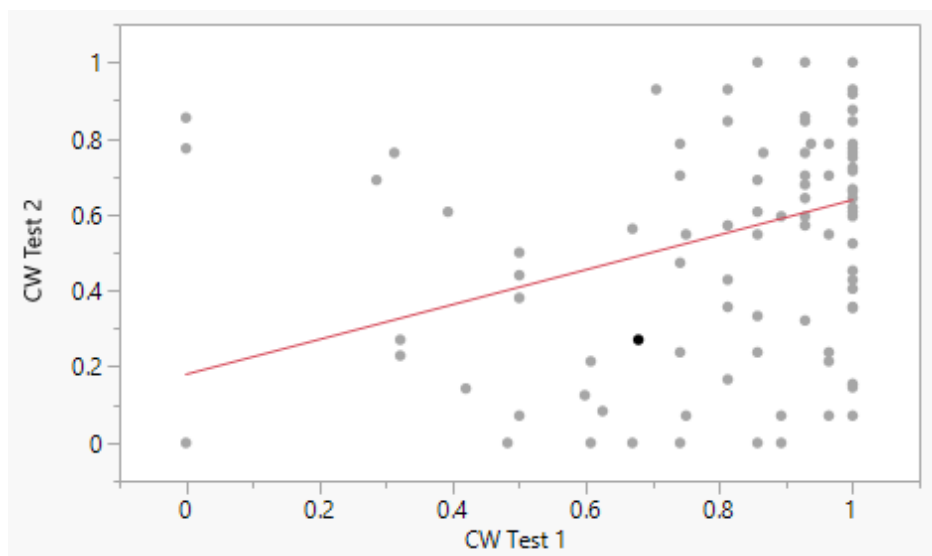


Figure 5.4: Regression between the scores on CodeWorkout exercises in the two tests for Practice Group 1

to help them prepare for this test. This workout had ten related programming exercises. Test 2 had two programming exercises, the same number as for Test 1 on CodeWorkout. Test 2 was a 'much harder' exam compared to Test 1, as evidenced by the low average score for it (64.12) compared to Test 1 (86.67).

We examine the idea that increased practice through working on optional assignments positively benefits students performance on exams, regardless of the student abilities. While practice in theory should have a measurable impact, its effect should be disentangled from the intuition that high performing students generally practice more. To measure the effect of practice alone, we focused this analysis on the Test 2 Practice workout comprising of ten programming exercises. Since it was not for grade, only 81.6% attempted any exercise in that workout. Just 35.3% attempted all the exercises in this practice workout.

To study the benefits of practice, we used Test 1 scores as a proxy measure for the current skill level of a student. Since both tests are summative in nature and cover different skills, both exams already play this role in the course's plan for assessing student learning. However, Test 1 covered different content from Test 2 and so we could not directly use it as a pre-test or consider differences between the two test scores as a learning gain measure. Nonetheless, by using Test 1 as an ability proxy, we could employ it as an independent variable in testing our hypotheses about its impact on Test 2 scores.

We also wanted to characterize participation in the optional practice workout that was given in between the two exams. Absolute scores on mandatory exercises have questionable value as predictors of outcomes since students were able to continue working on

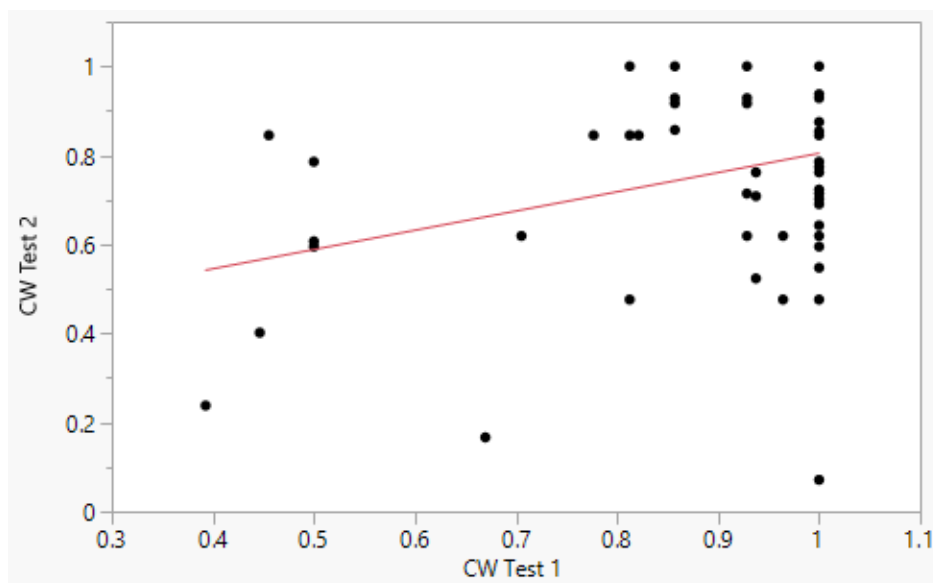


Figure 5.5: Regression between the scores on CodeWorkout exercises in the two tests for Practice Group 2

exercises until they mastered them. Since we are interested in the effects of voluntary practice, we divided the students into two groups: *practice group 1* (PG1) did not attempt every exercise in Test 2 Practice (including students who did no practice at all), and *practice group 2* (PG2) attempted every exercise in Test 2 Practice at least once.

5.2.1 Relationship between practice workout performance and exam performance

Figures 5.4 and 5.5 shows the relationship between scores in CodeWorkout Test 1 and in CodeWorkout Test 2 for both groups. It can be clearly seen that students from PG2, the ones who have practiced more on CodeWorkout, performed better on Test 2 as compared to their classmates from PG1. Since both tests included code writing questions, they require skills similar to those in the practice exercises involving MCQs and coding exercises, we can consider the two types of questions separately. On code writing questions answered on CodeWorkout, PG1 students earned a average score of 76.5%, compared to 56.7% for students in PG1 (S.D. = 28.7%). On MCQs and coding questions, PG2 students earned 63.6%, compared to PG1 students at 57.9% (S.D. = 20.5%).

Considering the practice group and Test 1 scores as independent variables, as well as their cross interaction, an analysis of variance indicates a significant effect on the coding exercise scores in Test 2 ($df = 189, F = 49.1, p < 0.0001$). Separate effect tests for the two variables show that both Test 1 scores ($F = 43.0, p < 0.0001$) and practice group ($F = 15.9, p$

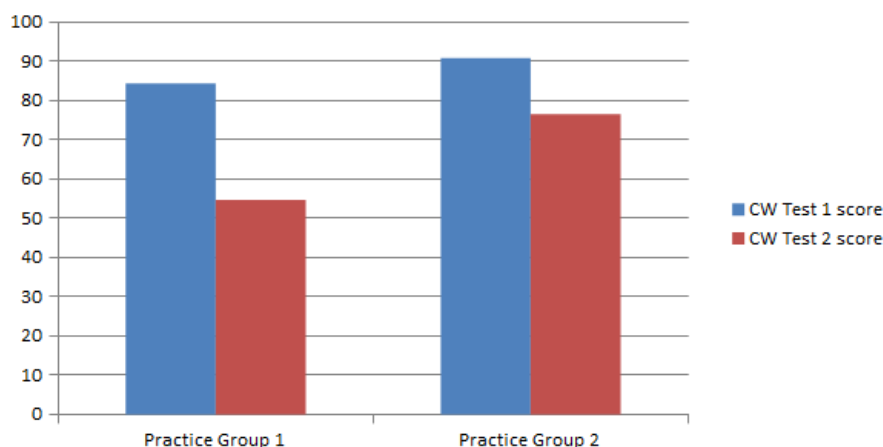


Figure 5.6: Attempts on an exercises and perfectly completed exercises versus final score

< 0.0001) are significant, but there is no significant interplay between them ($F = 0.60$, $p = 0.44$).

Partial η^2 values were computed for both the Test 1 effect and the practice group effect to determine effect sizes on coding exercise scores. The effect size for Test 1 scores is $\eta^2 = 0.264$, indicating that 26.4% of the variance in Test 2 code writing question performance stems from the student's earlier Test 1 score, which is a large effect size. However, the effect size for the practice group independent of Test 1 score is $\eta^2 = 0.143$, indicating that approximately 14.3% of the variance in Test 2 coding exercise performance stems from whether or not the student chooses to practice all exercises in the ungraded workout, which is a large effect size. Figure 5.6 looks at the practice workout from a different perspective. It compares the mean test score in both tests for both groups. While there is a marginal difference (roughly 5%) between average score on Test 1 between the two groups, there is more than a twenty percent difference between the average Test 2 score between the two groups. *Cohen's d* calculated from the test averages for PG1 vs. PG2 on code writing exercises in Test 2 is 0.69.

Putting it differently, students from Practice Group 2 on average scored *forty percent* more than their Group 1 counterparts on Test 2 after their extra practice on CodeWorkout. On the other hand, Figure 5.7 shows the performance of these two groups on the mostly knowledge-based multiple-choice questions in Test 1 and Test 2. Again, there is a marginal difference between the two groups on Test 1 and there is an equally marginal difference between the performance of these two groups on Test 2. This indicates while the extra practice that CodeWorkout gave to students from Practice Group 2 helped them score better on coding questions, it did not significantly help them in multiple-choice questions. This behavior is as expected because Code-

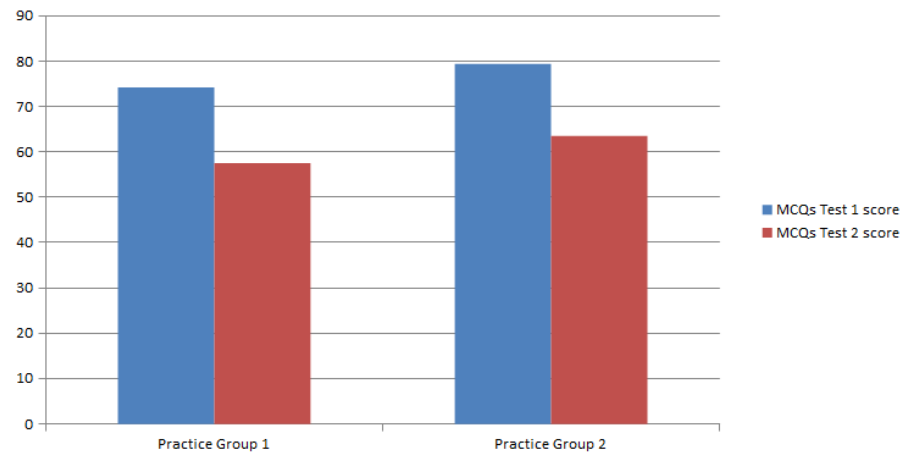


Figure 5.7: Performance of the two practice groups on the multiple-choice questions in the two tests

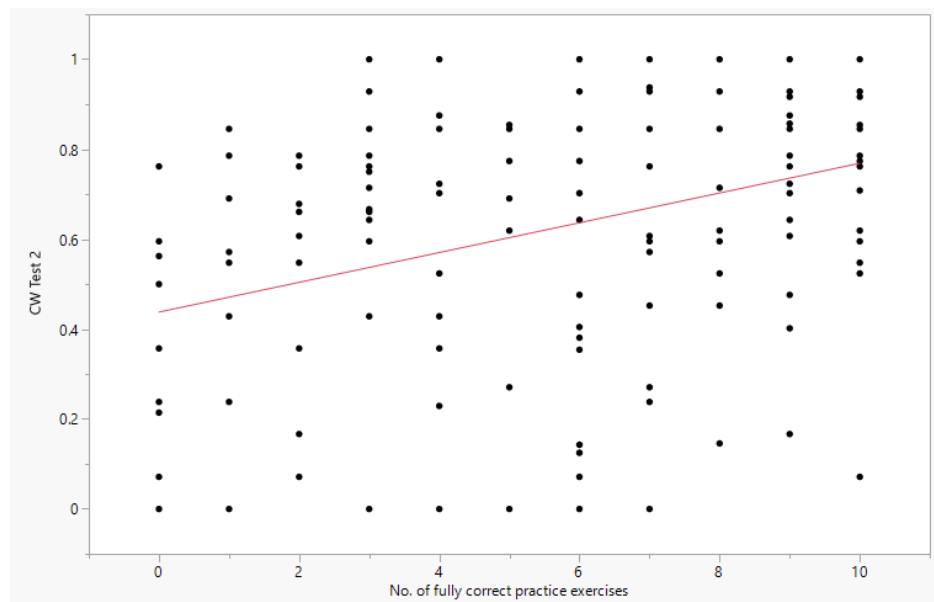


Figure 5.8: Relation between the number of perfectly completed practice exercises and Test 2 score

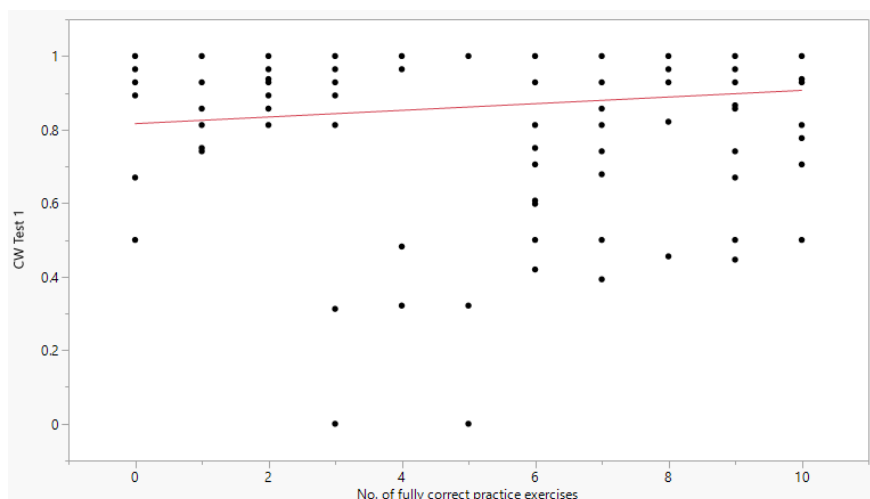


Figure 5.9: Relation between the number of perfectly completed practice exercises and Test 1 score

Workout is more for programming practice than helping with knowledge-based MCQs.

In Figure 5.8, we turn to the lens of the number of fully completed exercises and how it relates to the performance of students in Test 2's coding questions. The p-value after ANOVA is < 0.0001 and the F-ratio is 25.5486; both these, along with low F-ratio of 0.6791 in the Lack of Fit analysis, show that there is a very strong positive relation between a student's number of perfect practice exercises and his or her score on the coding questions in Test 2. Figure 5.8 on the other hand shows the relationship between the performance of students on the Test 1 coding questions and the number of practice exercises that they perfectly completed. As can be seen from the figure, there is no significant relation between the two, i.e., those who perfectly completed more practice exercises were not necessarily the ones who scored more on the preceding exam. This is backed up by the ANOVA analysis on the data which has a relatively high p-value of 0.0847 and a low F-ratio of 3.0118. This indicates that students scored better on the coding questions of Test 2 because of the extra practice that they obtained through CodeWorkout. All this is further proof that more practice on CodeWorkout leads to better coding skills.

5.3 Analysis of MCQs vs. Coding exercises

Herein we present the results of the analysis that we conducted on students performance two exam workouts. We analyzed how their performance on MCQs relates to their performance on coding questions in the same test.

From Figure 5.10, it can be understood that there is a strong positive relation between student performance on MCQs and on coding exercises in tests. This is further backed

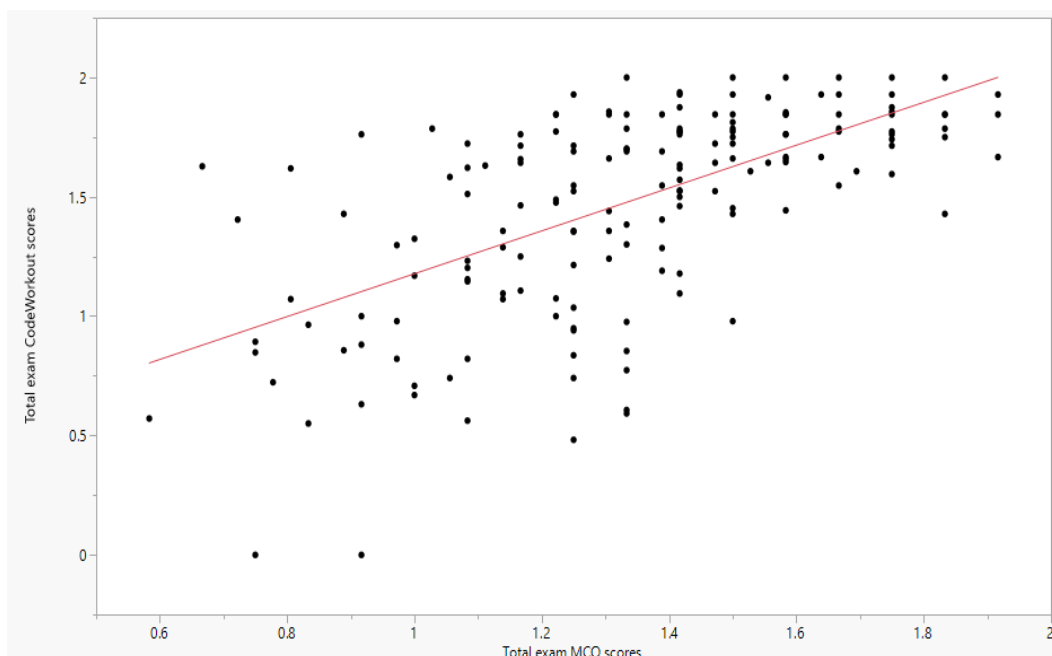


Figure 5.10: Regression between the scores on CodeWorkout exercises and Canvas MCQs in the two tests

by the ANOVA which gives a p-value of < 0.0001 and a high F-ratio of 122.9544. This is reinforced by the F-ratio of 1.0288 in the Lack of Fit analysis. This indicates that well-designed multiple-choice questions, along with programming exercises, can also be used to gauge the students in a programming course, making the case for MCQs to offered in CodeWorkout's exercises along with coding questions. Thus the combination of coding exercises along with multiple-choice questions make for a more complete drill-and-practice system in introductory programming.

We can also consider the effect on the MCQ portion of Test 2, which does not directly involve coding skills. Considering both practice group and Test 1 scores as independent variables, as well as their cross interaction, an ANOVA indicates a significant effect on the MCQ scores in Test 2 ($df = 189$, $F = 36.1$, $p < 0.0001$). Separate effect tests for the two variables indicate that Test 1 scores ($F = 49.1$, $p < 0.0001$) are significant while practice group ($F = 0.14$, $p = 0.71$) are not. There is no significant interplay between them ($F = 0.17$, $p = 0.68$).

Partial η^2 values were computed for the Test 1 effect to determine effect sizes on code writing scores. The effect size for Test 1 scores is $\eta^2 = 0.265$, which is a large effect size. Cohen's d on the non-code-writing portion of Test 2 for PG1 vs. PG2 is 0.28. In addition to being much smaller than the difference on coding exercises, there is no significant evidence that practice on coding exercises contributed to the differences between groups on the MCQ

and coding portion of Test 2.

To further explore practice, note that both PG1 and PG2 can be further subdivided. PG1 includes students who did not practice at all as well as students who practiced some but all exercises. Another ANOVA was performed between these groups and found no significant difference independent of Test 1 scores ($F = 0.12$, $p = 0.73$). This indicates that students who do not practice coding do not show distinguishable effects from students who practice some. Similarly, PG2 includes students who scored full marks on every exercise, as well as students who tried every exercise but did not earn full marks on some. Here we also fail to find a significant difference between perfect scores and others ($F = 0.56$, $p = 0.46$). This is more evidence that the distinguishing factor here is whether a student opts to attempt every exercise or choose to forgo some exercises during practice.

Chapter 6

Survey Results

We felt that a survey on CodeWorkout to students was one of the best ways to test how well it is doing on its goals. After all, students are both the primary users and beneficiaries of CodeWorkout. CodeWorkout was used more extensively in Spring 2016 for the CS 1114 course. This is the ‘Introduction to Software Design’ course primarily for CS majors and this course featured a lot of Java programming. CodeWorkout was used in this course to facilitate drill-and-practice type of assignments as well exams involving coding. The survey was distributed to 378 students in that course in the first week of April. At that point, these students had already been using CodeWorkout for multiple assignments (i.e. workouts) for more than two months.

6.1 Survey method

The survey was designed using the Qualtrics online survey system and was distributed anonymously via email to all the students in the course. The survey was available completely online and was distributed only after approval from Virginia Tech’s IRB. Overall, there were nineteen questions and the vast majority of these were statements to which the students indicated the degree of their agreement on a five-point Likert scale. The available levels of agreement were Strongly Agree, Agree, Neither, Disagree and Strongly Disagree. In order to ascertain the consistency of results, these statements were both negatively and positively phrased. These statements deal with various topics pertaining to how well CodeWorkout was meeting its objectives, such as the usability of its interface, ease of access and usefulness of feedback. In Figure 6.1, we summarize question 16 that asked students about their previous programming experience. The mean for this question was 2.45 and this indicates that most students did not have extensive prior programming experience, especially not in Java.

The remaining three free-form text answer questions were used to gauge initial impressions, perceived drawbacks and overall usefulness of CodeWorkout. 179 students responded to the survey and Qualtrics was used to aggregate these responses.

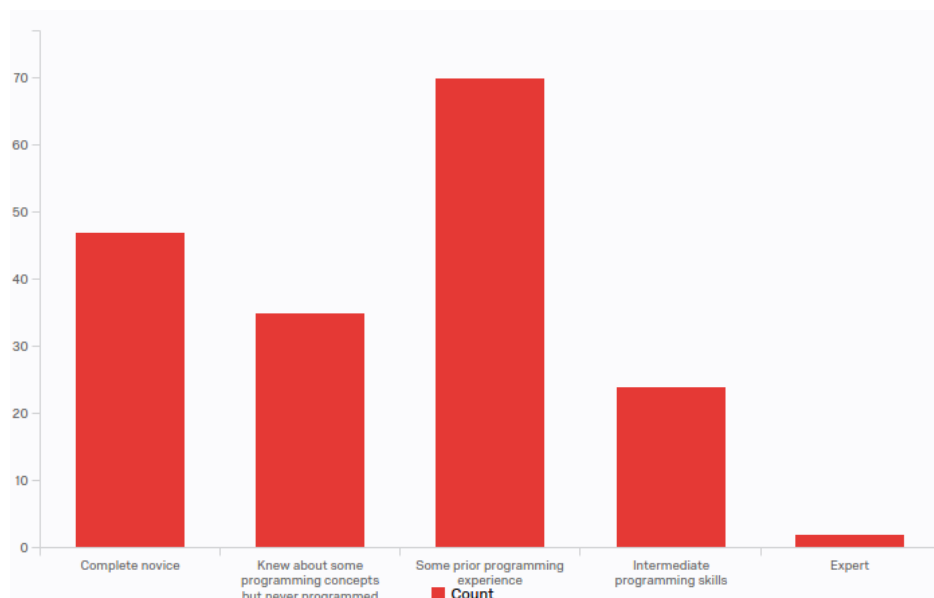


Figure 6.1: Prior programming experience of students

6.2 Likert-Scale Question Results

Table 6.1 lists out the mean scores for the Likert-scale questions and their variances. The ‡ in the table indicates negatively phrased questions. Overall results showed that students have a very positive impression of CodeWorkout and it has been beneficial for them. The high means for questions 1, 2, 6, 8, 9, 10 and 15 were especially notable. These questions primarily dealt with the usability and ease of access of CodeWorkout, two of its primary goals, and CodeWorkout was shown to have done well in these. The hugely positive response to these questions clearly pointed out that most students feel that CodeWorkout is both highly usable and accessible. Responses to two of the negative questions, Q3 and Q12, further supported this since these have a low mean response of 2.39 and 2.21 respectively. The first of these dealt with the usability of its coding interface and the last dealt with the navigability of the system. The low mean to these negatively-phrased questions also indicated that CodeWorkout does not impose any unnecessary burden to student learning. Though we have not yet performed any controlled user studies on CodeWorkout, these results are highly indicative of its usability and ease of access. The high mean scores of 4.17 in Q8 (regarding the continuous availability of CodeWorkout) and 4.00 in Q15 (regarding ease of course enrollment) is an affirmation of this. The highest mean of all (4.30 in Q6) indicated that CodeWorkout was better to use than traditional paper-based settings for coding.

6.2.1 Results from Likert-Scale Questions

Question		% of agree- ment or dis- agreement	Mean	Variance
<i>Usefulness</i>				
9	CodeWorkout is helpful to use for learning introductory programming	82.5% agreed or strongly agreed	3.99	0.90
11	The programming practice I get from CodeWorkout has improved my programming skills	76.9% agreed or strongly agreed	3.82	0.87
13	I believe that practicing on CodeWorkout will positively impact my performance on exams in this course, and on my grade in this course	79.2% agreed or strongly agreed	3.86	0.79
14	I wish to use CodeWorkout in my next programming course	64.0% agreed or strongly agreed	3.66	1.12
<i>Usability</i>				
1	It was easy for me to submit my solutions to exercises on CodeWorkout	94.3% agreed or strongly agreed	4.29	0.47
6	Practicing and submitting exercises on CodeWorkout is simpler as compared to paper-based systems	88.7% agreed or strongly agreed	4.30	0.61
10	I find CodeWorkout's coding interface easy to use	85.9% agreed or strongly agreed	4.10	0.53
12	I found it difficult to write my solutions for the exercises on CodeWorkout because of its interface‡	76.4% disagreed or strongly disagreed	2.21	0.76
<i>Access and navigability</i>				
2	It was simple for me to login to CodeWorkout for the first time	82.4% agreed or strongly agreed	4.08	0.78
3	I was difficult for me to navigate to the exact course content that I was looking for on CodeWorkout‡	66.3% disagreed or strongly disagreed	2.39	1.07

↓ Continued on next page ↓

Table 6.1 – continued from previous page

Question		% of agree- ment or dis- agreement	Mean	Variance
5	I was able to access my answers to previous assignments and exercises without complications	61.0% agreed or strongly agreed	3.56	1.09
8	CodeWorkout is always available and accessible to me	89.3% agreed or strongly agreed	4.17	0.49
15	It was easy for me to find and enroll in my course on CodeWorkout	79.7% agreed or strongly agreed	4.00	0.60
<i>Feedback</i>				
4	The automatic feedback generated by CodeWorkout was slow‡	40.6% dis- agreed or strongly disagreed	2.87	0.95
7	The compilation error messages that CodeWorkout gives are useful for fixing syntax errors in my code	58.4% agreed or strongly agreed	3.43	1.22

Table 6.1: Likert-survey questions means and variances

The only questions for which the means were not as strongly supportive were Q4 and Q5. The first question is a negatively phrased question asking how slow is CodeWorkout’s feedback. The response to this has a mean of 2.87 (lower the better). While this certainly indicated that most students felt that the system was not slow, it does mean that a substantial minority of students felt that its speed can be improved. The positively phrased question with the lowest mean was Q5. It was a question that was directed towards a specific feature of CodeWorkout that allows students view their work in past assignments. While the mean of 3.56 is certainly not bad and indicated that the feature works, it also conveys that we could do more to improve this feature. Some students even gave specific suggestions regarding this in the free-form text questions.

An overwhelming majority of the respondents for Q1 (94.3%) said that they either agreed or strongly agreed to the statement that it was easy to submit exercise solutions in CodeWorkout. CodeWorkout’s coding interface has received special praise: more than 85% of the respondents said that it was easy to write code with it.

What is surprising is that even though CodeWorkout is still a system that is very much under development, a majority of the students (64.0%) expressed their wish to use CodeWorkout in

their next course involving programming. More tellingly, 82.5% of the students in Q9 either agreed or strongly agreed that CodeWorkout is useful for learning introductory programming skills, which is its most important objective. Putting it a little differently, Q11 asks how much CodeWorkout has improved their programming skills. 76.9% of the responding students indicated that they either agreed or strongly agreed with this statement. Finally, with regard to usefulness in a traditional classroom setting, 79.2% of the respondents either agreed or strongly agreed that the practice afforded by CodeWorkout will positively impact their performance in this course. All these results present compelling evidence that CodeWorkout is a viable drill-and-practice system in introductory programming settings.

6.3 Free-form Text Question Results

Equally interesting results were obtained from the last three questions in the survey. The first of these three asked students about their initial impressions about CodeWorkout at first use. The next question asked whether it impeded their learning of programming in any way. The final question asked students about their overall opinion about CodeWorkout and we were satisfied with the numerous positive reviews that we obtained. In the subsequent subsections, we discuss these responses and analyze them to determine what their implications are. To give the reader a glimpse, we quote ten responses verbatim for each of the three text questions. Please note that the spelling and grammatical mistakes in these responses are those of the students, since we present them unchanged.

6.3.1 Impressions

The first free-form text question was: “What were your initial impressions about CodeWorkout as a programming practice system?”. 167 of the total 179 respondents to the survey answered this question. From the student responses to this question, we can see that a large majority students had a positive first experience.

A hundred of the responses to this question indicated a positive impression. An additional Twenty eight responses were mostly positive impressions with some suggestions for improvement, mainly dealing with improving the error feedback. Two of the responses indicated that the respondents felt neutral towards CodeWorkout. Five of the responses indicated a negative impression about CodeWorkout. Nine of the responses commented on CodeWorkout’s similarity to previously encountered code grading systems, especially CodingBat. Four of these indicated that the respondents were confused because of CodeWorkout’s color scheme being different from that of Greenfoot. Finally, eleven of the responses were either not relevant or were not understandable. The following are some of the sample responses:

- *It was helpful to see the tests ran against my code to see where I went wrong. The error messages also help me find the faults in my code*
- *Simple and gave feedback (which is invaluable).*

- *A little bulky and cluttered due to the assignments being on the left, the question being on the top and feedback being on the right.*
- *I like how it feels like an online game. Something that makes me want to achieve more and fill out the progress bar.*
- *It was fun to be given a problem, develop a solution, and immediately receive feedback about the task I completed, which was a welcome change from the needed practice in other formats in the course.*
- *I really enjoyed it. I felt like it was a great way to help actually practice my coding skills. I personally go back to codeworkout when I am studying for tests or may have forgotten how to properly code something.*
- *It was similar to other programming practice systems I have used in the past. I liked the set up and found it easy at first, and I was able to see how programming increases in difficulty as time goes on.*
- *Interface was a little confusing because it is different (colors, organization) than Green-Foot.*
- *CodeWorkout is a helpful programming practice system, and its interface is much easier to use than something like coding bat. I think it might be helpful if there were extra practice problems that we could do using the interface just like on CodingBat.*
- *Very easy to use, problems are challenging, but doable, and short enough that they can improve one's ability to problem solve without feeling overwhelmed. // This thing is great!*

Several students commented particularly on the immediate feedback that was made available to them and how useful this was since it enabled greater practice. This re-affirms that drill-and-practice systems are useful in introductory programming courses. In fact, a small number of the students who had some prior programming experience mentioned how easy it was for them to get started on this system because of its resemblance to CodingBat. They also liked the concept of multiple submissions since it let them practice in an unrestricted manner. One of the students even mentioned how pleased he or she was to use such a quick feedback system, and how previous programming courses did not have any such thing. Many students also remarked on how easy and simple to use it was. Our personal favorite response to this question was that CodeWorkout “*looked legit*”, which some might say is the highest praise possible.

6.3.2 Impediments

The next question was: "What were the aspects of CodeWorkout that you found to be an impediment to your learning?". With this question we intended to capture the shortcomings

of CodeWorkout, as perceived by students, and use it as springboard for future improvement. 164 of the total 179 students who completed the survey answered this question. Forty respondents to this question indicated that there were no drawbacks at all to this system. Eighty four respondents indicated that there were only one or two drawbacks with the system, mostly to do with the speed of the feedback, navigability and the clarity of error messages. Only three respondents indicated CodeWorkout was deeply flawed and there were numerous errors. Interestingly, eighteen of the respondents pointed to drawbacks that were not in CodeWorkout's purview; they raised issues like lack of appropriate instructions in using CodeWorkout and some of the problems being too difficult for them. Twelve of them gave responses that were either not relevant or were not understandable.

The following are some of the sample responses:

- *There is little direction for helping to debug your code.*
- *Sometimes slow in providing feedback, did not show alternative solutions after submitting a correct answer, no hints provided as to how my code could be improved.*
- *Code workout was no problem, the questions themselves had me in knots for a long while.*
- *Many of the concepts expected for CodeWorkout solutions are not detailed in class - general topics are covered but not enough to satisfy the CodeWorkout grader. I think it would be more beneficial to learning if CodeWorkout were sometimes done in class and thoroughly explained, also so students can ask questions as the solution is being generated.*
- *Nothing*
- *Nothing sever, I think that CodeWorkout does its job well. It will teach someone how to solve simple problems, while leaving the more serious and difficult assignments with multiple files to a site such as WebCat.*
- *None*
- *Just having to go to another website on top of the several that I already have to go to was an impediment.*
- *There were no facets of CodeWorkout that impeded me. Most of my difficulties came from my own inexperience in coding. Maybe the instructions on certain problems could be more thoroughly explained, or offer tips.*
- *Nothing on the site explicitly felt like an impediment. One thing I would perhaps suggest is showing an example solution after the student correctly completes the problem once, though. Often times, a student might over-complicate a problem when there's a much*

simpler solution, and it helps to see another's code to learn different strategies from them.

A large portion of students in the CS 1114 course complained about the lack of extensive feedback when their solution did not work correctly and the lack of hints. This is understandable because most of the students using the system were first time Java programmers and in general may find compiler error messages to be cryptic. The same goes for test cases: students in this course were not able to figure out why their code was repeatedly failing a few test cases. Unfortunately, this situation arose because hints and negative feedback at the test case level were not specified for the exercises *even though CodeWorkout had the capability to do so*. Negative feedback and hints were not provided in the exercises in this course because of lack of time than anything else.

From the student feedback to this question, we understand that they had some difficulties in working out the computational logic for the exercise and they were frustrated that CodeWorkout couldn't help them out. This, of course, was not an impediment *per se* to learning of students but is an interesting direction to feedback that we did not consider before. Overall, the responses to this negatively-phrased question were rather positive and this reinforces the inference made from the Likert-scale questions that CodeWorkout does not pose any barriers to student learning. Per Pieterse's paper, CodeWorkout does indeed satisfy one of the major requirements in order to be successful at the level of MOOCs.

6.3.3 Overall usefulness

The final question in the survey was: "Overall, how useful do you think CodeWorkout is to your learning and improvement of programming skills?". In essence, we asked the students how successful CodeWorkout was in fulfilling its grand objectives. All of the 179 of the respondents to the survey answered this question. Of all three free-form text questions in the survey, the responses to this question were the most favorable. 151 of the respondents indicated that CodeWorkout was useful for their learning of programming skills. Nine of the respondents indicated that CodeWorkout was useful but pointed out flaws like its feedback mechanism that needs fixing. Four of the responses were decidedly neutral and only five respondents indicated that CodeWorkout was not useful to them. Five of the responses were either not relevant or were not understandable. The few negative replies to this question pointed out the lack of certain features like more descriptive feedback or display of alternative working solutions. We would like to point out that these are features we are already building into CodeWorkout for next semester. The following are some of the sample responses:

- *It helps me learn*
- *Useful for studying.*
- *I think it's been quite useful for learning*

- *CodeWorkout hasn't helped me much in terms of improving my programming skills. It is good practice, and if hadn't had prior programming experience when using this website I would have benefited much more from it.*
- *Very useful*
- *Pretty useful*
- *CodeWorkout is good because Dr. Edwards sets our tests up using some CodeWorkout questions. Having the ability to practice on an interface that is the same as the interface that is used on the test is really nice to have.*
- *It's good. They bite sized logic and code is a nice break from the larger projects we have to do.*
- *on a scale from 1-10, it would be a 7*
- *It gives an opportunity to practice standard coding skills in a non graphical context. Personally I think that exposure to coding in this manner will be much more reflective of future/actual coding we will do.*

Since a large number of the respondents said the same thing, viz., that CodeWorkout was very useful to their learning of programming, we chose responses that mostly bucked this trend. Students also gave out some suggestions for how CodeWorkout could be improved in the next run. One example involved the lack of practice exercises on the course in CodeWorkout. Ironically, the practice workout for this course was released just a few days before this survey and most students were simply not aware of it. Regardless, this indicates a clear wish on behalf of students for more practice exercises on CodeWorkout. Since CodeWorkout already has a collection of programming exercises, this would be easy enough to provide if the situation calls for it.

Chapter 7

Conclusion

This thesis has discussed the design and implementation of CodeWorkout a new drill-and-practice system for introductory programming. The motivation and necessity for such a system, especially in universities, were explained. The various features and facilities of the system, which included the following, were explained:

- Gradual engagement
- Flexible and secure permission model
- Extended drill-and-practice support
- Simultaneously providing Gym and exercise courses
- Exercise organization features
- Collaborative and active learning features

The various implementation details regarding how this can be achieved were also discussed. There included the core autograding facility which can concurrently grade many simultaneous submissions through its multithreaded capabilities. The system also provides features where instructors can share workouts with other instructors. There was also discussed the open and accessible nature of CodeWorkout stemming from the gradual engagement policy that it follows. The other aspects of CodeWorkout like its many course management features and facilities for collaborative learning were discussed.

7.1 Contributions

Simply put, the main contribution of this work is the CodeWorkout system. It incorporates under one roof a host of features and functionalities found in other drill-and-practice systems but it also does have a bevy of its own unique features. CodeWorkout's objectives are admittedly grand but its implementation has borne it out, as shown by its initial results.

No other related system even remotely offers the same amount of capabilities and usefulness for introductory programming courses. Unlike all other drill-and-practice systems, CodeWorkout's viability has been shown in various settings like exams, assignments and general programming practice. Its potential in CS Education research through data mining of the different kinds of information about student behavior in various scenarios was shown. The most important benefit of CodeWorkout is that it hosts under one roof the best features of various other drill-and-practice systems while adding its own unique features to offer a comprehensive platform for programming practice.

Some of the key findings of this work through data analysis of student performance on exercises:

- We compared the performance of students programming exercises in an exam as compared to their performance in an optional practice workout that was offered before it. We showed that the extra coding practice on CodeWorkout improves the programming skills of students on related topics.
- We found that there is a strong relation between the number of exercises fully completed and the scores of students. We also found that this relation is much stronger than the same relationship for the number of attempts.
- We considered the performance of students in exam MCQs and compared it to their performance in programming exercises in the same setting. We found there is a strong positive relation between the two and this implies that multiple-choice questions are also beneficial to examine students in a drill-and-practice system.

7.2 Future Work

The grand objectives of CodeWorkout mean that even though much has been accomplished on it in the past there remains so much that can be done to make it the ultimate learning platform.

- **An IRT-based adaptive system:** CodeWorkout is planning to implement ideas from Item Response Theory (IRT)[46] to refine the student-contributed exercises and hints. When an exercise has been practiced by potentially a thousand students, we can find the discrimination and difficulty associated with the question to automatically assess its quality. Questions that are too difficult or are too easy can be quickly found out from the IRT data gathered for exercises. Similarly, the discrimination enforced by these questions can also be automatically gauged by CodeWorkout. These two together can automatically help determine how good an exercise is. This can be leveraged by the system to automatically weed out bad student-written exercise and help come up with a repository of high quality exercises. In a similar vein, hints can also be tracked to see how well they help students in fixing their missing test cases and good hints can be determined.

- **Anti-plagiarism support:** Plagiarism of code is the one of major problems in introductory programming courses.[9] Dishonest students either copy one another's code or use the Internet to obtain a solution from the web that magically solves their problems. This is all the more prevalent in online systems since they feel that there is no one watching over them. While MOOCs may not require such features, institutions that offer grades for its student need such features. Anti-plagiarism mechanisms are all the more difficult to implement in drill-and-practice systems because they deal with small programs wherein duplicates are difficult to weed out.
- **LTI integration:** Since CodeWorkout is primarily a system for education, it would be beneficial if it could interact with other such system. Learning Tools Interoperability (LTI) goes a long way in helping and we hope to implement LTI 2.0 so we can embed CodeWorkout inside Learning Management Systems and interact with other Smart Learning Contents.
- **Extended language support:** CodeWorkout currently only support Java, Python and Ruby. Since CodeWorkout uses a referential testing model for its autograding, it can be extended to support all major programming languages. Specifically we look towards implementing support for C++, JavaScript and Swift in the future.
- **Data analytics support:** CodeWorkout can incorporate the various data analytics modules for education that can help analyze student performance across various attempts, exercises and topics. This in turn can help the system gauge the progress of the student and act as an intelligent tutor by recommending other practice exercises.
- **Charts and Graphs:** It would be useful if CodeWorkout can provide visual results of its statistical data to both students and instructors. One example would be the a line graph that can show the progress of a student score in an exercise over a series of attempts. Another example, for instructors, would be the progression of the average class score for an exercise over the duration of its availability. Visual tools like these will help instructors gain better insight into the current state of their students.

All in all, there are great expectations for CodeWorkout and we hope that the development of CodeWorkout continues. The future is indeed exciting.

Bibliography

- [1] N. Barry. Gradual engagement, <http://www.uxbooth.com/articles/a-lesson-in-gradual-engagement>, last accessed 04-07-2016.
- [2] M. Bleigh. Omniauth, <http://intridgea.github.io/omniauth/>, last accessed 04-07-2016.
- [3] G. Braught and J. Midkiff. Tool design and student testing behavior in an introductory java course. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, pages 449–454. ACM, 2016.
- [4] P. Brusilovsky, S. Edwards, A. Kumar, L. Malmi, L. Benotti, D. Buck, P. Ihantola, R. Prince, T. Sirkiä, S. Sosnovsky, J. Urquiza, A. Vihavainen, and M. Wollowski. Increasing adoption of smart learning content for computer science education. In *Proceedings of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference*, ITiCSE-WGR '14, pages 31–57. ACM, 2014.
- [5] K. Buffardi and S. H. Edwards. Introducing codeworkout: An adaptive and social learning environment (abstract only). In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 724–724. ACM, 2014.
- [6] G. Colvin. Talent is overrated: What really separates world-class performers from everybody else, 2016.
- [7] CS@VT. Virginia tech’s codingbat, <http://codingbat.cs.vt.edu/codingbat/>, last accessed 04-06-2016.
- [8] J. Cuny. Cs education: Catching the wave. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, pages 3–3. ACM, 2016.
- [9] C. Daly and J. Horgan. Patterns of plagiarism. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '05, pages 383–387. ACM, 2005.
- [10] P. Denny, D. Cukierman, and J. Bhaskar. Measuring the effect of inventing practice exercises on learning in an introductory programming course. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research*, Koli Calling '15, pages 13–22. ACM, 2015.

- [11] P. Denny, A. Luxton-Reilly, and J. Hamer. The peerwise system of student contributed assessment questions. In *Proceedings of the Tenth Conference on Australasian Computing Education - Volume 78*, ACE '08, pages 69–74, Darlinghurst, Australia, Australia, 2008. Australian Computer Society, Inc.
- [12] P. Denny, A. Luxton-Reilly, E. Tempero, and J. Hendrickx. Codewrite: Supporting student-driven practice of java. In *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11, pages 471–476. ACM, 2011.
- [13] S. Edwards. Codeworkout wiki, <http://web-cat.org/group/codeworkout>, last accessed 04-07-2016.
- [14] S. H. Edwards. Improving student performance by evaluating how well students test their own programs. *J. Educ. Resour. Comput.*, 3(3), Sept. 2003.
- [15] S. H. Edwards. Webcat, <https://sourceforge.net/projects/web-cat/>, last accessed 04-06-2016.
- [16] S. H. Edwards and K. Panamalai Murali. Codeworkout. <https://github.com/web-cat/code-workout>, 2016.
- [17] S. H. Edwards, Z. Shams, and C. Estep. Adaptively identifying non-terminating code when testing student programs. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 15–20. ACM, 2014.
- [18] I. T. Force. OAuth2, <https://tools.ietf.org/html/rfc6749>, last accessed 04-07-2016.
- [19] J. Gikas and M. M. Grant. Mobile computing devices in higher education: Student perspectives on learning with cellphones, smartphones & social media. *The Internet and Higher Education*, 19:18–26, 2013.
- [20] C. W. Group. Canterbury questionbank, <http://web-cat.org/questionbank/>, last accessed 04-07-2016.
- [21] D. H. Hansson. Ruby on rails, <http://rubyonrails.org/>, last accessed 04-07-2016.
- [22] M. Haverbeke. Codemirror, <https://codemirror.net/>, last accessed 04-07-2016.
- [23] B. Hilkert. Suckerpunch, https://github.com/brandonhilkert/sucker_punch, last accessed 04-07-2016.
- [24] J. Hollingsworth. Automatic graders for programming classes. *Commun. ACM*, 3(10):528–529, Oct. 1960.
- [25] M. J. Hull, D. Powell, and E. Klein. Infandango: Automated grading for student programming. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, ITiCSE '11, pages 330–330. ACM, 2011.

- [26] P. Ithantola, T. Ahoniemi, V. Karavirta, and O. Seppälä. Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, Koli Calling '10, pages 86–93. ACM, 2010.
- [27] Q. Inc. Quora, <https://www.quora.com/>, last accessed 04-07-2016.
- [28] D. Jackson and M. Usher. Grading student programs using assyst. In *Proceedings of the Twenty-eighth SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '97, pages 335–339. ACM, 1997.
- [29] I. Kiselev. *Aspect-oriented programming with AspectJ*. Sams, 2003.
- [30] F. M. Lord. *Applications of item response theory to practical testing problems*. Routledge, 1980.
- [31] P. Mohan. Student perceptions of various hint features while solving coding exercises. Master's thesis, Virginia Tech, VTechWorks, September 2015.
- [32] C. Mullins. Responsive, mobile app, mobile first: Untangling the ux design web in practical experience. In *Proceedings of the 33rd Annual International Conference on the Design of Communication*, SIGDOC '15, pages 22:1–22:6. ACM, 2015.
- [33] A. Ng. Coursera, <https://www.coursera.org/>, last accessed 04-07-2016.
- [34] Oracle. Java security manager. <https://docs.oracle.com/javase/8/docs/api/java/lang/SecurityManager.html>, 2015.
- [35] A. Papancea, J. Spacco, and D. Hovemeyer. An open platform for managing short programming exercises. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research*, ICER '13, pages 47–52. ACM, 2013.
- [36] N. Parlante. Codingbat, <http://codingbat.com/>, last accessed 04-06-2016.
- [37] E. Phoenix. Puma, <http://puma.io/>, last accessed 04-07-2016.
- [38] V. Pieterse. Automated assessment of programming assignments. In *Proceedings of the 3rd Computer Science Education Research Conference on Computer Science Education Research*, pages 45–56. Open Universiteit, Heerlen, 2013.
- [39] Plataformatec. Devise, <https://github.com/plataformatec/devise>, last accessed 04-07-2016.
- [40] J. Spacco, D. Hovemeyer, W. Pugh, F. Emad, J. K. Hollingsworth, and N. Padua-Perez. Experiences with marmoset: Designing and using an advanced submission and testing system for programming courses. In *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITICSE '06, pages 13–17. ACM, 2006.

- [41] K. P. M. Stephen Edwards. Codeworkout, <https://codeworkout.cs.vt.edu>, last accessed 04-07-2016.
- [42] I. Sysoev. Nginx, <https://www.nginx.com/>, last accessed 04-07-2016.
- [43] E. Thompson, A. Luxton-Reilly, J. L. Whalley, M. Hu, and P. Robbins. Bloom's taxonomy for cs assessment. In *Proceedings of the Tenth Conference on Australasian Computing Education - Volume 78*, ACE '08, pages 155–161, Darlinghurst, Australia, Australia, 2008. Australian Computer Society, Inc.
- [44] Twitter. Bootstrap, <http://getbootstrap.com/>, last accessed 04-07-2016.
- [45] C. Wilcox. The role of automation in undergraduate computer science education. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 90–95. ACM, 2015.
- [46] R. Yale and Jasig. Cas, <https://www.unicon.net/opensource/cas>, last accessed 04-07-2016.
- [47] S. Zweben. *Computing Degree and Enrollment Trends - 2012-2013 CRA Taulbee Survey*, Computing Research News, 2013.

Appendix A

Text of Survey

Preamble

CodeWorkout Feedback Survey

This survey includes questions about the CodeWorkout system used in class this semester. The results will be used for research purposes and course improvement. Please complete all items, even if you feel that some are redundant. This may require about 5 minutes of your time.

Usually it is best to respond with your first impression, without giving a question much thought. Your answers will remain confidential, and will not affect your grade in any way.

Your responses (without your name, of course--your identity is always strictly confidential) may be made available for use by other education researchers outside the course who are conducting studies on computer science education.

Your participation is voluntary. If you do not wish to participate, simply do not fill out the survey. **You must be 18 or older** to take part in this research--if you are under 18, please skip this survey.

Thank you for your participation.

Questions

Please respond to the following statements by selecting the option that best describes your opinion.

	Strongly disagree	Disagree	Neither	Agree	Strongly agree
It was easy for me to submit my solutions to exercises on CodeWorkout	☐	☐	☐	☐	☐
It was simple for me to login to CodeWorkout for the first time	☐	☐	☐	☐	☐
I was difficult for me to navigate to the exact course content that I was looking for on CodeWorkout	☐	☐	☐	☐	☐
The automatic feedback generated by CodeWorkout was slow	☐	☐	☐	☐	☐
I was able to access my answers to previous assignments and exercises without complications	☐	☐	☐	☐	☐
Practicing and submitting exercises on CodeWorkout is simpler as compared to paper-based systems	☐	☐	☐	☐	☐
The compilation error messages that CodeWorkout gives are useful for fixing syntax errors in my code	☐	☐	☐	☐	☐

CodeWorkout is always available and accessible to me	☐	☐	☐	☐	☐
CodeWorkout is helpful to use for learning introductory programming	☐	☐	☐	☐	☐
I find CodeWorkout's coding interface easy to use	☐	☐	☐	☐	☐
The programming practice I get from CodeWorkout has improved my programming skills	☐	☐	☐	☐	☐
I found it difficult to write my solutions for the exercises on CodeWorkout because of its interface	☐	☐	☐	☐	☐
I believe that practicing on CodeWorkout will positively impact my performance on exams in this course, and on my grade in this course	☐	☐	☐	☐	☐
I wish to use CodeWorkout in my next programming course	☐	☐	☐	☐	☐
It was easy for me to find and enroll in my course on CodeWorkout	☐	☐	☐	☐	☐

How would you rate programming skills before joining this course?

- ☐ Complete novice
- ☐ Knew about some programming concepts but never programmed till now
- ☐ Some prior programming experience
- ☐ Intermediate programming skills
- ☐ Expert

What were your initial impressions about CodeWorkout as a programming practice system?

What were the aspects of CodeWorkout that you found to be an impediment to your learning?

Overall, how useful do you think CodeWorkout is to your learning and improvement of programming skills?

Appendix B

Survey Report

#	Question	Strongly disagree	Disagree	Neither	Agree	Strongly agree	Total Responses	Mean
1	It was easy for me to submit my solutions to exercises on CodeWorkout	<u>1</u>	4	5	<u>100</u>	<u>68</u>	178	4.29
2	It was simple for me to login to CodeWorkout for the first time	<u>1</u>	<u>14</u>	<u>16</u>	<u>89</u>	<u>57</u>	177	4.06
3	I was difficult for me to navigate to the exact course content that I was looking for on CodeWorkout	<u>30</u>	<u>88</u>	<u>26</u>	<u>29</u>	<u>5</u>	178	2.39
4	The automatic feedback generated by CodeWorkout was slow	<u>10</u>	<u>62</u>	<u>51</u>	<u>49</u>	<u>5</u>	177	2.87
5	I was able to access my answers to previous assignments and exercises without complications	<u>6</u>	<u>26</u>	<u>37</u>	<u>78</u>	<u>30</u>	177	3.56
6	Practicing and submitting exercises on CodeWorkout is simpler as compared to paper-based systems	<u>1</u>	5	<u>14</u>	<u>78</u>	<u>80</u>	178	4.30
7	The compilation error messages that CodeWorkout gives are useful for fixing syntax errors in my code	<u>9</u>	<u>34</u>	<u>31</u>	<u>79</u>	<u>25</u>	178	3.43
8	CodeWorkout is always available and accessible to me	<u>0</u>	<u>6</u>	<u>13</u>	<u>103</u>	<u>56</u>	178	4.17
9	CodeWorkout is helpful to use for learning introductory programming	<u>7</u>	<u>7</u>	<u>17</u>	<u>96</u>	<u>51</u>	178	3.99
10	I find CodeWorkout's coding interface easy to use	<u>1</u>	4	<u>20</u>	<u>105</u>	<u>48</u>	178	4.10
11	The programming practice I get from CodeWorkout has improved my programming skills	<u>5</u>	<u>15</u>	<u>21</u>	<u>103</u>	<u>34</u>	178	3.82
12	I found it difficult to write my solutions for the exercises on CodeWorkout because of its interface	<u>26</u>	<u>110</u>	<u>24</u>	<u>14</u>	<u>4</u>	178	2.21
13	I believe that practicing on CodeWorkout will positively impact my performance on exams in this course, and on my grade in this course	<u>5</u>	<u>11</u>	<u>21</u>	<u>108</u>	<u>33</u>	178	3.86
14	I wish to use CodeWorkout in my next programming course	<u>10</u>	<u>13</u>	<u>41</u>	<u>78</u>	<u>36</u>	178	3.66
15	It was easy for me to find and enroll in my course on CodeWorkout	<u>1</u>	<u>6</u>	<u>29</u>	<u>98</u>	<u>44</u>	178	4.00

3. What were your initial impressions about CodeWorkout as a programming practice system?

Text Response

It was helpful to see the tests ran against my code to see where I went wrong. The error messages also help me find the faults in my code

I enjoyed the automatic feedback and grade.

pretty good

Loved it. So convenient to get immediate feedback. n

-

It was difficult knowing what some of the questions were asking, but once I figured it out it wasn't bad.

It was decent. Errors are too vague.

It was easy to use

It's not a bad system, if it works correctly. Many times this was not the case.

Simple and easy to use

It was a reasonable way to help learn good programming skills and technique.

It was very useful and easy to code into

it is simple and easy to use.

It looks a bit hard because it is different from greenfoot; however, once you get a bit of a foundation, it is very helpful.

the first couple of problems was not exactly sure what was going on, but after a problem or two I figured out what it was saying and found it a good practice with good feedback.

I like the layout of the site and the immediate feedback on every question. Looking at the test cases also helped me to create a lot of the methods, because I wasn't always sure how the method should act from just the instructions on the top. It, however, is very hard to debug any logic errors while in CodeWorkout. I would often just output part of my solution so that I could check if a certain part of the method was working correctly. Maybe if there was a button to run the code once, instead of against all the tests, it can save the student time while debugging.

Good programming practice, hard to navigate

very good at programming practice

it was like codingbat, which i used in high school

Automatic feedback for simple codes is nice.

Simple and gave feedback (which is invaluable).

Pretty general submit your code and get results system

It is a good way to assess and practice programming knowledge.

It works well, and helps me apply the knowledge I have learned in class.

It seemed very helpful

I thought this would be a good way to practice the skills we are learning in class.

Pretty simple. Interface was good, but feedback was sometimes limited

a

Clean, easy-to-use interface Not the best feedback (should be more detailed and/or provide hints)

That it would be challenging

Good

At first I thought it was a good programming practice system.

It works well, yet Logging in can be slightly tedious

I'm not entirely sure. I think I liked it

Okay

I liked the fact that the problems were short and focused on one specific thing each.

Apprehensive

I thought it would be useful, but the only times there are practice questions available are for homework or test review. I think it would be more helpful if there was always some questions available for practice.

I thought it was very similar to CodingBat

It seemed very simple and easy to use.

It was exactly like codingbat.com, which is what i used in a previous course in high school, so i was very familiar and used to it.

I like the automatic testing and that it tells us the expected value and the value that resulted from our code.

A little bulky and cluttered due to the assignments being on the left, the question being on the top and feedback being on the right.

It seems very buggy and like certain things are hard to understand particularly the error messages.

Great way to practice

Good practice for programming

impressive

I thought it would be a simple practice system that would guide you through the code writing steps seeing as some of the problems on code workout were things that were recently covered in class or were somewhat foreign.

It was fun to be given a problem, develop a solution, and immediately receive feedback about the task I completed, which was a welcome change from the needed practice in other formats in the course.

Easy to use, easy to navigate through

It was pretty decent.

It seemed like an interesting idea

Great system, easy to use.

Liked how it would tell you what line errors were on.

I thought it was a well thought out system for simple conceptual problems.

CodeWorkout was nice since it allowed for unlimited submissions, but I don't like that the colors are different from what they are in Greenfoot because it got confusing.

I liked that you could submit answers to coding problems online and receive instant feedback.

It works pretty well for practicing programming logic although it's slow.

I like that it gives feedback and multiple submission are allowed.

I thought it was a good system that helps students understand basic programming by giving hints as to what needs to be fixed in their code. Relatively simple to use and clean interface.

Similar to other programming exercise websites.

I thought it was well designed

I knew it had potential in teaching me java.

The interface and accessibility are very easy to understand and useful but the compilation errors are very hard to understand and when you do get a solution the format of what you missed is difficult to understand. I also did not like that there was no type of assistance that could be given if you were not understanding the question unless you look outside of the program.

Easy to use

I thought that it was an interesting idea, as I had never worked with an automatic programming grading system before.

I really enjoyed it. I felt like it was a great way to help actually practice my coding skills. I personally go back to codeworkout when I am studying for tests or may have forgotten how to properly code something.

It was easy to understand and easy to begin practicing on.

It was very simple to use and gave somewhat quick feedback.

It seemed cool and convenient

The questions are a lot of the time confusing on exactly what am I doing and for being an introduction class the questions are nearly impossible for me to accomplish by myself. Even with help I still end up taking multiple hours to complete assignments.

It was similar to other programming practice systems I have used in the past. I liked the set up and found it easy at first, and I was able to see how programming increases in difficulty as time goes on.

It seemed like a great way to learn and practice specific subject material without the hassle of dealing with a class or large program. It was a great way to focus.

It was kind of cool, seeing the interface and the instant feedback.

I thought it was helpful

It was easy to use and fast

Boring.

it was an easy way to get automatic feedback on code

Sometimes I do not understand what exactly I have to type but I understand what I am supposed to do.

It seemed easy to use although the error messages were vague

It's very clean and sleek. Was easy to use. Would like if it was easier to navigate.

CodingBat is better

looks legit

Overall, it is really useful. But it would be better if there were more problems to practice that won't affect our grade. Because all the problems given are problems that affect our grade, it creates more stress on completing the problems.

it was alright

It was good, I had used CodingBat before and it was very similar. It was a useful way to practice coding.

CodeWorkout is a little confusing when it comes to finding the correct assignments as well as seeing old assignments.

it is a good interface, but the material wasn't quite kind to complete novices

I was initially impressed that programming could be condensed to such a short exercised form.

Interface was a little confusing because it is different (colors, organization) than GreenFoot.

It is a copy of coding bat.

It was formatted differently from what we did in class, but the content was similar.

It was easy to understand what you had to do, but in no way taught you anything. Also the first few times I did not know there was the side bar up top that told you what answers were right and it was frustrating.

I like how it feels like an online game. Something that makes me want to achieve more and fill out the progress bar.

It is very easy to use and has an appealing user interface.

Code workout was simple to use and easy to understand

It was easy to navigate and helpful, although the feedback I received wasn't as helpful as I hoped

Pleased to use a quick feedback simple problem system such as CodeWorkout, previous programming courses didn't have anything similar.

CodeWorkout is really nice because it has a simple aspect to it. It sort of gives you good feedback and it helped me to get a basic understanding of coding past classroom lectures, programs, and labs.

CodeWorkout is a helpful programming practice system, and its interface is much easier to use than something like coding bat. I think it might be helpful if there were extra practice problems that we could do using the interface just like on CodingBat.

This table has more than 100 rows. [Click here to view all responses](#)

Statistic	Value
Total Responses	162

4. What were the aspects of CodeWorkout that you found to be an impediment to your learning?

Text Response
Sometime the instructions for the code confused me
No help as to how to solve certain problems that I could not do.
page kind of feels a little squished at the bottom
n/a
sometimes the answer I saved got reset to a previous save state and I had to redo part of the solution. has happened 2 or 3 times
There is little direction for helping to debug your code.
There were some errors that were specific to CodeWorkout
When it didn't work. That hindered progress and discouraged correct practices.
-
The error messages weren't helpful in helping me find the problems with my code a lot of the time.
I think it would be a tremendous help if after the assignments date passed, it showed an example solution because there were times I did not know how to do a question at all, and still don't because it was never gone over
sometimes when I submitted my code or clicked on test, it erased all my code and I had to rewrite it
If it could give better feedback than just compilation errors it would help the LEARNING process more. For example, if there is a common mistake people make, have an explanation as to why it is wrong.
Every once in a while I do get stuck and I can figure out what I needed to do, then I do need to seek out help from a person who can explain what I am doing wrong
Having to wait for codeworkout to test against all of the test cases. If there was an option to run the method only once against parameters that the student can choose, it would make initial testing of the code much faster. Then the student only has to run the code against the tests after they think their code is working.
none
feedback
The compilation only points out the line of an error, not the specific point of the error on the line like in greenFoot4Sofia.
nothing
None.
It crashing when we tried to take a test.
Error feedback is too vague sometimes
More clarity in the error messages would help to fix programming mistakes.
The syntax error feedback is not always very descriptive.
Doesnt help if youre totally lost on a code
CodeWorkout can be very slow at times, especially if a lot of people are on it at once.
Feedback when I submitted code was vague at times
a
Sometimes slow in providing feedback, did not show alternative solutions after submitting a correct answer, no hints provided as to how my code could be improved
The exact details
During the first test and in the extra time given after I had huge difficulties in accessing the site and submitting my solutions.
I really don't like how if you get one line wrong and don't know how to fix it, you can't move on.
It doesn't tell you all your errors. Many times I don't know how to fix an error when one occurs
Sometimes it does weird things and won't let me get to assignments I need or see my grades
None really.
The error messages, highlighting the area where the problem is would be more helpful than just pointing to a line
The error messages are not very helpful.
The issues during test 1 , the slow automatic feedback
It gives feedback on the scenarios that you either get correct or incorrect
Sometimes the errors are not very specific when comparing the actual answer to the correct one
The compiler sometimes is not completely clear, and occasionally there is a problem with the root file screwing up the code even when the user has done nothing, e.g. the compiler hints that it cannot load some Test file, which is not my writing.
if there was an error it wouldn't really tell me, and the tests it ran weren't always clear and didn't really show why my answer wasn't working
The error messages are not very clear. They sometimes have nothing to do with the actual error in the code
Code workout was no problem, the questions themselves had me in knots for a long while.
The frustrating error messages and lack of method help.
I think it should be just practice rather than an assignment and just have them for each unit as very simple problems to help you get the hang of each skill
It's usually very slow, the errors sometimes don't point you to the exact problem in your code
N/A
The fact that CodeWorkout is sometimes slow really makes it tedious for checking to see if your code actually works. Especially if we are in class and everyone is submitting to

check to see if their code works, the system becomes very slow.

Sometimes it does get stuck or slow, and it is also a bit tricky to navigate to previous exercises, because you have to go your course first and sometimes I momentarily get lost. I would prefer if all the exercises completed for class showed up in the section about your exercises, even though they were completed as part of the course.

Longer code takes a longer time to process

If CodeWorkout went down.

navigating the area where you enter the code is annoying do to dual scroll bars. This is easily the worse thing about code workout.

When it checked for the possible solutions and had some of them as hidden, I didn't find that very beneficial.

Can't think of anything in particular.

The fact that it doesn't save work if you hit "next exercise" or at least prompt you. I lose work frequently due to mis-clicks.

I think the confusion as to how some syntax changed in CodeWorkout was confusing and I did not like it.

I wish the error messages were more specific or highlighted where on the line there was an issue.

Many of the concepts expected for CodeWorkout solutions are not detailed in class - general topics are covered but not enough to satisfy the CodeWorkout grader. I think it would be more beneficial to learning if CodeWorkout were sometimes done in class and thoroughly explained, also so students can ask questions as the solution is being generated.

Sometimes would not submit answers as the system took a long time to process and give feedback.

No feedback on why some exercises were not solved completely.

Nothing negatively impacted my learning

The "glitchiness", so to speak.

the fact that there were hidden solutions that even after getting correct you cannot view made it difficult to use code workout as a study tool.

Feedback on your code is hard to interpret.

I cant really think of anything, maybe make the description blurbs more robust, but even that's a stretch.

Overall, codeworkout is great, but it does have some downsides. For instance, sometimes the errors that it provides back are hard to understand. This in return makes it hard for me to figure out what I did wrong in my code. Another downside is that it sometimes is extremely slow to respond.

It allows me to go back and practice things as much as I want and has multiple problems for similar concepts allowing me to fully understand those concepts.

The only problem I have had with CodeWorkout is when a hidden answer was the only incorrect response and that didn't help with my debugging of my code.

syntax and constant feedback

CodeWorkout assumes I know more than I actually do, so I have an extremely difficult time completing assignments.

The only problem I found with CodeWorkout was unless you submit, it does not save your work. I would right down my ideas and possible solutions to the problem and then click next exercise and it would erase what I had written down. This was easily fixed by just submitting it before I was ready to submit it just so I would avoid losing work. However, sometimes I would forget to do so and lose my previous work.

The interface (css) of the webpage was broken in some areas (the banner and nav bars overlap the content, there are two scroll bars, ...) this was a distraction.

the questions are sometimes unclear and errors sometimes occur outside of my code lines, causing confusion.

None

None

None of them.

nothing

I like that I can submit multiple times however maybe some hints during practice problems or example solutions to code would help as well.

I wish the error messages were easier to interpret

Hard to navigate to the assignments you need to do.

It doesn't explain logic errors

nothing

Sometimes the error codes did not help on solving the problem. I had to search up to see if I could find a simpler explanation of my error.

?

The error messages were very general and I sometimes found it hard to figure out what I did wrong.

CodeWorkout has been confusing to use and often times I loose my work because it doesn't save easily.

I would like to see the "hidden/s"

A majority of the instructions for the various workouts were somewhat difficult to comprehend. My lack of knowledge of return types also impeded my ability to efficiently solve the exercises.

The feedback does not tell me exactly what is wrong with my code. If there is a specific word that causes the error, it does not highlight that word

None.

Using it for the first time it was difficult at first to understand how to use it.

sometimes the prompts were too broad

None that I can recall right now.

The error messages received are not very well explained so it's hard to fix some errors.

N/a

The fact that you really had to dive deep into the problem to understand what they were asking

Nothing sever, I think that CodeWorkout does its job well. It will teach someone how to solve simple problems, while leaving the more serious and difficult assignments with multiple files to a site such as WebCat.

Me being a beginner programmer, it was hard for me to jump right in to programming with CodeWorkout. There were really no walk-through questions that sort of gave me an idea of how to do some of the homework questions, and I wish there were some more tutorial questions.

A lot of times I was not sure how to fix that compiler errors I was getting. This may just be since I am brand new to java, but, that being said, if you want CodingWorkout to be helpful for new coders, then it might be helpful to give a little extra feedback in these intro course.

This table has more than 100 rows. [Click here to view all responses](#)

Statistic	Value
Total Responses	159

5. Overall, how useful do you think CodeWorkout is to your learning and improvement of programming skills?

Text Response
I think it helped me because it allowed me to experiment with trial and error to write a properly functioning code.
It was very useful in improving coding skills.
very
useful, I'd be worse off without it.
-
It definatly provided needed practice
It is a nice part, demanding skills learned, but could be improved.
It helps me learn
Not useful to learning. It has far too many bugs to be effective.
I think it was very helpful.
I think it is overall useful for new programmers.
It will definitely help
it is very useful
Very. I wish there were more exercises on more things than just algorithms.
I think it is useful, it gives me a change to practice concepts
Decent, occasionally buggy but overall beneficial
It helped me with programming but it would have been better if we were learning that kind of coding in class, especially when it came to the CodeWorkouts on strings.
I think that CodeWorkout is a very good way to learn and improve programming skills.
It was very useful to practice for tests.
It's kind of helpful.
pretty good
I really do feel like i am learning good programming with CodeWorkout.
it's really useful, i like it.
Useful for studying.
Very
Yes
Overall, it was useful and convenient.
I think it is very useful.
Quite useful
Very useful.
Useful. Would use it again.
Quite useful, test style but also helps for practice.
Very useful, more effective than submitting something to Scholar/Canvas and getting a grade a week later
I think it's very useful
It is a good site, but it may not be reliable in times of stress.
I think it is pretty helpful for learning.
It fairly useful. I definitely am glad I have it, but it could use some work.
I think it is pretty useful
can be useful, but is frustrating most of the time for the lack of explanation of what im doing wrong
I think it provides a good source of practice. It is the most beneficial kind of homework we have done so far.
I think it's been quite useful for learning
I think it has the potential to be very useful, but the way it was used in this course didn't help me much.
Useful but needs to be improved to run smoother
I find it more frustrating than useful because the questions are sometimes hard to understand, which causes me to write long, unnecessary code.
I think it is good for general practice that does not involve a full program
I think it is a useful tool and has helped me to be a better programmer.
I think if I didn't already take an intro to java course in high school, extensive use of codeworkout to get basic syntax and logic would be extremely useful.
Pretty useful
Well, this is my first experience and since nothing to compare it to. My grades will probably speak for it.
Moderately useful.
Good, I think if implemented the way I stated above it could help a lot with understanding each technique.

Pretty useful. Practice is key to learning programming and it's nice to have short problems rather than long ones for labs and programs. Good to focus on small, key skills.
Would be better if we could get answers
It is somewhat useful since some of the practice on it helped for the first exam. It would be better if the system gave a hint or two when we struggle to get it working. The directions on some of the problems were a bit vague and examples would help as well.
Very useful. It definitely built upon skills that I was using in the programming and lab assignments through repetition, and frequently made me consider problems in a novel context.
I feel like I learned better on the programming assignments
It is making a positive impact.
CodeWorkout hasn't helped me much in terms of improving my programming skills. It is good practice, and if hadn't had prior programming experience when using this website I would have benefited much more from it.
I thought it was very useful and very easy to use. I am very satisfied with how this course integrated codeworkout.
I think it will be somewhat useful in learning and improving of programming skills.
Very useful and I found the feedback it gives to be very helpful.
I think that the unlimited submissions is vital to allow us to have a learning process and a similar system with minor bugs fixed should be used moving forward.
I think it is good practice but the interface could be improved upon. It isn't always reliable and easy to understand what errors are occurring.
It's definitely helpful although truthfully any practice is helpful.
Upon seeing the answers I understood how to logic made sense, but I don't think it was very useful to the labs and programs assigned in addition.
I believe it has been useful since it helped me get to understand how to use some of the methods and get used to the syntax of the language.
Fairly useful. Allows application of concepts from class.
I think CodeWorkout is very useful to my learning and improvement of my programming skills
Decent, it helped apply things we learned from readings and in class.
i would rate it a 5 out of ten because its very easy to read and fairly quick to respond but the responses are of no real help when it comes to learning what you need to know to get it right next time.
It helped me a lot
I feel that it has been pretty useful. The instantaneous feedback with the results of the different tests is very helpful for getting a grasp on the concepts.
I overall feel like codeworkout is a great tool for me to learn and improve my programming skills. There are a few things that I feel like can be improved, but like I said, overall it's great.
Very useful because it allows me to practice what we will have on the tests and it helps me to understand what I am doing wrong.
Very useful
very
I do not think that it is helpful, it just stresses me out and I think that it helps minimally with my improvement of my programming skills
I find it helpful in improving my problem solving skills and developing new ways to complete the programming task at hand.
Very helpful
I think this had=s helped me to code, but I had to look up methods to use outside of what codeWorkout game me
It helped solidify some of the programming methods because each problem really only focused on one at a time (they were simple problems).
Very
Not useful.
very useful
I think it very useful overall.
very useful
I believe it is very useful and it has helped me out quite a bit this semester! Being able to practice coding in such a environment drills in the correct syntax and helps improve logic.
7/10
very useful
It could be much useful in providing other problems to solve it. It would also be nice if some of the problems (not counted for a grade) had solutions. Mostly, it would be nice seeing an example of the concept in a code that works so that you know if you are ball parking the code correctly or not.
it's useful however I would recommend having a reminder system, that sends a reminder that homework is due, a day before the actual due date.
Pretty useful
I like the idea of CodeWorkout but tend to practice more on CodingBat just because the interface is easier for me to understand.
on a scale from 1-10, it would be a 7
I don't believe it has noticeably improved my programming skills, but it is an excellent way to practice.
It is a good base for practicing programming with a more realistic compiler than GreenFoot
Useful.
I believe it is useful, however I would like the option to do other practice problems rather than only those that are assigned for homework.
It was helped and given me practice.
I like CodeWorkOut. It is very useful and easy to use. I also like how it gives me the feedback so I can fix my syntax error.

This table has more than 100 rows. [Click here to view all responses](#)

Statistic	Value
Total Responses	174

Appendix C

Virginia Tech IRB Approval

MEMORANDUM

DATE: April 5, 2016

TO: Stephen H Edwards, Manuel A Perez-Quinonez, Kevin John Buffardi, Krishnan Panamalai Murali

FROM: Virginia Tech Institutional Review Board (FWA00000572, expires January 29, 2021)

PROTOCOL TITLE: CodePractice: Developing Coding Skills Using Social and Adaptive Drill-and-Practice Exercises

IRB NUMBER: 13-270

Effective April 5, 2016, the Virginia Tech Institutional Review Board (IRB) Chair, David M Moore, approved the Amendment request for the above-mentioned research protocol.

This approval provides permission to begin the human subject activities outlined in the IRB-approved protocol and supporting documents.

Plans to deviate from the approved protocol and/or supporting documents must be submitted to the IRB as an amendment request and approved by the IRB prior to the implementation of any changes, regardless of how minor, except where necessary to eliminate apparent immediate hazards to the subjects. Report within 5 business days to the IRB any injuries or other unanticipated or adverse events involving risks or harms to human research subjects or others.

All investigators (listed above) are required to comply with the researcher requirements outlined at:

<http://www.irb.vt.edu/pages/responsibilities.htm>

(Please review responsibilities before the commencement of your research.)

PROTOCOL INFORMATION:

Approved As: **Expedited, under 45 CFR 46.110 category(ies) 7**

Protocol Approval Date: **March 13, 2016**

Protocol Expiration Date: **March 12, 2017**

Continuing Review Due Date*: **February 26, 2017**

*Date a Continuing Review application is due to the IRB office if human subject activities covered under this protocol, including data analysis, are to continue beyond the Protocol Expiration Date.

FEDERALLY FUNDED RESEARCH REQUIREMENTS:

Per federal regulations, 45 CFR 46.103(f), the IRB is required to compare all federally funded grant proposals/work statements to the IRB protocol(s) which cover the human research activities included in the proposal / work statement before funds are released. Note that this requirement does not apply to Exempt and Interim IRB protocols, or grants for which VT is not the primary awardee.

The table on the following page indicates whether grant proposals are related to this IRB protocol, and which of the listed proposals, if any, have been compared to this IRB protocol, if required.

Date*	OSP Number	Sponsor	Grant Comparison Conducted?
03/13/2013	12259505	National Science Foundation	Compared on 03/14/2013

* Date this proposal number was compared, assessed as not requiring comparison, or comparison information was revised.

If this IRB protocol is to cover any other grant proposals, please contact the IRB office (irbadmin@vt.edu) immediately.