Manual Analysis of Homework Coding Errors for Improved Teaching and Help

Mr. Nabeel Alzahrani, University of California, Riverside

Nabeel Alzahrani is a Computer Science Ph.D. student in the Department of Computer Science and Engineering at the University of California, Riverside. Nabeel's research interests include causes of student struggle, and debugging methodologies, in introductory computer programming courses.

Prof. Frank Vahid, University of California, Riverside

Frank Vahid is a Professor of Computer Science and Engineering at the Univ. of California, Riverside. His research interests include embedded systems design, and engineering education. He is a co-founder of zyBooks.com.

Dr. Alex Daniel Edgcomb, Zybooks

Alex Edgcomb is Sr. Software Engineer at zyBooks.com, a startup spun-off from UC Riverside that develops interactive, web-native learning materials for STEM courses. Alex is also a research specialist at UC Riverside, studying the efficacy of web-native content and digital education.

Manual Analysis of Homework Coding Errors for Improved Teaching and Help

Abstract

Previous research reports common student errors in introductory programming (CS1) classes. Knowing common errors enables us to improve teaching and content to train students to avoid those errors, and to provide an automated help system like providing a hint based on a particular auto-detected error. Finding and fixing some errors is part of learning, so our focus is specifically on errors that cause struggle, meaning excessive time or attempts. Struggle may lead to giving up, losing confidence, or cheating. For 89 online auto-graded C++ coding homework problems in our CS1 class of 100 students (mostly engineering/science majors), we first automatically determined the 12 problems with the highest struggle rates. Then, we spent about 100 hours manually examining incorrect student submissions to determine what errors caused struggle and the time spent on each error. Like previous work, we found many common general errors, like using = rather than ==. However, we also found problem-specific errors, like misusing a particular library function, leading to a first conclusion that a help system should allow teachers/authors to add problem-specific hints. Furthermore, we analyzed errors that caused the longest struggle, and found some uncommon "one-off" errors, leading to a second conclusion that a help system will not be able to detect all errors and thus might need automated recommending or alerting for human assistance (or other techniques).

1 Introduction

Issues that students face in introductory programming classes (CS 1) can cause stress and frustration among students, which can lead to attrition [1]. One issue is spending excessive time dealing with errors in programs for homework or lab. Thus, much previous research lists common programming errors in CS 1. McCauley's survey [2] from 1970 to 2008 yielded these common general errors: zero is excluded, output fragment, off-by-one, wrong constant, wrong formula, variable not declared, inappropriate use of a non-static variable, type mismatch, non-initialized variable, method call with wrong arguments, method name not found, infinite loops, misunderstanding of operator precedence, dangling else, code inside a loop that does not belong there, not using a compound statement when one is required, array index out of bounds, improper casting, invoking a void method when you want a result, failure to return a value, confusion regarding declaring parameters in methods and providing them in the call, a class declared abstract due to omission of function. Some researchers have created tools to help alert students of common errors. Hristova's Espresso tool [5] alerts students to common syntax and logical errors in Java, including improper casting, ignoring a method's return value, flow

reaching end of a non-void method, confusing parameters and arguments, return type mismatch, and more. Simon [6] created debugging videos to help Java learners debug common errors, including == versus .equals, empty while loop body, infinite loop, incorrect iterator update, missing input statement, boolean expression, confusing || and &&, array off by one leads to incorrect answer, linear search of an array, updating found incorrectly, string methods / ignoring return values, instance variable masking, pass by value misunderstanding, and unable to return value through primitive parameter. Many other works list common learner errors [7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21].

Some researchers include a focus on common general errors that students take much time to fix. Altadmri [3] automatically analyzed 37 million compilations from 250,000 students learning Java using BlueJ to find runtime errors, considering frequency and time-to-fix, yielding a list of time-consuming errors like confusing &&/|| and &/|, using == instead of .equals for strings, ignoring a method's return value, putting a semicolon after if, and dozens more. Median time-to-fix for the above errors were 17 min to 7 min. Ettles [4] analyzed 51,000 submissions from 809 students solving 10 problems in C using CodeWrite, yielding time-consuming errors of: accessing an invalid array element, off-by-one errors, boundary errors, and more, with median times ranging from 20 min to 8 min.

In this paper, like previous works, we analyzed common errors in CS1, in this case for 89 auto-graded C++ coding homework problems provided in widely-used zyBooks. Our class used the C++ zyBook, but similar homework problems exist in the C, Java, and Python zyBooks. Our analysis differs from previous work in that we performed extensive manual analysis (since programs often had multiple errors, we determined, as best we could, the time spent on each error) and focused almost exclusively on struggle-causing errors. *While we found many general common errors similar to previous works, we also found many common problem-specific errors. This means if teaching or help systems only focus on general common errors, then many students will still struggle. In fact, via further analysis, we found that "one-off" errors also cause struggle -- errors that are not common but that a particular student got stuck on. This means we need to go even further to detect such situations and provide customized hel*p. We provided zyBooks with our results to consider for incorporating into their contents.

2 Auto-graded homework problems

We use a zyBook, an online interactive textbook content for learning programming used by about 500 universities and 200,000 students in 2018 per the company's information [22]. A zyBook has integrated coding homework problems known as challenge activities (or CAs). Each problem has students finishing a small program by writing a few lines of code, like a for loop as shown in Fig. 1. Clicking Run compiles and executes the program on a server, tests using various

test cases, and shows results. Students can repeat any number of times until passing all test cases. Each problem is designed to take about 3-5 minutes to solve.

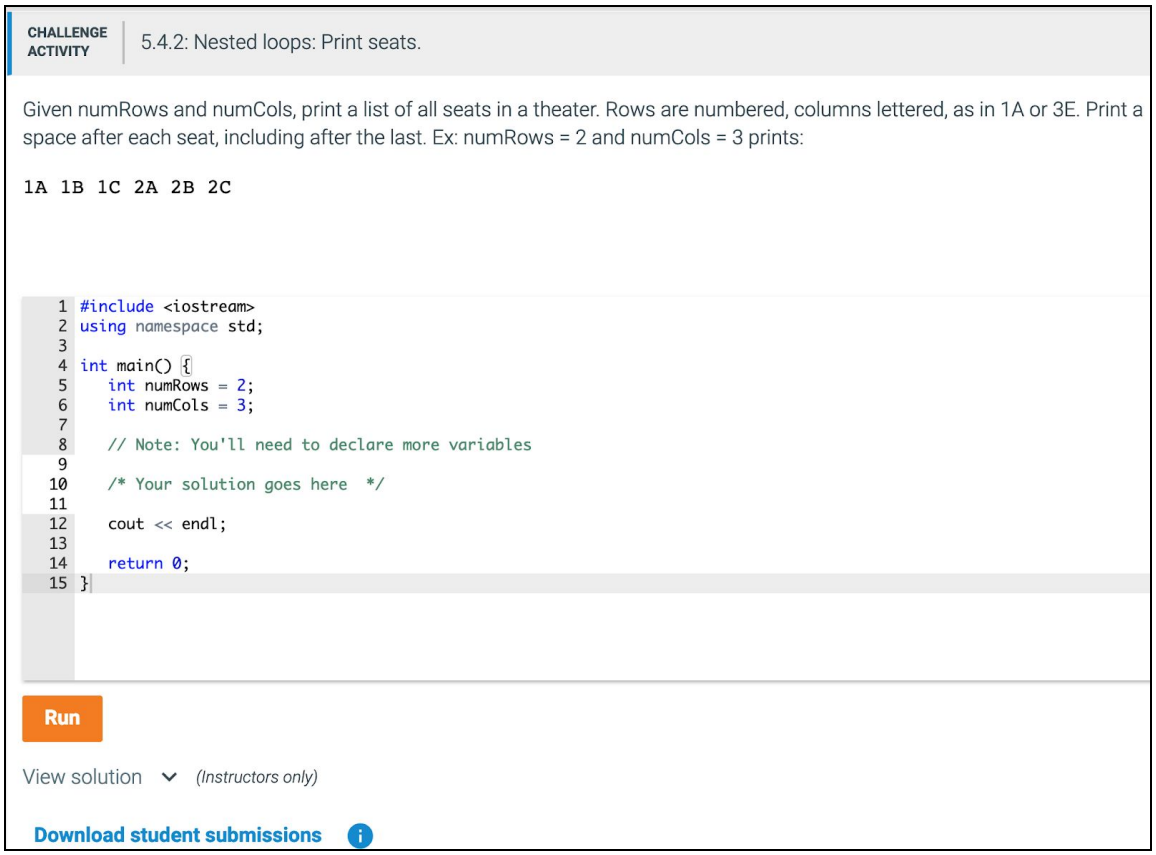

Figure 1: A coding activity example: Writing nested loops. This activity has a high struggle rate.

An instructor can download all submissions for any CA, yielding a csv file having every submission, both incorrect and correct. Each submission has: the timestamp, a user unique ID #, Yes or No (correct or incorrect), and the submitted code.

We downloaded all submissions for all 89 CAs in our zyBook. For each CA, we computed the struggle rate, defined as the % of students who struggled. We used the struggle metric defined by researchers in [23] (other metrics are possible of course):

- (Student time > 5 min) **AND** (Student time > 2 * Baseline time) **AND** (Student attempts > 3) **AND** (Student attempts > 2 * Baseline attempts)
- **OR** the student spends more than 15 min.

Figure 2: The struggle metric.

For a gap between a student's submissions of more than 10 minutes, we assume the student stepped away, and thus excluded such gaps from time.

3 Manually finding errors experienced by struggling students

Using the above automated analysis, we found the 10 CAs with the highest struggle rates. We manually (i.e., human analysis) examined student submissions to determine the common errors, and number of attempts and time spent by students specifically fixing each error.

Human analysis was preferred because submissions commonly had multiple errors. Humans used judgement to determine what error astudent was likely working on. For example, the following submissions have three errors: incorrect left-side variable in line 2, incorrect squaring in line 2, and misspelled area variable in line 3.

```
1: int area = 0;
2: r = PI * r * 2;
3: cout << arae << endl;
```

- 1 min later, the student fixes the spelling of area, so we say that 1 min was spent on the spelling error.
- 1 min later, the student adjusts the initial value of area, probably to see the impact on output. We attribute this attempt to the incorrect left-side variable error.
- 1 min later, the student adds a cout of r as well, probably to make sure r's value is as expected. We again attribute this attempt to the left-side error.
- 1 min later, the student changes the left-side to area. We attribute to the left-side error, which is now fixed.
- In 5 submissions over the next 9 minutes, the student tries changing line 2's expression to PI * 2 * r, then 2 * r * PI, then PI * (r * 2), then PI * r * 2.0, and finally PI * r * r.

Humans can recognize what the errors were, and can attribute 3 attempts and 3 minutes to solve the left-side error and 5 attempts and 9 minutes to the squaring error. Most previous work was automated, and would have considered the squaring error as 8 attempts and 13 minutes because that error existed in each submission, but a human can see that the student was focused on a different error for the first 3 minutes. Such overestimates can be significant in CAs that commonly see multiple errors.

The analyses were carried out by 3 upperclass-student CS majors, with close monitoring by a professor and a postdoctoral researcher. Total time was about 10 hours per CA.

Because our focus is on errors experienced by students who struggle, we manually examined every submission for every struggling student, and tallied their errors, attempts, and time spent, as in the above example. Our focus was not on non-struggling students, and our resources did not

allow manually examining every non-struggling student. Nevertheless, comparing is interesting, so we also manually examined a random subset of non-struggling students and calculated values for their errors, and scaled those values to gets estimates for all non-struggling students.

4 Common errors among strugglers

Fig. 3 shows a sample CA (Whitespace replace CA 4.3.2) with a student's submission in the top-right corner), with a student errors circled in red. Fig. 4 to Fig. 9 show common errors for 6 out of 12 CAs with the highest struggle rates (56% to 34% struggle rates for those 6 CAs). The struggle rates are calculated according to the struggle metric defined in Fig. 2. The remaining 6 figures are omitted for space reasons. The results are separated for struggling students and non-struggling students. In the bar charts, a (G) at the end of the error stands for a general error while (S) stands for a problem-specific error. A bar's label x/y means x is the number of students, and y is the median number of submissions containing that error. The time is the average time spent solving that specific error.

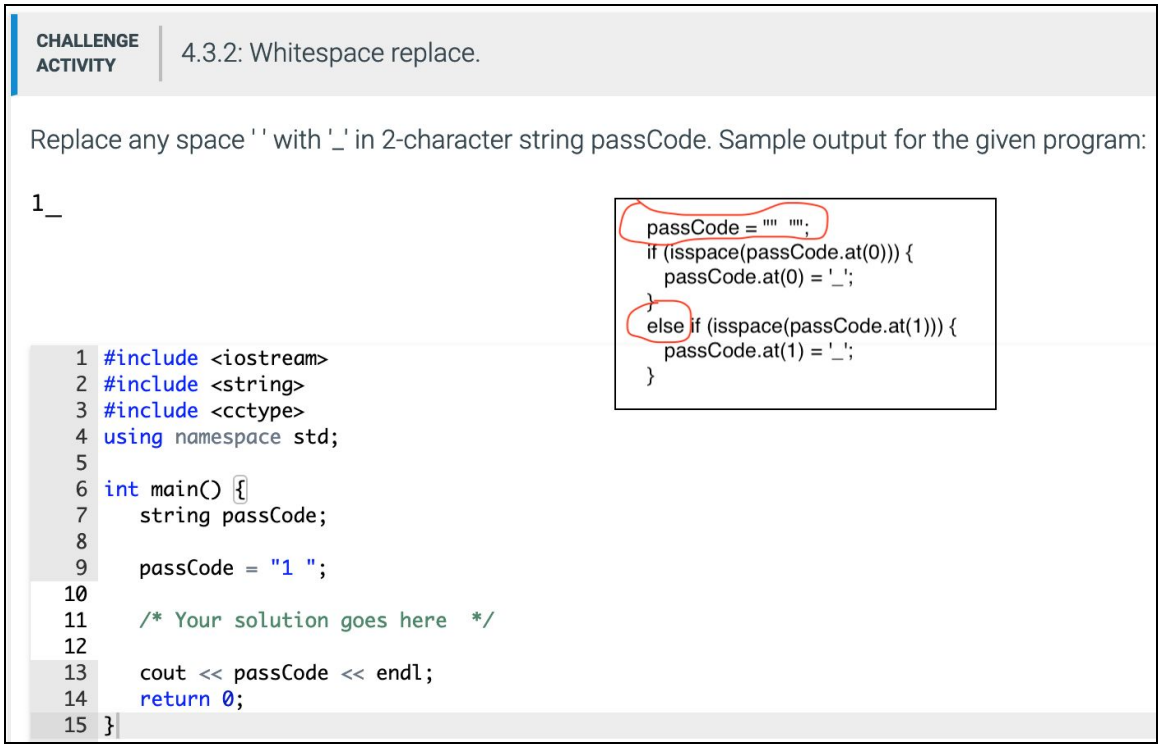

Figure 3: Whitespace replace (CA 4.3.2) CA with a student's submission in the top-right corner.

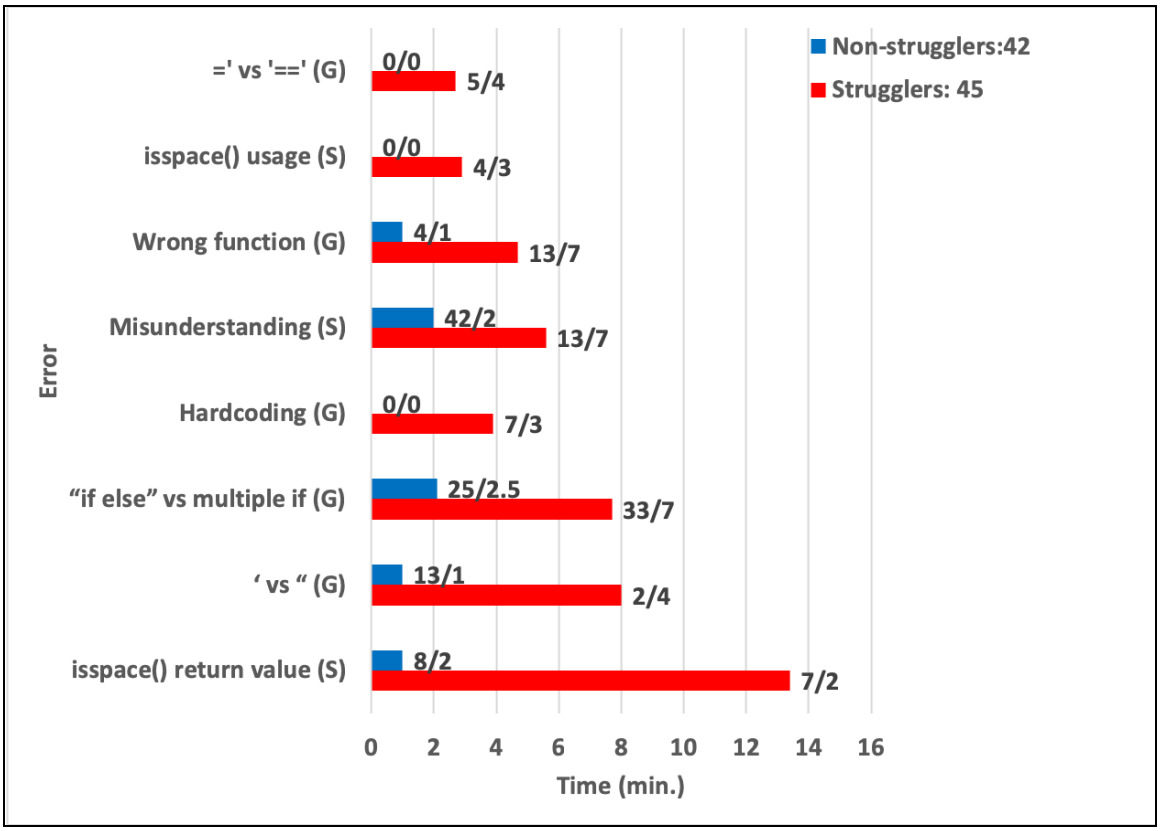

Figure 4: Whitespace replace (CA 4.3.2).

Fig. 4's CA appeared in the fourth week covering branches, and asked students to replace any spaces in a two-character string by an underscore. The error that caused the most struggle was students mis-understanding the return value of library function isspace(), which returns 0 (not a space) or a non-zero value (is a space) -- 7 students compared with true or 1 instead of just using "if (isspace(x))", causing struggle averaging 13.4 minutes. 8 other students did so too, but quickly fixed their mistake. 2 students struggled due to using ' instead of " or vice-versa. 33 students struggled due to using if-else instead of multiple if statements. 7 students struggled due to trying to hardcode a solution based on the test cases (trying to "cheat" the auto-grader). 13 students struggled due to misunderstanding the question, trying to solve a different problem (something that human analysis could determine, but a fully-automated analysis would not have). 13 struggled due to using a wrong library function other than isspace (such as isalpha()).

The CA in Fig. 5 was also in the fourth week with branches, asking students to set a boolean with true if a 3-character string contained a digit. 33 students struggled due to syntax errors. In fact, 11 used camel-case IsDigit as taught by the textbook, rather than the lowercase of the isdigit() library function, and took much time realizing the error. 24 struggled due to not checking each character individually. We omit further discussion of the errors for this CA, and for subsequent CAs, as the discussions are similar.

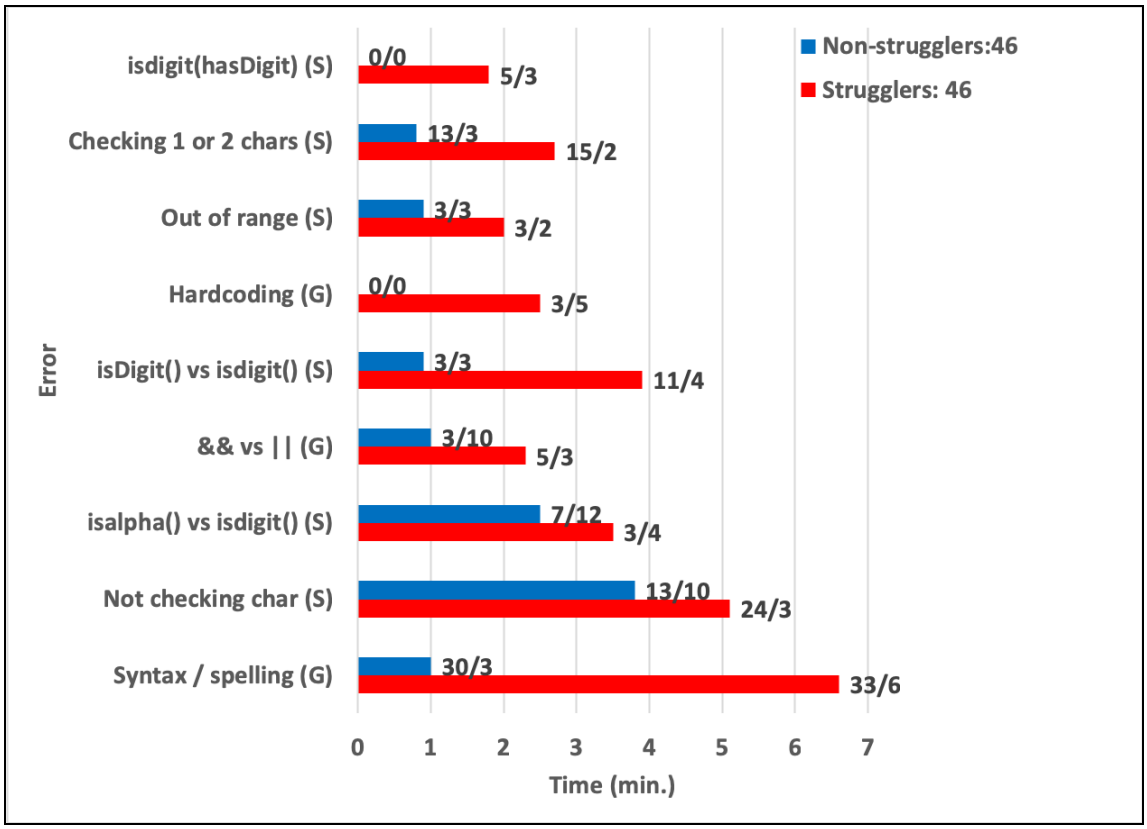

Figure 5: String with digit (CA 4.3.1).

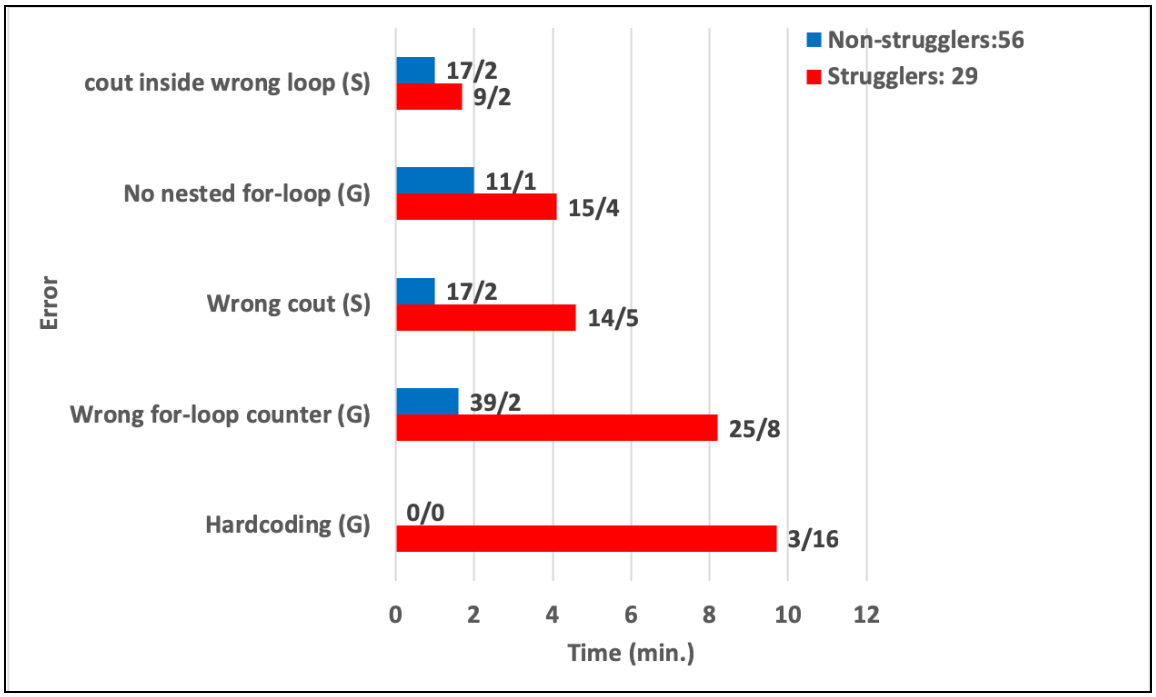

Figure 6: Nested loops Indent (CA 5.4.1).

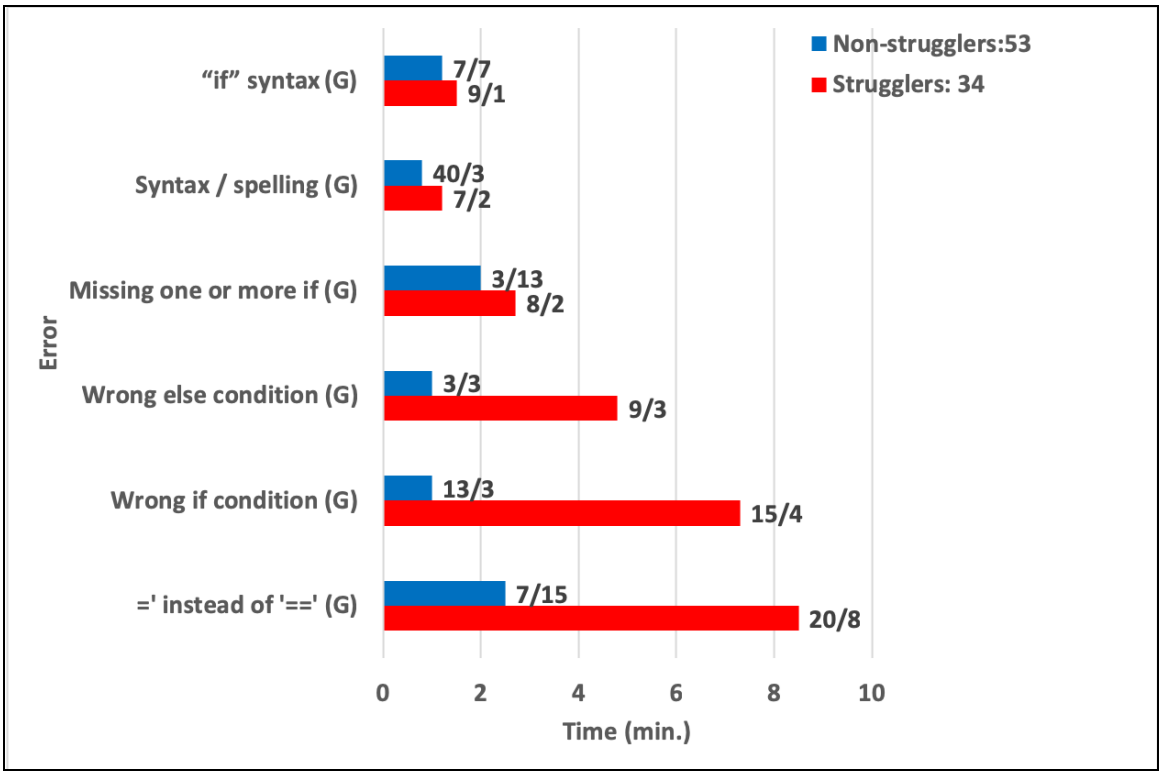

Figure 7: Bool in branching statements (CA 3.10.2).

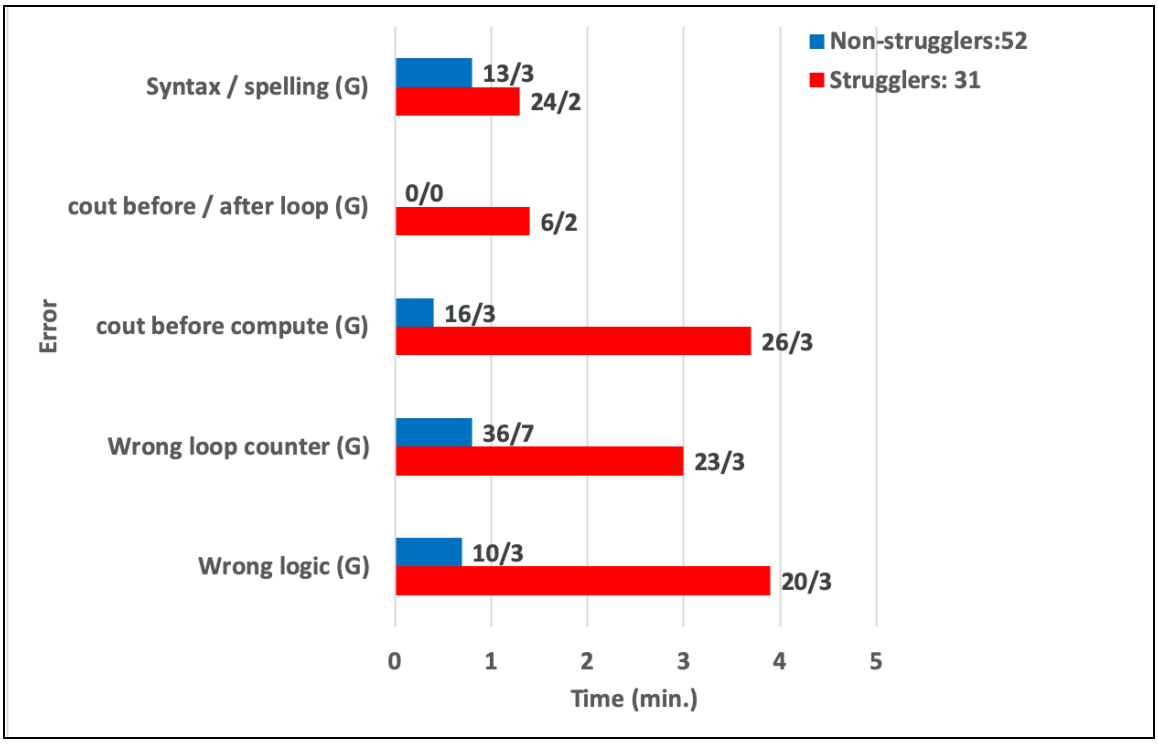

Figure 8: Basic while loop expression (CA 4.7.3).

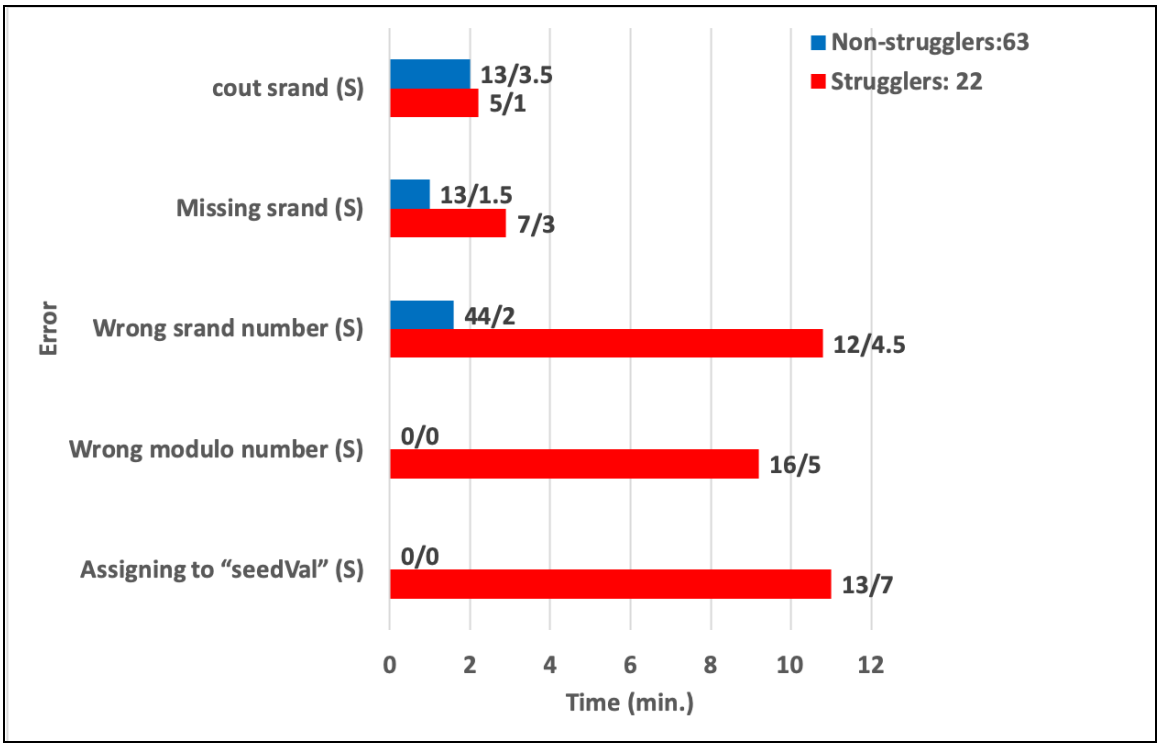

Figure 9: Rand function seed and then get random numbers (CA 5.5.2).

5 Common errors discussion: General vs. problem-specific

A key finding of our analysis is that while the top struggle causing errors included "general" programming errors, they also included "problem-specific" errors. General errors include errors like confusing ' and ", confusing if-else and multiple ifs, confusing && and ||, using = rather than ==, or using a wrong if condition.

Error
Using if and else instead of else if
Using = to compare
Syntax / spelling
For loop/while loop conditions/logic (infinite loop, not updating variable, etc)
Wrong if condition logic
Confusing single quotes with double quotes
Cout before cin
Logic error (answer logic, using multiplication, subtraction, division, strings, etc)

Table 1: Most common general errors.

In contrast, problem-specific errors were very specific to a particular CA, such as misunderstanding the return value of isspace(), using the wrong library function like using isalpha() instead of isdigit(), placing a cout statement before a computation in a loop (causing values to be off by one iteration versus the desired output), or failing to seed a random number generator per the problem's instructions. These errors are unlikely to show up in a typical analysis of common errors, yet are just as problematic for students. In fact, some errors even depend on where the problem appears: the isDigit() vs. isdigit() error in Chapter 4 (Strings/Loops 1) is due to the CA being the first to use a two-word library function -- Chapter 2 (Variables/Assignments) requires only one-word math functions like pow() and sqrt().

While those problem-specific errors could be generalized as "misusing functions" or "mis-ordering statements", such generalizations do not afford teachers much opportunity to improve and provide little chance for automated hints.

Thus, a key finding of our analysis is that error analysis should include detecting problem-specific errors, so teachers can provide improved problem-specific instructions, and an automated help system should allow adding problem-specific hints. For example, for CA 4.3.2 in Fig. 4, a hint system could auto-detect "isspace(x) == true" and provide a specific hint like "isspace() returns 0 or non-zero; your code has isspace(x) == true. Should not assume non-zero is true or 1. Instead, use just isspace(x)." Creating an exhaustive list of misuses of all library functions is unreasonable; focusing on the particular errors causing struggle is more feasible. Similarly, a problem that requires seeding a random number generator can detect absence of srand() and give a hint like "This problem requires srand() to seed the random number generator, but your code is missing srand()." Such hints can reduce much struggle while keeping the student learning.

For completeness, we list the most common general errors in Table 1, as done in most prior research. Our list has many similarities to previous lists, being a subset due to us focusing on 89 CAs from a zyBook, and also due to us focusing on errors that caused struggle.

6 Top struggle situations for struggle-causing errors

We were curious to know what particular errors caused the worst struggle situations. For the 12 CAs that we manually examined and recorded the time per error for each student, we created a list of the time spent by each student on each CA on each error, and then sorted in descending order. Table 2 shows the top 30. We see that the worst scenarios are due largely to both general and problem-specific errors, reinforcing the conclusion that we must focus on both if we wish to help students.

But additionally, we see that some errors that caused the worst struggle scenarios are not common errors, neither general nor problem-specific, but rather are just "silly" mistakes that students made and didn't recognize. We call these "one-off" errors. They appeared throughout the top 100 scenarios as well.

For example, one student accidentally declared a variable as int instead of char, spending 37 minutes because of that error. Another student just couldn't get a newline (endl) placement correct to pass the auto-grading, spending 34 minutes. One student failed to include a condition in an if statement, having an open parenthesis, and simply could not understand the compiler error, spending 19 minutes. Looking further in the list, we saw a semicolon after an else keyword (which is not a syntax error) causing struggle. On the one hand, experiencing and fixing errors is part of learning; on the other hand, no human teacher would leave a student to struggle for 15-30 minutes because of such "silly" mistakes.

Thus, a second conclusion from our analyses is that one-off errors, though uncommon, should be detected if possible (like semicolon after else); and if not possible, students spending excessive time need a way to obtain quick help.

User ID	CA	Time (min.)	General error Problem-specific error One-off error
5680	4.4.2	50	Hard coding
1055	5.4.2	48	Nested loop conditions
0847	5.4.1	46	Wrong for loop num and operator
1055	4.4.2	43	Hard coding
1131	5.4.2	37	Using int instead of char
2055	3.7.1	35	Misunderstanding question (thinks problem is asking for a range of values, when it just wants specific values)
0514	4.8.2	34	No endl at the end or endl inside the while loop
0525	2.10.2	31	Incorrect math (various combinations of division, modulus and subtraction of different numbers)
0765	3.10.2	30	Using = instead of ==
1929	5.4.2	29	Not realizing character has an ASCII value in loop condition (eg. for(seat = 'A' ; seat <= 3; ++seat) this loop will never be entered)
9967	9.1.2	28	Not taking 1 element array into account
9603	5.4.2	26	Outputting ASCII number instead of character
0676	4.3.2	25	Writing "str.isspace()==true" when it actually returns an integer
0241	3.10.2	25	Using = instead of ==
1902	4.3.2	24	Using "if/else" instead of "if and if"
9602	3.10.2	24	Wrong if condition logic (e.g. if (!isBalloon && isRed) instead of if(isBalloon && isRed))

0676	4.3.2	24	Writing "str.isspace()==true" when it actually returns an integer
2055	2.10.2	22	Wrong mod number
0525	3.10.2	22	Wrong if condition logic (e.g. if (isBalloon && isRed couts "Balloon" instead of "Red balloon"))
1131	4.4.2	21	Hard coding
0518	4.7.3	20	Not updating userNum correctly (failing to update the loop variable)
1770	4.8.2	20	Multiplying before couting (the counter variable is multiplied before outputted instead of outputted and then multiplied)
0266	5.4.1	20	Wrong cout statement, and wrong placement of the cout statement (happens at same time)
9599	2.14.1	20	cout before cin (outputting the variables before taking in the inputs)
0184	3.10.2	20	Wrong else logic (student used if, if, else instead of 4 ifs or if, else if, else that covers all 4 possibilities)
5611	9.1.2	19	Accessing array out of range
1770	4.3.2	19	Replacing only one space with _ (ex// using if and else, using single if statement, using ||)
0637	3.10.2	19	Using = instead of ==
0091	4.4.2	19	Not putting condition for if statement
5680	4.4.2	19	Wrong usage of string functions causing compile-time errors (append, push_back and insert)
9967	3.7.1	18	Misunderstanding question

Table 2: Top 30 errors.

7 Conclusions

We analyzed our students' submissions on 89 auto-graded coding Challenge Activities (CAs) from a zyBook used in our CS 1 class. For the 12 CAs with the highest struggle rates, we manually analyzed what errors caused struggle. We found many were due to common general errors, as found in various previous works. However, we also found that many errors were specific to a particular CA ("problem-specific"), such as misusing a particular library function, or misunderstanding instructions. We conclude that problem-specific errors are as important to detect as general errors, so that teachers can focus their teaching or modify instructions to reduce those errors. Moreover, automated hint systems should allow creating problem-specific hints based on such errors. Furthermore, we noticed that one-off errors -- errors that were uncommon -- were causing some of the worst struggle scenarios for students. These errors are neither general nor problem-specific errors. We thus also conclude that help systems should strive to detect as many one-off errors as possible and provide hints for those (the list may be huge), and that students struggling for more than some period of time should have a way to get quick help. We intend to make use of these finding to improve our own teaching and content, and to begin developing an automated help system for coding homework problems.

References

[1] Beaubouef, T. & Mason, J. Why the high attrition rate for computer science students: some thoughts and observations. ACM SIGCSE Bulletin, ACM, 2005, 37, 103-106.
[2] McCauley, R.; Fitzgerald, S.; Lewandowski, G.; Murphy, L.; Simon, B.; Thomas, L. & Zander, C. Debugging: a review of the literature from an educational perspective. Computer Science Education, Taylor & Francis, 2008, 18, 67-92.
[3] Altadmri, A. & Brown, N. C. 37 million compilations: Investigating novice programming mistakes in large-scale student data. Proceedings of the 46th ACM Technical Symposium on Computer Science Education, 2015, 522-527.
[4] Ettles, A.; Luxton-Reilly, A. & Denny, P. Common logic errors made by novice programmers. Proceedings of the 20th Australasian Computing Education Conference, 2018, 83-89.
[5] Hristova, M.; Misra, A.; Rutter, M. & Mercuri, R. Identifying and correcting Java programming errors for introductory computer science students. ACM SIGCSE Bulletin, 2003, 35, 153-156.
[6] Simon, B.; Fitzgerald, S.; McCauley, R.; Haller, S.; Hamer, J.; Hanks, B.; Helmick, M. T.; Moström, J. E.; Sheard, J. & Thomas, L. Debugging assistance for novices: a video repository. ACM SIGCSE Bulletin, 2007, 39, 137-151.
[7] Garner, S.; Haden, P. & Robins, A. My program is correct but it doesn't run: a preliminary investigation of novice programmers' problems. Proceedings of the 7th Australasian conference on Computing education-Volume 42, 2005, 173- 180.
[8] Ahmadzadeh, M.; Elliman, D. & Higgins, C. An analysis of patterns of debugging among novice computer science students. Acm sigcse bulletin, 2005, 37, 84-88.
[9] Fitzgerald, S.; Lewandowski, G.; McCauley, R.; Murphy, L.; Simon, B.; Thomas, L. & Zander, C. Debugging: finding, fixing and flailing, a multi-institutional study of novice debuggers. Computer Science Education, Taylor & Francis, 2008, 18, 93-116.
[10] Vipindeep, V. & Jalote, P. List of common bugs and programming practices to avoid them. Electronic, March, 2005.
[11] Hall, M.; Laughter, K.; Brown, J.; Day, C.; Thatcher, C. & Bryce, R. An empirical study of programming bugs in CS1, CS2, and CS3 homework submissions. Journal of Computing Sciences in Colleges, Consortium for Computing Sciences in Colleges, 2012, 28, 87-94.
[12] Cherenkova, Y.; Zingaro, D. & Petersen, A. Identifying challenging CS1 concepts in a large problem dataset. Proceedings of the 45th ACM technical symposium on Computer science education, 2014, 695-700.
[13] Alqadi, B. S. & Maletic, J. I. An Empirical Study of Debugging Patterns Among Novices Programmers. Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, 2017, 15-20.

[14] Brown, N. C. & Altadmri, A. Investigating novice programming mistakes: educator beliefs vs. student data. Proceedings of the tenth annual conference on International computing education research, 2014, 43-50.
[15] Mow, I. C. Analyses of student programming errors in Java programming courses. Journal of Emerging Trends in Computing and Information Sciences, 2012, 3, 739-749.
[16] Efopoulos, V.; Dagdilelis, V.; Evangelidis, G. & Satratzemi, M. WIPE: a programming environment for novices. ACM SIGCSE Bulletin, 2005, 37, 113- 117
[17] Pritchard, D. Frequency distribution of error messages. Proceedings of the 6th Workshop on Evaluation and Usability of Programming Languages and Tools, 2015, 1-8.
[18] Lee, G. C. & Wu, J. C. Debug it: A debugging practicing system. Computers & Education, Elsevier, 1999, 32, 165-179.
[19] Sirkiä, T. & Sorva, J. Exploring programming misconceptions: an analysis of student mistakes in visual program simulation exercises. Proceedings of the 12th Koli Calling International Conference on Computing Education Research, 2012, 19-28.
[20] Ebrahimi, A. Novice programmer errors: Language constructs and plan composition. International Journal of Human Computer Studies, London; San Diego: Academic Press, c1994-, 1994, 41, 457-480.
[21] Spohrer, J. C. & Soloway, E. Novice mistakes: Are the folk wisdoms correct? Communications of the ACM, ACM, 1986, 29, 624-632.
[22] zyBooks. https://www.zybooks.com/, August 2018.
[23] Alzahrani, N., F. Vahid, A. Edgcomb, R. Lysecky, and S. Lysecky. An Analysis of Common Errors Leading to Excessive Student Struggle on Homework Problems in an Introductory Programming Course. Proceedings of ASEE Annual Conference, 2018.