CodeWrite: Supporting Student-Driven Practice of Java

Paul Denny, Andrew Luxton-Reilly, Ewan Tempero, Jacob Hendrickx
Department of Computer Science
The University of Auckland
Auckland, New Zealand
{paul,andrew,ewan}@cs.auckland.ac.nz,jhen095@auburn.edu

ABSTRACT
Drill and practice exercises enable students to master skills needed for more sophisticated programming. A barrier to providing such activities is the effort required to set up the programming environment. Testing is an important component to writing good software, but it is difficult to motivate students to write tests. In this paper we describe and evaluate CodeWrite, a web-based tool that provides drill and practice support for Java programming, and for which testing plays a central role in its use. We describe how we have used CodeWrite in a CS1 course, and demonstrate its effectiveness in providing good coverage of the language features presented in the course.

Categories and Subject Descriptors
K.3.2 [Computers and Education]: Computer and Information Science Education—computer science education

General Terms
Design, Human Factors

Keywords
CodeWrite, contributing student pedagogy, assessment, online, student-generated content, constructive evaluation

1. INTRODUCTION
One innovative approach for encouraging students to practice programming and to focus on testing is to challenge them to develop a shared repository of short programming exercises. Students develop associated test cases for the exercises they author, and receive immediate feedback on their own code when solving exercises created by their peers. This learning activity is supported by CodeWrite, a web-based tool. We report here on our experience using CodeWrite in a Java-based CS1 course.

Learning to program, as with any skill, requires practice. Syntax must be mastered, principles need to be understood, and techniques for problem solving need to be developed through regular effort. Programming is a practical subject and consequently much of the learning takes place at a computer while students tackle assignments or lab exercises, both of which are common forms of assessment in introductory programming courses [7]. However, due to their scope, assignments often require background reading or related contexts to be understood before a student can make progress. Similarly, lab exercises may require students to download resource files or configure their programming environment before code can be written. While writing complete programs is critical for developing basic design skills, it can be complemented by the repetitive practice of fundamentals. Drill and practice activities have a low cognitive load, but enable students to master the skills prerequisite for solving more sophisticated programming assignments [13].

CodeWrite presents students with a large repository of short programming exercises with which they can practice. They can begin writing code immediately, and can use the tool from any computer with a browser and network connection.

An important component of writing good software is the ability to design and execute effective tests. The teaching of testing in academia has begun to mature [3], however there is still debate as to how it should be taught and when it should be introduced in a curriculum [16, 17]. A barrier to integrating testing into the first year programming course is that beginners may struggle to appreciate the purpose of testing — it needs to be incentivized. A study by Barrioscanal et al. into whether students enjoy writing test cases reported that, when given the choice, only 10% of the class developed tests for their assignments [2]. Testing is central to the way CodeWrite works as exercise authors begin by defining the set of tests that characterize their exercise. Only once these test cases have been developed, and the exercise author has contributed an implementation that satisfies the tests, is the exercise available to other students. The incentive for developing test cases is clear as they are necessary to automatically verify and provide feedback on code submitted by the class.

In CodeWrite, students are responsible for developing the exercises which are shared with their classmates, a pedagogical approach called constructive evaluation [11]. CodeWrite shares some of the benefits offered by peer review, a learning activity that has been used successfully in a range of contexts over many years [1, 10]. One specific advantage of peer review in programming courses is that a student is exposed to a variety of solutions to a problem they have also solved. Chinn suggests this may enable students to recognize there...
are multiple correct solutions to a given problem [8], thus advancing them in Perry’s scheme of intellectual development [15]. A comparison of solutions can reveal alternative techniques for writing code that may be more succinct and easier to understand than their own. Once students successfully answer an exercise in CodeWrite, they are given access to the solutions for that exercise contributed by their peers. Students can then take part in a light-weight peer review process where they rate the quality of the exercise, provide feedback to the exercise author in the form of comments, and endorse solutions contributed by their peers.

In this paper, we describe the CodeWrite tool and report on its use in a CSI course at the University of Auckland. Our primary focus is the language feature coverage of the exercises in the CodeWrite repository, and the variety and feature coverage of the submitted solutions.

2. RELATED WORK

Palma [13] emphasises the importance of building confidence using drill and practice, particularly for female students, before progressing on to writing larger sections of code. Plentiful opportunities for drill and practice are provided by Parlante’s CodingBat tool (previously named JavaBat) [14]. CodingBat is a web-based system that includes a database of exercises expressed as short descriptions of the specification for a single method. Students using CodingBat need only write the method body, and they get immediate feedback in the form of passed or failed test cases that are highlighted green or red. In addition to the large bank of available exercises, course instructors may author their own exercises. The purpose of CodingBat is to build student skill and confidence by presenting them with many practice problems, and unlike larger assignments that may include a realistic context and backstory [19], it is not intended to motivate the bigger picture of Computer Science. Confidence building is also the motivation behind Edmonson’s use of proglets [5]. Well designed proglets allow students to focus on a single aspect of a programming problem, highlighting fundamentals and helping students build confidence before being asked to produce “assignment quality” code.

Incorporating software testing into the curriculum continues to be a challenge in academia [3]. Test driven development (TDD) has been suggested as a process that offers promise in the classroom, and a growing body of evidence to support its use sits alongside a number of innovative tools [6, 18] to realize its benefits. Test driven learning (TDL) is a somewhat more general approach that proposes introducing testing early and has been successfully employed in first-year programming classes [9, 4]. The goal of TDL is to encourage students to think about the kinds of errors that may occur in their code, and to develop tests to ensure such errors are not present.

The CodeWrite tool presented in this paper emphasises testing by requiring exercise authors to define test cases prior to implementation, presents opportunities for peer feedback and comparison, and provides students with many simple confidence-building exercises for drill and practice.

3. CODEWRITE

When developing a new exercise in CodeWrite, a student provides both a brief and a detailed description of the purpose of a method. The brief description is displayed as a preview to students selecting an exercise to answer from the repository (figure 1). The detailed description is shown once an exercise is selected to be answered, and should include all the information necessary to complete the method definition. Ideally, this description should include several examples of inputs and output for the method. The exercise author then defines the method signature. The return type of the method is selected from a drop-down list. The method name should ideally be descriptive of the task and up to 5 parameters can be defined, again with their types being selected from a drop-down list. The types are restricted to a subset of the most commonly used primitive types, arrays of these types, Strings and String arrays.

Next, the test cases must be defined. CodeWrite displays several rows of empty input and output text boxes, derived from the method signature. Each row corresponds to one test case, and the student types the input values and the output value directly into the empty boxes on a row to define one test case. Initially three rows are shown, but the form is expandable and the student can add up to 10 test cases if they wish.

Once the form is submitted, the exercise is in a pending state. Before the exercise is visible to other students in the class, the author must provide an implementation of the method that passes all of the test cases they have defined. If any issues are detected at this stage, the student can edit the exercise. Once an implementation that passes all provided test cases has been provided, the exercise becomes available to other students in the class.

A student practicing writing code with CodeWrite can sort the exercises chronologically or by the quality ratings assigned to the exercises by their peers (figure 1). Once an individual exercise is selected, the detailed description of the exercise is displayed, and a text area is shown beneath...
4. METHODOLOGY

4.1 Context of use

The COMPSCI 101 course at the University of Auckland is a 12 week CS1 course in Java taken primarily by students intending to major in Computer Science. Students attend three one-hour lectures and one two-hour lab session per week. CodeWrite was introduced to the class at the end of Week 4 by way of a 10 minute demonstration in the lecture. Attendance in lectures is not compulsory (typically attendance rates are around 50% of enrolled students) so a short document was also prepared that included several screenshots and explained the basic process of authoring an exercise with test cases and submitting a solution to an existing exercise. The topic of “methods and parameter passing” was introduced during the lectures of Week 3, and students were given relevant exercises during the lab sessions of Week 4. Therefore when the CodeWrite activity was introduced to the class, students already had some practical experience writing methods.

Students were required to contribute one exercise, and to successfully answer 10 exercises (all test cases passing) to meet the participation requirements of the activity. They were not restricted regarding the content of the exercise, other than being asked to target only language features that had been covered in class. A categorization of these language features is given in Table 1.

Completing the participation requirements contributed 1% towards the final grade of a student. Although they were given 4 weeks to complete this activity, only 2 of these weeks were during the semester as there was a 2-week mid-semester break during this time. The final deadline for the activity, the end of Week 6, was the day of the mid-term test for the course that contributed 15% of the final grade.
4.2 Research questions

Our primary focus in this paper is to investigate how well the CodeWrite activity supports students in practicing the language features taught in the course. Specifically, our investigation considers the 8 language features in Table 1 and the extent to which these have been used by students when solving CodeWrite exercises.

RQ1: Do the exercises in the CodeWrite repository encourage the use of all of the features of Java the students have been taught?

This question addresses the concern that students may have built a repository of exercises that does not require students to practice the complete range of language features. We say a question encourages the use of a feature if there is at least one successful submission for the question that makes use of the feature. For each feature, we calculate the total number of questions in the repository in which it is used.

RQ2: When solving exercises from CodeWrite, do students use the full range of features they have been taught?

This question addresses the concern that even if the repository does offer good coverage of the language features, many students may not practice with the full range. We say a student uses a feature if there is at least one successful submission by the student that makes use of the feature. For each feature, we calculate the total number of students in the class who have practiced using it. We also, for each student, determine the number of features used by that student.

RQ3: Are students exposed to a variety of solutions for a given exercise?

Once a successful submission for an exercise has been made, the student is able to see how their peers have answered the same exercise. The greater the variety of these solutions, the greater the opportunity for comparison and for making judgements about their quality. For this analysis, we considered solutions to the ten exercises with the greatest number of successful solutions. We measured variety by looking at the length and complexity of successful solutions to each of these exercises.

5. RESULTS

In this section we summarize the overall use of the system and report on our analysis of the code students have submitted while using CodeWrite.

In the results given below, we removed from our analysis any student that did not sit the final exam. Also, during data collection, there was a short period of time where one of the CodeWrite modules stopped functioning correctly. Submissions made by students during this time were recorded, however students did not receive feedback on their submissions. We have removed from our analysis all student submissions where feedback was not correctly received (amounting to 3.9% of total submissions). Table 2 summarizes the number of solutions that were submitted by students across 280 available exercises.

The majority of the students in the course logged in to CodeWrite and submitted at least one successful solution, however, relatively few students met the requirements of the assessment task (i.e. authored 1 exercise and submitted successful solutions to 10 exercises). We speculate perhaps that 1% of the final grade was insufficient motivation for a majority of the students. A summary of the number of students using CodeWrite is given in table 3.

5.1 Do the exercises in the repository encourage use of all the syntax taught?

If a student has used a language feature in a successful solution to an exercise, then we consider that the exercise has encouraged the student to practice with that feature. Since there are many possible solutions for a given exercise, we categorized the exercise as pertaining to a given topic if there was any occurrence of the corresponding Java language feature in any of the successful submissions to that exercise. The columns labelled Questions in table 4 summarize the number and percentage of questions that have had at least one successful solution containing a language feature corresponding to each topic.

From the table we can see that 88% of the exercises have at least one solution that uses the assignment operator. Although this is unsurprising given that assignment is a simple and common language feature, we see that 52% of exercises evoke the use of a conditional in at least one solution, one quarter of the exercises have a solution that uses a loop, and 17% of the exercises have at least one successful solution that uses an array. We believe that this shows good evidence that the exercises provide ample opportunities for students to use

### Table 2: Summary of CodeWrite repository contents

<table>
<thead>
<tr>
<th>Description</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number of passing submissions:</td>
<td>3939</td>
</tr>
<tr>
<td>Total number of failing submissions:</td>
<td>4708</td>
</tr>
<tr>
<td>Total number of non-compiling submissions:</td>
<td>11604</td>
</tr>
<tr>
<td>Total number of submissions:</td>
<td>20251</td>
</tr>
</tbody>
</table>

### Table 3: Summary of the students using CodeWrite

<table>
<thead>
<tr>
<th>Description</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Students with exam marks</td>
<td>437</td>
</tr>
<tr>
<td>... and submitted something to CodeWrite</td>
<td>330</td>
</tr>
<tr>
<td>... and a successful CodeWrite solution</td>
<td>299</td>
</tr>
<tr>
<td>... and met the requirements</td>
<td>138</td>
</tr>
<tr>
<td>... and answered more than required</td>
<td>102</td>
</tr>
</tbody>
</table>

### Table 4: Number and percentage of students that practiced each topic

<table>
<thead>
<tr>
<th>Topic</th>
<th>Questions</th>
<th>%</th>
<th>Students</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assignment</td>
<td>247</td>
<td>88</td>
<td>286</td>
<td>95</td>
</tr>
<tr>
<td>Arithmetic</td>
<td>215</td>
<td>76</td>
<td>283</td>
<td>94</td>
</tr>
<tr>
<td>API use</td>
<td>181</td>
<td>64</td>
<td>282</td>
<td>94</td>
</tr>
<tr>
<td>Relational</td>
<td>161</td>
<td>57</td>
<td>265</td>
<td>88</td>
</tr>
<tr>
<td>Logicals</td>
<td>69</td>
<td>24</td>
<td>206</td>
<td>68</td>
</tr>
<tr>
<td>Conditions</td>
<td>147</td>
<td>52</td>
<td>257</td>
<td>85</td>
</tr>
<tr>
<td>Loops</td>
<td>72</td>
<td>25</td>
<td>159</td>
<td>53</td>
</tr>
<tr>
<td>Arrays</td>
<td>49</td>
<td>17</td>
<td>72</td>
<td>24</td>
</tr>
</tbody>
</table>

474
5.2 Do students practice the full range of topics taught when solving exercises?

The columns labelled Students in Table 4 show the number and percentage of students that have practiced a given topic. We can see that the majority of students used language features corresponding to most topics in the course, although “harder” topics have been used by fewer students. Slightly more than half (53%) of the students submitted at least one solution that used a loop, and almost one quarter (24%) wrote code that used an array.

Figure 5 shows how many topics an individual student covered over all successful submissions made by that student. Each cluster shows the proportion of students whose submissions covered a particular number of topics. The left bar in each cluster (light grey) considers all 299 students who made at least one successful submission, whereas the right bar (dark grey) considers only the 138 students who met the course requirements for use of CodeWrite (i.e. contributed at least 1 and successfully answered at least 10 exercises). So, 28.4% of all students covered 7 of the 8 topics, whereas 35.5% of those who met the requirements covered 7 of the 8 topics. 87.7% of students who met the requirements of the CodeWrite activity practiced with at least 6 of the 8 language features.

Figure 6 shows the breakdown of the Students column in Table 4 by quartile. Each cluster corresponds to one of the topics, with the first bar being for the top quartile and the 4th bar for the bottom quartile. So for the “Arithmetic” topic, 99% of the students in the top quartile submitted a solution that passed all tests and used some form of arithmetic, whereas the results for the other quartiles were 97%, 92%, and 85% respectively. This pattern of decreasing use of features across quartiles reflects the decreasing participation rates by each quartile.

5.3 Are students exposed to a variety of solutions?

To answer this question, we considered the ten exercises with the greatest number of successful solutions. For each, we examined the variability of the non-comment, non-blank number of lines of code (loc) and the Cyclomatic Complexity Number (CCN) [12] of their successful solutions. These solutions are visible to a student once an exercise is successfully answered.

Table 5 summarizes these two metrics for each of the ten most popular exercises. From this table, we can see that there is substantial variation in the number of lines of code for most questions, and that the CCN (i.e. the control flow) shows substantial variation for most exercises. However, some exercises, such as getPower and gstRate show little variation in CCN.

We can conclude that students using the CodeWrite system are likely to encounter a variety of different successful solutions to the same problems.

Table 5: Variety of answers submitted to the ten most popular exercises

<table>
<thead>
<tr>
<th>Question</th>
<th>N</th>
<th>min</th>
<th>max</th>
<th>avg</th>
<th>ccn min</th>
<th>ccn max</th>
<th>ccn avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxOfThree</td>
<td>244</td>
<td>1</td>
<td>16</td>
<td>3.20</td>
<td>1</td>
<td>7</td>
<td>1.44</td>
</tr>
<tr>
<td>willEatFood</td>
<td>192</td>
<td>1</td>
<td>14</td>
<td>5.35</td>
<td>1</td>
<td>9</td>
<td>3.06</td>
</tr>
<tr>
<td>isOddSum</td>
<td>190</td>
<td>1</td>
<td>11</td>
<td>5.23</td>
<td>1</td>
<td>6</td>
<td>1.86</td>
</tr>
<tr>
<td>minOfSix</td>
<td>182</td>
<td>1</td>
<td>17</td>
<td>4.22</td>
<td>1</td>
<td>17</td>
<td>1.35</td>
</tr>
<tr>
<td>getInitials</td>
<td>116</td>
<td>1</td>
<td>14</td>
<td>3.93</td>
<td>1</td>
<td>5</td>
<td>1.05</td>
</tr>
<tr>
<td>getBackwStr</td>
<td>105</td>
<td>1</td>
<td>11</td>
<td>5.87</td>
<td>1</td>
<td>4</td>
<td>2.00</td>
</tr>
<tr>
<td>diceRolled</td>
<td>102</td>
<td>3</td>
<td>18</td>
<td>8.74</td>
<td>3</td>
<td>15</td>
<td>7.07</td>
</tr>
<tr>
<td>gstRate</td>
<td>76</td>
<td>1</td>
<td>6</td>
<td>2.05</td>
<td>1</td>
<td>1</td>
<td>1.00</td>
</tr>
<tr>
<td>interlveStr</td>
<td>85</td>
<td>5</td>
<td>31</td>
<td>13.04</td>
<td>3</td>
<td>15</td>
<td>4.45</td>
</tr>
<tr>
<td>getPower</td>
<td>66</td>
<td>1</td>
<td>5</td>
<td>1.61</td>
<td>1</td>
<td>2</td>
<td>1.02</td>
</tr>
</tbody>
</table>

5.4 Gender differences

Although it wasn’t one of our research questions, we investigated the degree to which male and female students engaged with CodeWrite. Of the students who sat the exam (i.e. completed the course), 22.2% were female. We noted that 22.7% of the students who submitted at least one solution were female, consistent with the percentage of female students in the course.

6. DISCUSSION

As noted above, the solutions varied in length and complexity, indicating different solution approaches from quite naive to very sophisticated. For example, one exercise, named getLettersNotUsed, required students to calculate all of the letters not used in a particular String, and return these unused letters. One of the less sophisticated solutions to this
problem involved a sequence of 26 conditional statements, each of which included a test for both upper and lower case characters. An extract of this is shown in Figure 7. The student who developed this solution, would immediately have access to other solutions, including a more elegant example that first converts the input to lower case and utilizes character codes within a loop.

Upon submitting a solution to an exercise, students are shown all test cases for the exercise. The intent is to provide guidance, and to encourage students to complete the exercise so that they may move on to another. We observed a very small number of cases where students created a passing solution to an exercise simply by looking at the test cases and writing a conditional statement that returned the expected value given the appropriate input. Tools such as CodingBat, where the exercises are prepared by an instructor, prevent such reverse engineering by including private test cases that are not seen by the student. One difficulty with this approach in CodeWrite is that students could either accidentally or maliciously contribute a private test that makes the exercise unsolvable without knowing what the test is. One possible solution, to prevent reverse engineering of solutions, would be to hide a single test case chosen at random for each student.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we have described the CodeWrite tool and demonstrated its effectiveness in providing good coverage of the language features presented in a CS1 course. Examination of a repository of 280 exercises indicated that all course topics were covered (with 17% of the exercises encouraging use of arrays, and up to 88% encouraging use of variable assignment). We found that students practicing with CodeWrite would develop code covering many of the topics taught in the course (nearly 90% of those students answering at least 10 exercises practiced with at least 6 of the 8 topics). Exercises in CodeWrite also attracted a variety of solutions, all of which are viewable by students upon submission of their own successful solution.

Future work will investigate whether use of CodeWrite as a drill and practice tool has an effect on the confidence and performance of students tackling larger programming assignments and formal examinations.

8. REFERENCES


Figure 7: Excerpt of successful solution to getLettersNotUsed (reformatted for space).