What’s Wrong With My Code (WWWMC)

Mr. Adam Thomas Koehler, University of California - Riverside

Current Computer Science Ph.D. student at the University of California, Riverside with a research emphasis in computer science education.

Prior to my Ph.D. studies, I attended Marquette University and received a Bachelor of Science degree in Computer Science and a Master’s of Science degree in Mathematics, Statistics and Computer Science with a concentration in computer science.
What's Wrong With My Code (WWWMC)

Adam Koehler
Deptartmen. of Computer Science and Engineering
University of California, Riverside

Abstract

Student-instructor interaction and passage of knowledge is often optimal in a one-on-one setting. Knowledge, including information about topic-specific misconceptions, is imparted through individualized interactions between the student and the instructor. Unfortunately, these interactions can be time consuming. Automated assessment and directed educational tools attempt to address the time concerns by presenting the student with immediate feedback, but can often be lacking in other ways.

Our paper presents a teaching instrument, What's Wrong With My Code, that allows us to capture these one-on-one interactions. We provide our design methods used to create the What's Wrong With My Code problem sets and analyze the results of studies which utilized the problem sets within a teaching environment.

Introduction

Growing class sizes as well as dwindling resources have led instructors to pursue different tactics, methods, and implementations in scalable and effective teaching habits. With these dwindling means, certain teaching methodologies have to be revamped to accommodate for resource and personnel constraints with rising per-class student enrollments. The apprenticeship method is one such teaching strategy that requires adaptation; within this method, a single student learns through direct instructor feedback across various examples and implementations through several one-on-one interactions. One-on-one interactions help facilitate a great teaching environment, and are often utilized to teach students about programming misconceptions and errors in an introductory programming course. The repetitive nature of a substantial portion of these interactions makes them a prime candidate for improving scalability through automation.

Automated assessment of programming exercises is often utilized to bridge the scalability gap. However, the open-ended nature of programming assignments can lead to (1) misguided automatic feedback, (2) a disconnection between an errant student solution and proper advice, (3) a complete lack of advice due to the student not understanding the presented question, or (4) knowledge gaps due to students never encountering a problem/feedback pairing. What's Wrong With My Code (WWWMC) attempts to emulate the apprenticeship environment and solve these problems by using a much more stringent and guided experience while still maintaining scalability.
Background

As class sizes grow, the instructor-student relationship must be supported by growing technologies that allow interaction outside the normal lecture environment. Over several years, various forms of automated assessment\textsuperscript{[10][16][17]} tools have also been created to either grade homework or tutor a student on a specific topic or both. In recent years a few of note include Web-CAT\textsuperscript{[7]}, Marmoset\textsuperscript{[15]}, and Codelab\textsuperscript{[2]}. One of the goals of automated assessment is to replace both grading and grading feedback with automated testing suites, thus reducing the workload of instructors with growing class sizes. As an example, the Codelab\textsuperscript{[2]} environment allows both automated assessment through grading of a multitude of exercises as well as feedback on an errant solution in an attempt to guide the student to a correct submission. However, automated assessment utilities can have drawbacks based on how general or how specific the tool implementation and use is. Some autograders trade specific tutoring and feedback away for ease of implementing an automated grading system for a larger set of problems. With this in mind, the feedback becomes the most difficult part of scaling automated assessment.

As resources continue to change in computer science education, many old techniques utilized by instructors to distribute knowledge have moved to an online environment. Systems such as Codelab\textsuperscript{[2]} have established online presence as tutoring software for computer science education. Additionally, researchers\textsuperscript{[6][17]} have shown that increasing the interaction level of the tools utilized within computer science education can improve students’ scores. For example, Edgcomb and Vahid\textsuperscript{[6]} showed that using interactive web content, such as animations, within an online textbook is more effective than simply migrating a book to a static online version.

Identifying pitfalls, ranging from those of a specific programming language to pitfalls of an entire computer science class, has also been a long sought out task of researchers\textsuperscript{[5][15]}. When building and analyzing the findings of Marmoset, Spacco et al. set up repositories for student code that allowed the researchers to analyze intermediate student programs and not just final submissions\textsuperscript{[15]}. Often these pitfalls can be categorized; in the work of Garner, Haden, and Robins twenty-seven categories were outlined from the error reports gathered within a Java based CS 1 course\textsuperscript{[8]}. As we will show in the WWWMC Tool Development section, many of the categories we have chosen for our question sets overlap the outlined categories of Garner et al.

Once misconceptions are known, incorporating pedagogy into courses is an active pursuit of not only computer science education\textsuperscript{[4][5][9][11][12]} but other educational fields\textsuperscript{[3][13]} as well. Pair debugging\textsuperscript{[12]} by Murphy et al. shows how to incorporate debugging into the commonly utilized pedagogical technique of pair programming. Simon et al. developed several videos to help CS 1 students when debugging programs\textsuperscript{[14]}. Each of these pedagogical techniques, have had a modicum of success, but a few are only feasible in a smaller classroom and some are not easily ported to different programming languages.
WWWMC Tool Development

A problem description, problematic code, potential solutions, and instructor feedback are usually covered throughout a series of interactions between the instructor and the student. Additionally, multiple students often ask quite similar questions. Within the What's Wrong With My Code questions, these multi-part and potentially multi-day teaching moments have been streamlined into a single WWWMC question. An individual WWWMC question allows all students to encounter the problem, solve the problem, and receive feedback at the same time.

![Example What's Wrong With My Code problem](image)

The WWWMC problems were designed to have four distinct parts, as seen in Figure 1. The left column initially contains the problem or question, as well as the problematic code. The right column contains potential answers for the student to choose from. When an answer is selected, the left column is updated with a reason for why the choice is correct or incorrect. Also, once an answer is chosen, textual coloring and answer highlighting is utilized to further emphasize whether the selected answer is correct or incorrect. Each of the problems were cultivated from the CS 1 course which utilizes the C++ programming language.

The initial part of the WWWMC question is a student question paired with code derived from a student submission. We gathered student programming questions across a two year period from course forums, program submissions, and office hour inquiries. From these collected questions, we isolated several pervasive and recurring issues to build our question sets around. The question for each WWWMC problem covers a distinct issue that many students encounter when learning to program for the first time. The set of created problems utilizes C++ syntax, but the concepts covered are portable to other introductory programming languages, such as Java. Over fifty problems were developed and categorized into some of the major CS 1 teaching topics as shown in Table 1. Many of the categories and errors overlap with the previous categorization by Garner, Haden, and Robin[8]. The first aspect of a "What's Wrong" interaction is the student presenting the problem to an instructor; with this in mind in a WWWMC exercise, the problem description
and code are always presented from the student perspective. The problem description may have been edited for spelling, grammar, or conciseness, but great emphasis was placed on posing the question as a student would. The code portion of the question was trimmed to exclude any unnecessary pieces of code to help focus the question, but we maintained the full program aspect by presenting a piece of code that could be copied and pasted into an editor, compiled, and executed. Lastly, we added highlighting to the code to further focus the student.

| 1. Style          | 4. Randomness | 7. Loops |

Table 1: What's Wrong With My Code Question Categories

The categories shown in Table 1 contain several distinct errors from both runtime and compile time error groupings. For our purposes, when referring to runtime we mean any error that occurs during program execution such as logical bugs or thrown exceptions. Also, we have several style questions to help build a basis of proper style for a novice programmer. For example, in Figure 1 we see a compile time error being presented from the input and output problems; this question stems from the fact that many students attempt to combine input and output statements when first learning. In Figure 2, we see that the code has a runtime error due to the use of variables prior to assigning values to those variables.

![Figure 2: What's Wrong With My Code problem from the Basic Variables and Math category](image)

If we delve into the String Member Functions category, we see more examples of the mix of runtime and compile time errors. These include syntax errors, such as leaving off parentheses when invoking a function (shown in Figure 3) and runtime errors, such as the out of range...
exception (shown in Figure 4). A combination of compile time and runtime errors allows students to learn about multiple types of errors, while gaining knowledge of how to fix specific errors.

As shown in Figures 1, 2, 3, and 4, each question has several potential solutions. Each solution is stated from the student perspective similar to the problem description. As with the questions, many of the solutions were gathered from correct and incorrect student answers to other student's inquiries about incorrect code. Additionally, some of the solutions were derived from correct and incorrect implementations of student programs. Most solutions are a simple description of what is wrong with the code. Occasionally, a code snippet (as seen in Figure 2) is required to aide the description of what is wrong. The potential solutions are presented in a multiple choice layout. Using multiple choice allows the student to easily navigate through all the explanations by selecting the various solutions. An additional caveat of student written or altered code is that the student may never encounter an opportunity to learn about the error, but our question setup of providing errant code with alongside multiple choice options explicitly exposes the student to the error forcing each student to investigate.
Each potential solution gives feedback. When the student selects a potential solution, this feedback and explanation is shown. The feedback or explanation is always written from the instructor's perspective; examples of feedback can be seen in Figures 1 and 4. The explanation describes why the solution is correct or incorrect as if an instructor is talking to a beginning programmer. Additionally, the explanation utilizes a green checkmark or red 'x' as well as colored text to clearly denote whether a solution is correct or incorrect.

Lastly, the explanations attempt to avoid using too much computer science terminology. This avoidance is a direct counter measure to help simplify the explanation and sidestep confusing the student with unfamiliar words or programming terms. The goal of the explanation is to provide feedback on the potential solution and why the solution is correct or incorrect, not to teach computer science terminology and definitions. The feedback provides immediate justification to the student on why an answer should or should not be selected, rather than simply marking a question right or wrong.

**Study Implementation**

An initial investigative study was conducted over two quarters, Spring and Fall. Each study had a randomly assigned lesson chosen from two lessons. The first lesson option for all studies was the What's Wrong With My Code teaching instrument. The alternate lesson in the Spring study had the student complete several exercises with the Codelab instructional programming environment. The Fall study utilized a home grown automated assessment system to replace Codelab, and the participants also had to read a brief article about programming. The exercise used in the Fall study was a combination of several Codelab exercises but only provided correct or incorrect marking, no guided feedback. Codelab was unavailable for use in studies after the Spring quarter, because the tool was no longer used by the course and thus students did not have a subscription. In each study, the alternate lesson was chosen due to the fact that the lesson, in its entirety, attempted to teach similar items to the WWWMC questions.

As a whole, each study contained five parts, presented through a single web page, one part at a time. Each part was at least partially integrated with a Google form to allow data and timestamp collection at the completion of the section. The five parts of the study were: background survey, pretest, lesson or instruction, post-test, and follow-up survey. At the beginning of the course, the students were provided a unique four digit ID. The ID was utilized within the study to anonymize the results, provide credit to participants by passing only the IDs back to the course grader, and maintain continuity in data collection across the multiple parts of the study.

1. Write an expression that calculates the floating-point value of the fraction 1213 / 57101.
2. Given the code above [containing declarations and initializations], write the code to store the floating-point average of x, y, and z in the variable avg.
3. [Given multiple literal values] Which of the following is a character literal?

Table 2: Example written response (1&2) and multiple choice (3) test questions
The pretest and post-test were the same series of ten questions, comprised of three multiple choice questions and seven free response essay questions. Two example written response questions are provided in Table 2. Each of the seven free response questions required the student to write no more than a few lines of code. Item number 3 of Table 2 is an example multiple choice question. All questions within both the pretest and post-test were optional, and the participant was informed that responses could be left blank. Additionally, the participants were informed that credit was awarded for participation in the study, and not based on the score of either test.

To avoid grading bias toward a specific study, pretest over post-test, or toward a specific lesson, the student answers were all combined together regardless of test, lesson, or quarter. After merging the test responses, the collected student answers were randomized and then graded. To establish consistency in grading, a standard rubric was established for each written question to allow partial credit for the question between a score of 0 (no credit) and 1 (full credit).

**Study Participation Breakdown**

All participants in the study were enrolled in the Introduction to Computer Science course at the University of California, Riverside. This is the first course taken by all computer science majors. However, since the study was performed in two different quarters, there were many more computer science majors during the on-track Fall quarter. Even with the increase in computer science majors during the Fall study, the majority of participants did not have prior programming experience (66% of the 333 participants).

Even though participants were offered credit within the quarter for completing the study, not everyone chose to complete the study properly; some completing the background section and skipping all other sections or something similar. Additionally, some students in the course opted to skip the entire study.

The Spring study had 201 total participants, seven of which were computer science majors. The class breakdown of participants was 23% Freshmen, 34% Sophomores, 20% Juniors, and 19% Seniors with 4% being either non-matriculated or outside the normal classifications. The Fall study had 333 total participants, 80 of which were listed as computer science majors. The yearly distributions were 53% Freshmen, 17% Sophomores, 14% Juniors, 9% Seniors, and 7% other.

After the study was concluded, three features were used as exclusion criteria: (1) the student must complete all parts of the study, (2) the student must complete the pretest, lesson, and post-test within eighty minutes, and (3) the student had no prior programming experience before the course. The first criteria simply shows that the student was engaged in the study enough to complete the five required steps. The second criteria stems from the fact that the participants were informed that the study should be completed in a single sitting. An eighty minute allotment allowed twenty minutes each for the pretest and post-test, along with forty minutes for the lesson. Over 90% of the students that completed the study were able to do so in under eighty minutes. Of the remaining students, many were over several hours for completion time which indicated the student did not complete the study in one sitting. Lastly, due to the basic concepts
covered in the early quarter studies, only participants with no prior programming experience were considered as these students were new to programming as a whole.

With exclusion criteria 1 and 2, the Spring study had 128 eligible participants. After applying all three exclusion criteria, 91 participants remain. With the Fall study, 267 participants remain after applying exclusion criteria 1 and 2, and 172 participants remain after applying all three exclusion criteria.

Results and Analysis

The results of the study were analyzed for three different scores: the average pretest score, the average post-test score, and the performance improvement from pretest to post-test in the Spring and Fall studies. Additionally, during the Spring quarter participant engagement was evaluated during the follow-up survey using a six point Likert scale question with no neutral option; the neutral option was eliminated to force students to choose a side.

<table>
<thead>
<tr>
<th>Participants (N)</th>
<th>128</th>
<th>91</th>
<th>34</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exclusion Criteria</td>
<td>1 &amp; 2</td>
<td>1, 2, &amp; 3</td>
<td>1, 2, &amp; 3</td>
<td>1, 2, &amp; 3</td>
</tr>
<tr>
<td>Engagement Level</td>
<td>No Restriction</td>
<td>No Restriction</td>
<td>&gt;= Slightly Agree</td>
<td>&gt;= Agree</td>
</tr>
<tr>
<td>WWWMC Pretest</td>
<td>3.87</td>
<td>3.39</td>
<td>3.51</td>
<td>3.51</td>
</tr>
<tr>
<td>Codelab Pretest</td>
<td>4.60</td>
<td>4.68</td>
<td>4.38</td>
<td>4.74</td>
</tr>
<tr>
<td>WWWMC Post-test</td>
<td>5.73</td>
<td>5.12</td>
<td>5.41</td>
<td>5.42</td>
</tr>
<tr>
<td>Codelab Post-test</td>
<td>5.75</td>
<td>5.72</td>
<td>5.43</td>
<td>5.71</td>
</tr>
<tr>
<td>WWWMC Change</td>
<td>1.86</td>
<td>1.73</td>
<td>1.89</td>
<td>1.91</td>
</tr>
<tr>
<td>Codelab Change</td>
<td>1.15</td>
<td>1.03</td>
<td>1.05</td>
<td>0.97</td>
</tr>
</tbody>
</table>

Table 3: Scores for Spring study, out of 10 points

As shown in Table 3, both the alternate lesson and the WWWMC lesson showed improvements to the student test score in the Spring study. Students taking the WWWMC lesson posted a statistically better average improvement of 1.73 points versus students taking the alternate lesson who posted an average improvement of 1.03 points (p-value 0.033). The 1.73 points corresponds to a 48% improvement on the pretest score, and an actual grade percentage increase of 17.3%. When examining the post-test for differences to determine if the students achieved a different level of knowledge, no significant difference could be shown between the post-test averages for the two groupings despite the WWWMC average score being slightly lower than the Codelab grouping (p-value 0.162).

In the Spring study we also evaluated whether students who were more engaged with each of the lessons benefited more. We first attempted to determine whether the group of students for each lesson had different levels of engagement; when comparing engagement level averages between groups no significant difference could be shown (p-value 0.47). With that in mind, we looked at the scores of each lesson as we made greater restrictions on engagement level.
When we restrict the engagement to only participants who marked that they at least slightly agree that they were engaged in the lesson, the WWWMC had an average score increase of 1.89, an average pretest score of 3.51, and an average post-test score of 5.41. The Codelab lesson has an average improved score of 1.05, with averages scores of 4.38 and 5.43 for the pretest and post-test respectively. When we compare the scores for each lesson across engagement levels, neither lesson can show a significant difference. This may be due to the decrease in population size, as only 34 students meeting the additional restriction.

When we go one step further and only look at individuals that replied "agree" or "strongly agree" to the engagement question, the scores improve even more. The WWWMC lesson has an improvement of 1.91 points with average scores of 3.51 and 5.42 for the pretest and post-test. However, the Codelab scores maintain similar footing despite the increase in engagement, with a 0.97 points of score improvement and pretest and post-test scores of 4.74 and 5.71. Yet, we cannot show statistically significant difference, again most likely due to the decrease in population size that meets the criteria of further increased engagement.

Despite the lack of statistically significant evidence, we can see the positive improvement in Table 3 for WWWMC when students are more engaged, but similar improvement is not shown for the alternate lesson. Even though pretest scores remain similar, the greater increase could mean that when students are engaged in the WWWMC lesson, they are benefiting more than when they are engaged in the alternate lesson since the improvement scores do not rise for the alternate lesson.

![Table 4: Scores for Fall study, out of 10 points](image)

The Fall study had positive results similar to the Spring study, as shown in Table 4. The WWWMC group showed significant improvement from pretest to post-test (p-value less than 0.001). However, the alternate lesson takers did not show statistically significant improvement from pretest to post-test despite having a positive score change. The lack of improvement may have been due to the students performing a rather innocuous task that had minimal feedback to help them solve the problem they were tasked with, additionally having a single exercise to complete may have prevented the students from encountering all the potential errors before achieving a successful and complete submission.
Students taking the WWWMC lesson in the Fall posted a statistically better average improvement score of 1.16 points (a 24% improvement from the pretest) when using two of the three exclusion criteria. When all three exclusion criteria are applied, the improvement goes to 1.35 points (a 32% improvement from the pretest). Respectively those are grade percentage increases of 11.6% and 13.5%. Additionally, the students with the WWWMC lesson clearly benefited with significantly stronger post-test scores when compared to the alternate lesson (p-value less than 0.001).

A common sentiment among the follow-up feedback for the WWWMC lesson stated that students enjoy knowing why something is correct versus simply getting points. A common myth, often reinforced by correct/incorrect auto graders, is that if a student got the answer correct, then he or she clearly understands why. The explanations for correct and incorrect answers create teachable moments, reiterating knowledge the student may not have fully grasped. Of the students who took the WWWMC lesson, 78% of the students stated they explored both correct and incorrect answers to read the provided instructor feedback.

The results from Spring and Fall allowed us to conclude that the What's Wrong With My Code questions were a positive influence on the success of the students. The WWWMC lesson provided significant changes in scores, often increasing the student's score an entire letter grade. With this knowledge in mind, we moved to the third part of our implementation: integrating WWWMC into CS 1 for a full academic quarter.

Full Course Integration

Working with Zyante[1], the research team created a supplemental zyBook for the students of a Winter quarter. The zyBook was named PreLab to designate the time for completion of all activities in the supplemental text. The What's Wrong With My Code chapter had ten sections (one per week of the quarter): an introduction followed by nine sections containing WWWMC exercises. The introduction explained the purpose of the WWWMC questions and offered a few tips for effective use of the questions, based on prior feedback from students. All exercises in the supplemental textbook were required activities, similar to labs and programming assignments for the course.

To compare to previous studies, the same survey (from Spring and Fall) was utilized to study the effectiveness of the WWWMC problems that were now integrated into the weekly course materials. Since the WWWMC questions were fully integrated, there was no pretest given in Winter or Spring 2. The singular survey could be considered in a similar light to the previous post-tests since the survey was conducted after the relevant WWWMC questions should have been completed by the students. When analyzing the data, we applied exclusion criteria 1 & 3. Exclusion criteria 1 was modified to mean that all WWWMC exercises were completed prior to the survey. We could not apply exclusion criteria 2 since the exercises were not required to be completed in one sitting because the WWWMC exercises were fully integrated into the weekly required course work.
Table 5: Scores for Winter study, out of 10 points

<table>
<thead>
<tr>
<th></th>
<th>Winter</th>
<th>Spring 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participants (N)</td>
<td>118</td>
<td>147</td>
</tr>
<tr>
<td>Exclusion Criteria</td>
<td>1 &amp; 3</td>
<td>1 &amp; 3</td>
</tr>
<tr>
<td>WWWMC Test</td>
<td>6.75</td>
<td>6.46</td>
</tr>
</tbody>
</table>

Table 5 shows us the results of the Winter and Spring 2 studies. The Winter study had 118 participants, 4 of which were computer science majors. The Spring 2 study had 147 participants, and 14 students were computer science majors. The averages for Winter and Spring 2 studies were 6.75 and 6.46 points respectively, each out of 10 possible points. When we compared these results to the post-test results of Fall and Spring, the Winter and Spring 2 scores were significantly better than the Spring and the Fall scores with p-values less than 0.001 for Winter comparisons and p-values less than 0.009 for comparisons to Spring 2. These quiz results show that the complete integration of WWWMC into the course further benefited the students.

In addition to the quiz results, we collected all the errors that the students encountered during the multi-quarter study. For errors covered within What's Wrong With My Code we analyzed whether the percent of students that encountered the error was different during full quarter integration that the percent of students that encountered the error given no exposure to What's Wrong With My Code lessons during the quarters prior to full concept integration. We collected errors by redirecting g++ with a script that automatically filled in a Google form when an errant compilation occurred. The redirection script allowed us to gather errors for all students that used the online IDE for the course, when setting up the script an initial form submission occurred to show the instructors that online IDE workspace was set up. Students who did not set up the online IDE (either used their own setup or used a completely different IDE) were removed from the population before percentages for each error were calculated.

In Table 6 we show the percent of students that never encountered an error in a given quarter. The error search strings were the values utilized to search through all the errant submissions. In most cases this search string corresponds to a single error that was taught about within WWWMC exercises. However, a few strings such as operator<< will gather several errors together, in this case all errors relating directly to misuse of the operator. We chose to group these similar errors because WWWMC exercises covered several different potential all relating to a singular concept, such as the proper use of the operator.

As shown in Table 6, full integration does not win out in every comparison, but more often than not the comparison results are in favor of the quarters with full WWWMC integration. For example, the expected semicolon error row shows that Winter and Spring 2 have similar percentages of students that never encounter the error, but in Spring 1 the percent of students that did not encounter the error was significantly better than the fully integrated quarters. Similar results that favor one or both the partial integration quarters include errors for expecting primary expressions prior to the << operator and assignment into a read-only variable.
<table>
<thead>
<tr>
<th>Error Search String</th>
<th>Spring 1*</th>
<th>Fall*</th>
<th>Winter</th>
<th>Spring 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>expected ';' before '{'</td>
<td>93.69%</td>
<td>59.63%</td>
<td>76.00%</td>
<td>79.39%</td>
</tr>
<tr>
<td>no return statement in function returning non-void</td>
<td>47.75%</td>
<td>43.58%</td>
<td>99.11%</td>
<td>97.33%</td>
</tr>
<tr>
<td>expected primary-expression before '&lt;&lt;'</td>
<td>63.06%</td>
<td>44.95%</td>
<td>58.22%</td>
<td>60.69%</td>
</tr>
<tr>
<td>operator&lt;&lt;</td>
<td>56.76%</td>
<td>67.89%</td>
<td>85.78%</td>
<td>87.02%</td>
</tr>
<tr>
<td>operator&gt;&gt;</td>
<td>75.68%</td>
<td>91.74%</td>
<td>96.00%</td>
<td>96.56%</td>
</tr>
<tr>
<td>invalid use of member</td>
<td>79.28%</td>
<td>68.35%</td>
<td>83.11%</td>
<td>85.50%</td>
</tr>
<tr>
<td>void value not ignored</td>
<td>97.30%</td>
<td>91.28%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>cannot be used as a function</td>
<td>81.98%</td>
<td>69.27%</td>
<td>92.44%</td>
<td>91.98%</td>
</tr>
<tr>
<td>assignment of read-only variable</td>
<td>100.00%</td>
<td>95.41%</td>
<td>98.22%</td>
<td>99.24%</td>
</tr>
</tbody>
</table>

Table 6: Percent of students that never encountered an error per quarter.
*Spring 1 & Fall students were those that were not exposed to WWWMC lessons.

The fully integrated quarters were significantly better in several cases (p-values under 0.01); the rows in Table 6 with the following search strings showed significant improvements: no return statement, operator<<, operator>> (Spring 1), invalid use of member, void value not ignored, and cannot be used as a function. The positive results showing the percent of students that never encountered an error help further support our claim that What's Wrong With My Code exercises benefit the students.

**Future Work**

We hope to work with Zyante\(^1\) textbooks to create a free online textbook containing all the What’s Wrong With My Code problems, at present the questions are available upon request. This free zyBook will allow open access to the developed WWWMC problem sets. The developed questions will be sorted by the CS 1 concepts mentioned in the WWWMC Tool Development section. Additionally, we would like to translate the questions into other programming languages leading to an entire zyBook of WWWMC problems with several chapters, each chapter in a single programming language.

We would like to stem off a subset of WWWMC specifically for style. Style is often a "learn through experience" endeavor, and we believe that using WWWMC type questions can help eliminate some of the programming style frustrations encountered by novice programmers when the student is forced to adapt to the programming style of an instructor.

Lastly, we are investigating whether a more structured tutoring system can effectively be built using What's Wrong With My Code problems. We are unsure whether this would create further benefit for students, but it is certainly worth investigating given our positive results thus far.
Conclusion

The What's Wrong With My Code teaching instrument provided positive results to the students’ education. In all of the studies conducted, the students showed improvement from pretest to post-test when using the WWWMC lesson as a teaching instrument. As the tool was improved, the scores for the students using the WWWMC tool also improved. The improvement to student test scores for the WWWMC lesson participants was significantly greater than the improvement shown by students taking the alternate lesson in the Fall study with p-values less than 0.001. Additionally, we showed that the the percent of students to never encounter an error covered within WWWMC exercises was increased in the quarters with full integration of WWWMC exercises.

Ultimately, we believe our more guided approach with curated problems and solution-explanation pairings helps minimize some of the known drawbacks in automatically assessing open-ended programming problems. First, by providing curated problems with select answers, each problem provides concise instructor feedback. Second, the streamlined nature of presenting all the pieces of the problem in one exercise helps to reinforce the connection between all the pieces: the problem, the errant code, the solution, and the feedback. Third, to avoid students struggling to understand the question, each question is framed from the student perspective. If the student is still struggling, due to the nature of the implementation of WWWMC problems, the student has access to the code, solutions, and feedback to provide contextual support in understanding the problem. Lastly, every student that completes the exercise will have seen the problem/solution/feedback pairing, leaving no knowledge differences caused by free form student implementations that may never trigger a certain portion of feedback. The results shown with this paper suggest that students will benefit from continued integration of What's Wrong With My Code problems within an introductory programming course.

References


