2021 ASEE ANNUAL CONFERENCE
Virtual Meeting | July 26–29, 2021 | Pacific Daylight Time
Paper ID #32801

Common Logic Errors for Programming Learners: A Three-decade Literature Survey

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.

©American Society for Engineering Education, 2021

Common Logic Errors for Programming Learners: A Three-Decade Literature Survey

Nabeel Alzahrani, Frank Vahid*

Computer Science and Engineering

University of California, Riverside

{nalza001, vahid}@ucr.edu

*Also with zyBooks

Abstract

We surveyed common logic errors made by students learning programming in introductory (CS1) programming classes, as reported in 47 publications from 1985 to 2018. A logic error causes incorrect program execution, in contrast to a syntax error, which prevents execution. Logic errors tend to be harder to detect and fix and are more likely to cause students to struggle. The publications described 166 common logic errors, which we classified into 11 error categories: input (2 errors), output (1 error), variable (7 errors), computation (21 errors), condition (18 errors), branch (14 errors), loop (27 errors), array (5 errors), function (24 errors), conceptual (43 errors), and miscellaneous (4 errors). Among those errors, we highlighted 43 that seemed to be the most common and/or troublesome. As interest in computer science education continues to grow, with college majors tripling in the past decade, this survey can help instructors, authors, and tool developers focus on helping learners detect or avoid these common errors in CS1.

Introduction

Interest in computer science continues to grow, with college computer science majors tripling between 2006 and 2015 [1, 2]. However, failure rates in introductory programming courses ("CS1") have been at a rather high 25-30% for several decades [3].

One contributor to poor CS1 performance is students struggling with programming errors. Thus, numerous researchers over the past decades have published errors made by students learning programming, hoping to aid instructors, authors, and tool developers in helping students detect or avoid such errors. Publications report different subsets of errors, due to variations in the language used, in the assignments students worked on, in the tools and instructional materials used, and in the help provided to students. We thus reviewed the publications to develop a more comprehensive summary of the common errors made by novice programmers. We focus on logic errors rather than syntax errors.

A syntax error is a program error that violates language rules and thus prevents execution. For compiled languages, a syntax error results in a compiler message, typically pointing to the erroneous program line. An example message is "Line 23: Missing semicolon". Syntax errors may annoy students and cause some struggle, but our experience is that logic errors cause more struggle. Syntax errors are covered by other works, such as Hristova [4] or Denny [5]. The latter lists top syntax errors as: cannot resolve identifier, type mismatch, and missing semicolon.

In contrast, a logic error appears in a syntactically-correct program that compiles and runs, but incorrectly attempts to solve the assignment given to the student programmer. An example is a loop that should iterate through an array but incorrectly stops one short of the array's last element. In our teaching experience, logic errors can be harder to detect and find than (most) syntax errors, and are a more common cause of substantial student struggle.

During our review, we found two publications, [7, 8], to be of particular interest due to not just reporting common errors, but also indicating the time required by students to find and fix the errors. Time is important, because some common logic errors are straightforward to find and fix, such as dividing by 0, as in sumItems / numItems where numItems is 0. That error may result in a runtime message that guides students directly to the offending program line and helps students immediately realize the problem, and whose fix may be a relatively straightforward check for 0 before dividing. On the other hand, some common logic errors are much harder to find and fix, such as performing integer division when intending for floating-point division, as in f = (9/5)*c + 32 (in Java, C, C++, etc.). That error may cause incorrect output, but the student doesn't know that the 9/5 (should be 9.0/5.0) or even that code line is the problem, and thus may try many different things, spending a lot of time and leading to struggle.

Altadmri [7] 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 was 17 min to 7 min. Ettles [8] 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. As will be seen, we give special weight to the time-consuming errors found by these two publications.

Our goal is to assist instructors, authors, and tool developers who wish to adapt their teaching techniques, learning content, languages, tools, and automated help systems to assist students in detecting or avoiding common logic errors.

Literature review method

To find publications relating to common errors, we carried out two tasks. (1) We searched on Google Scholar from 1985 to 2018 for a reasonable combination of these relevant (i.e., related to common novice-programmer logic errors) keywords: program, programmer, CS1, error, mistake, bug, fix, problem, novice, difficult, misconception, and manually examined titles of the search results to find relevant publications. (2) We manually examined titles of every paper published from 2008 to 2018 in the following 8 conferences and journals, which include a focus on CS education topics: the American Society for Engineering Education annual Conference (ASEE), the ASEE Computers in Education Journal (CoED), the ACM Global Computing Education Conference (CompEd), the Frontiers in Education Conference (FIE), the International Computing Education Research Conference (ICER), the Innovation and Technology in Computer Science Education Conference (ITiCSE), the Special Interest Group on Computer Science Education Conference (SIGCSE), and the ACM Transactions on Computing Education journal (TOCE). For both tasks, if a title seemed relevant (i.e., related to common novice-programmer logic errors), we manually examined the publication to see if the publication reported on common novice-programmer logic errors, and ultimately found 47 relevant publications.

For these 47 publications, we logged the errors described in each publication. The result was a raw list of 400 errors, but after processing these errors (remove duplication, remove non-CS1 related such as OOP, pointers, etc., and not counting generic errors such as "branching errors", "selection errors", etc.), we reduced the list to 166 errors. We classified the remaining specific 166 errors into 11 error categories based on commonality: input (2 errors), output (1 error), variable (7 errors), computation (21 errors), condition (18 errors), branching (14 errors), loop (27 errors), array (5 errors), function (24 errors), conceptual (43 errors), and Miscellaneous (4 errors), as shown in Table 1. Of the 166, we highlighted (in bold) 43 that appear in more than one paper. Of those 43, we further highlighted (with an asterisk) 11 that were reported to be time-consuming by either Altadmri et al. [7] or Ettles et al. [8]. Section "Common Errors" shows the errors within each category in a tabular format in Table 2 to Table 12.

For the categorization, we did not use any algorithm or formal process to develop the categories. We used our own judgments based on our own experiences to group together related errors based on commonality in 11 categories. The classification was done by the first author and was reviewed as needed by the second author. For verification and correctness, the first author redid the categorization multiple times. Different categorizations are possible of course; no one categorization is exclusively "correct".

Our study focuses on logic errors typically found in CS1 courses; therefore, we exclude errors commonly found in later courses (or late in some CS1 courses) like object oriented programming (classes, objects, interfaces, inheritance, overriding, constructors, accessors/modifiers, etc.), recursion, pointers/references, exception handling, data structures beyond arrays and strings such as linked lists, trees, and graphs, GUI and event-driven programming, etc.

No.	Category	# errors	Most common errors
1	Input	2	Erroneous prompting
2	Output	1	Order of output statements
3	Variable	7	Uninitialized variables
4	Computation	21	Integer division
5	Condition	18	&& and || operators
6	Branching	14	Multiple If vs If-else
7	Loop	27	Loop counter
8	Array/String	5	Indexing
9	Function	24	Return value
10	Conceptual	43	Lack of plan
11	Miscellaneous	4	Typos
Total no. of errors			166

Table 1: 11 error categories for the 166 errors in the 47 publications from 1985 to 2018 with an example of the most common error in each category.

Obviously, we cannot provide explanations and examples for all 166 logic errors. Instead, we highlighted in bold errors reported in multiple publications, and highlighted with an asterisk errors that [7, 8] found to be the most time-consuming, yielding 43 highlighted errors. References are included for all 211 errors, however, so that a reader can find details in previous publications of any error of interest.

Common errors

Table 1 lists the 11 error categories we defined for the 166 errors, with a column showing the number of errors in each category and another column for the most common error in each category. The sections below provide further details on those 166 errors. For each error category, we use a tabular format (Table 2 to Table 12) to decompose generic and specific errors that we defined, each with a bulleted list of errors. Generic and specific errors include citations (in the form of the first author last name and the last two digits of the year of publication) of several publications (but not necessarily all) that discussed the item. Note that some publications only discussed errors generically (like "Input errors") while others described specific errors (like "Waiting for input without prompting"). For each table, we highlighted common errors in bold and with an asterisk (as we discussed at the end of previous section “Literature review method”).

Generic errors	*Specific errors*
• Erroneous input *[Grandell05]*	• **Erroneous prompting** *[Efopoulos05, Simon07]* • Putting input statements in the wrong order *[Alzahrani18]*

Table 2: The 2 input specific errors.

Generic errors	*Specific errors*
• Output fragment *[Spohrer85]*	• **Putting output statements in the wrong order** *[Alzahrani18, Lee99, Spohrer85]*

Table 3: The 1 output specific error.

Generic errors	*Specific errors*
• Variables *[Caceffo16, Hanks08, Qian17, Robins06]*	• **Incorrect initialization** *[Garner05, Hall12, Fitzgerald08, Murphy08]* • Incorrect or redundant variables *[Grandell05]* • Subscripting variables incorrectly *[Hall12]* • **Uninitialized variables*** *[Ahmadzadeh05, Ettles18 (2min, 13%), Garner05, Raana15, Robins06, Truong04]* • Unset flags *[Hall12]* • Use of variables for input and output operations *[Qian17]* • Using the wrong variable type *[Hall12]*

Table 4: The 7 variable specific errors.

Generic errors	*Specific errors*
• Array / string errors *[Bryce10, Efopoulos05, Garner05, Hanks08, Robins06, Wiegand16]* • Array initialization *[Garner05, Hanks09, Robins06]* • String functions *[Garner05, Robins06]* *Indexing / iterating arrays [Alzahrani18, Bryce10, Garner05, Kurvinen16, Ettles18 (20 min. 33%), Robins06]*	• **Array declared with incomplete initialization** *[Garner05, Wiegand16, Robins06]* • **Buffer overflow** *[Alqadi17, Vipindeep05]* • Indexing into empty *[Cherenkova14]* • **Referencing data out of bounds *** *[Alqadi17, Efopoulos05, Ettles18, Hall12, Izu16, Mow02, Simon07, Teorey01]* • String comparison [Hanks09]

Table 9: The 5 array specific errors.

Generic errors	*Specific errors*		
• Computational problems *[Garner05, Hall12 (8%)]* • Expression *[Souza17, Robins06]* • Possible loss of precision *[Mow02]* • Referencing data *[Hall12]*	• Accumulate boolean *[Rosbach13]* • **Arithmetic** *[Garner05, Wiegand16, Robins06]* • **Arithmetic errors** *[Murphy08, Rosbach13]* • **Assignment** *[Caceffo16, Ebrahimi94, Garner05, Raana15, Robins06, Sirkia12, Souza17, Wiegand16]* • **Casting*** *[Ettles18, Garner05, Hristova03, Simon07, Robins06]* • Chained relational *[Wiegand16]*	• Incorrect operands or operators *[Hall12]* • **Integer division*** *[Alqadi17, Cherenkova14, Ettles18 (6 min. 72%), Fitzgerald08, Wiegand16]* • Inverted assignment *[Sirkia12]* • **Logical / boolean** *[Alzahrani18, Caceffo16, Ebrahimi94, Garner05, Wiegand16, Robins06]* • Incorrect calculations to support logical algorithm correctness *[Fitzgerald08]* • Missing computations *[Hall12]*	• Pre and post fix assignments *[Wiegand16]* • Random range *[Alzahrani18]* • Relational *[Wiegand16]* • Remainder operator with real operands *[Wiegand16]* • Rounding or truncation mistakes *[Hall12]* • **Type mismatch** *[Ahmadzadeh05, Garner05, Pritchard15, Seo14 (25%), Robins06]* • **Misunderstanding of operator precedence** *[Spohrer86, Teorey01, Robins06]* • Parenthesis used incorrectly *[Hall12]* • **Wrong formula** *[Simon07, Spohrer85]*

Table 5: The 21 computation specific errors.

Generic errors	*Specific errors*		
• Boundary case condition *[Spohrer86, Robins10, Rosbach13, Spohrer85]* • Conditionals *[Cherenkova14, Garner05, Qian17, Robins06]* • Relational operator *[Spohrer85]*	• **= vs ==** *[Alqadi17, Ebrahimi94, Hanks08, Kiran15, Raana15, Simon07, Sirkia12]* • Accidentally including sentinel values in a computation *[Simon07]* • Checking the wrong variable *[Hall12]* • Comparison *[Kurvinen16]* • Condition on rule wrong *[Winikoff14]* • **Condition variable has not been updated** *[Alqadi17, Rosbach13]*	• **Missing && and || operator *** *[Alqadi17, Altadmri15, Alzahrani18, Fitzgerald08, Simon07, Spohrer86]* • Missing condition tests *[Hall12]* • Not checked border value *[Cherenkova14]* • Numerical values are used as boolean operands *[Wiegand16]* • Perform unnecessary checking with Boolean expression *[Truong04]* • Reversed comparison operator *[Cherenkova14]*	• **Truth tables** *[Caceffo16, Garner05, Robins06]* • Unexpected cases problem. Boundary cases may not be considered [Robins10] • **Using == instead of equals() to compare strings *** *[Altadmri15 (17 min.), Brown14, Murphy08, Simon07]* • Wrong condition *[Rosbach13]* • Wrong False [Sirkia12] • Zero is excluded [Spohrer85]

Table 6: The 18 condition specific errors.

Generic errors	*Specific errors*
• Typo *[Garner05]*	• **Duplicate tail digit problems involve dropping the final digit from a constant with duplicated tail digits** *[Spohrer85, Spohrer86]* • **Empty statement blocks introduced with a misplaced semicolon** *[Simon07, Raana15]* • **Trivial typos - mistakes of typing (e.g. - for +) not caught by the compiler** *[Winikoff14]* • Wrong constant *[Spohrer85]*

Table 12: The 4 miscellaneous specific errors.

Generic errors	*Specific errors*	
• If statements *[Ebrahimi94, Garner05, Robins06]* • Jump *[Souza17]* • Selection *[Garner05, Souza17, Wiegand16, Robins06]* • Switch statements *[Alqadi17, Bryce10, Garner05, Wiegand16, Souza17, Robins06]*	• Break *[Souza17]* • Continue *[Souza17]* • Dangling else *[Teorey01]* • Failing to jump upon selection *[Sirkia12]* • Forgetting cases or steps *[Hall12]* • Identifying the output of an "if-else" statement with condition and nested "if" statements *[Wiegand16]* • **Missing "break" keywords in "switch" statement** *[Alqadi17, Raana15, Truong04, Wiegand16]*	• Wrong branch *[Sirkia12]* • Missing implication of if/else placing code outside the begin/end block *[Spohrer85]* • **Omitted "default" case in a "switch" statement** [Bryce10, Truong04] • Return *[Souza17]* • Swapping conditional block bodies in an "if" statements *[Fitzgerald08]* • Too many conditional statements *[Truong04]* • **Using multiple "if" flow structure instead of "if-else"*** *[Alzahrani18, Ettle18 (5 min. 11%), Souza17, Rosbach13]*

Table 7: The 14 branching specific errors.

Generic errors	*Specific errors*		
• Loop contents *[Garner05, Lee99, Robins06]* • Loop errors *[Alzahrani18, Bryce10, Caceffo16, Cherenkova14, Ebrahimi94, Garner05, Hanks08, Izu16, Qian17, Robins06, Rosbach13, Simon07, Souza17, Wiegand16]* • Loop with conditionals *[Cherenkova14, Garner05]*	• Classic logic errors in searches *[Simon07]* • Code inside a loop that does not belong there *[Teorey01]* • Code that appears and is executed after the loop exits *[Wiegand16]* • Conditional into loop control variable *[Sirkia12]* • Empty loop *[Izu16]* • Erroneous incrementing of a loop counter variable (i.e., outside the loop) *[Efopoulos05]* • **For loop is not inclusive*** *[Ettles18 (2 min. 13%)]* • How and when to terminate loops *[Ebrahimi94]* • **Improper / malformed loop** *[Caceffo16, Hall12, Lee99, Murphy08, Teorey01, Wiegand16]*	• Incorrect (including none) initialization of loop control variable *[Lee99]* • Incorrect update of the control variable *[Lee99]* • Indices in loops *[Kurvinen16]* • **Infinite loop** *[Bryce10, Izu16]* •Initialization of loop control variable is incorrectly placed *[Lee99]* • Loop containing "continue" statement *[Wiegand16]* • Loop has no body, extra semicolon *[Alqadi17]* • **Loop headers** *[Alzahrani18, Garner05, Robins06]* • Loop whose condition is	an assignment statement or a conjunctive logical expression *[Wiegand16* • Missing input statement inside the loop, resulting in only one set of data read *[Lee99]* • Nested loop initialization, expression *[Alzahrani18]* • **Off by 1 *** *[Alqadi17, Bryce10, Cherenkova14, Ettles18 (8 min. 19%), Fitzgerald08, Izu16, Spohrer85, Teorey01, Vipindeep05]* • Stop incrementing sum *[Cherenkova14]* • Too many loop and conditional statements *[Truong04]* • Unedited loop *[Izu16]* • Unnecessary output statement within the loop *[Lee99]* • Wrong semantics of nested loops *[Izu16]*

Table 8: The 27 loop specific errors.

Generic errors	*Specific errors*		
• Definition, data flow and header mechanics *[Garner05, Hanks08, Robins06]* • Function errors *[Qian17, Wiegand16]*	• **Always return -1*** *[Ettles18 (17 min. 12.8%)]* • **Call by reference vs call by value semantics** *[Caceffo16, Wiegand16]* • Data type of the value in the return statement is incompatible with the return type of the function *[Wiegand16]* • **Flow reaches end of non-void method** *[Altadmri15, Hristova03]* • Function name and scope *[Caceffo16]* • Incompatible types between method return and type of variable that the value is assigned to *[Altadmri15]*	• Incorrect / redundant variables or subroutines *[Grandell05]* • Initialization of formal parameters *[Caceffo16]* • Inverse nesting *[Sirkia12]* • Mismatch return type with its invocation *[Hristova03]* • Misplacing main method *[Simon07]* • Misplacing return value *[Sirkia12]* • Missing method call *[Rosbach13]* • Multiplying with a function call *[Sirkia12]* • Parameter values return value *[Qian17]*	• Parameters as local variables *[Caceffo16]* • Re-calling a function *[Sirkia12]* • **Return ignored*** *[Altadmri15 (15 min.), Brown14, Hristova03, Simon07, Sirkia12]* • Return statement is missing in the definition of a non-void function *[Wiegand16]* • **Returning 0 instead of -1*** *[Ettles18 (5 min. 22%)]* • **Unnecessary / not enough / too large method** *[Hall12, Truong04]* • **Wrong arguments** (out of order / type mismatches) *[Ahmadzadeh05, Altadmri15, Caceffo16, Hristova03, Simon07]*

Table 10: The 24 function specific errors.

Generic errors	*Specific errors*		
• Conceptual errors *[Hall12 (58%)]* • Misunderstanding / misinterpretation *[Spohrer86, Robins10]* • Problem solving *[Bryce10, Pillay06]*	• Action definition wrong *[Winikoff14]* • Action(s) of rule wrong (but legal) [Winikoff14] • Additional (wrong) rule *[Winikoff14]* • Cognitive load problem *[Spohrer86]* • Composition problem *[Spohrer86]* • Duplicating logic *[Hall12]* • **Expectation and interpretation problem** *[Bryce10, Spohrer85, Spohrer86]* • Fault in domain knowledge *[Winikoff14]* • Fault in initial beliefs / goals *[Winikoff14]* • Improper location of the assignment expression *[Ebrahimi94]* • Incorrect / missing algorithm *[Grandell05]* • Incorrect grouping *[Rosbach13]* • Incorrect identification of the control structure needed *[Pillay06]*	• Incorrect transfer of knowledge *[Pillay06]* • Inefficient problem solving approach *[Pillay06]* • Interpretation problem [Robins10] • **Lack of conceptualization of the execution of the problem** *[Kurvinen16, Qian17, Pillay06]* • Lack of knowledge or understanding of the programming language *[Pillay06]* • Lack of understanding of control structure *[Pillay06]* • Lack of understanding of the application domain *[Pillay06]* • Malformed right place (incorrect, but in the right place) *[Cunniff86]* • Misplaced (necessary but in wrong place) *[Cunniff86]* • Missing (required but not omitted) *[Cunniff86]* • Missing action in a rule *[Winikoff14]* • Missing rule *[Winikokk14]*	• Mixed up of constructs (if and while) *[Grandell05]* • Natural-language problem *[Robins10]* • Not supported *[Spohrer86]* • Not using a compound statement when one is required *[Teorey01]* • **Optimization problem** *[Spohrer86, Robins10]* • **Plan dependency problems** *[Spohrer85, Spohrer86]* • **Previous experience** *[Spohrer86, Robins10]* • Related knowledge interference *[Spohrer86]* • Specialization problem *[Robins10]* • Spurious (not needed) *[Cunniff86]* • **Summarisation problem** *[Robins10, Spohrer86]* • Swap two variables without using a helper variable *[Kurvinen16]* • Unnecessarily complicated *[Rosbach13]*

Table 11: The 40 conceptual specific errors.

Discussion

Table 1 summarizes the most common logical errors in each category. Instructors, authors, and tool developers could adapt their teaching techniques, learning content, languages, tools, and automated help systems to assist students in detecting or avoiding these common logic errors. For example, instructors could teach students to always initialize variables before students use them. Similarly, tool developers may adapt their programming tools to give warnings if variables are declared without proper initializations.

The data of the survey (in Table 2 to Table 12) helped us to identify the most difficult errors that could lead to student struggle in each of the 11 categories. The data shows that errors are more related to advanced general programming concepts such as algorithms, loops, functions, etc. The data also did not show that more errors are related to the syntax and semantics of a programming language (specific-language programming errors). Therefore, instructors might wish to focus on generic programming errors over specific programming errors when providing interventions to help students.

This survey has several limitations. One of the main limitations we faced while conducting this survey is the common lack of the study setup and research methodology in the surveyed publications. For example, many publications did not mention the year when the data were collected, the course setting (e.g., number of instructors, number of TAs, and type of learning material, type of university, CS1 or CS2, etc.), the activity setting (e.g., programming language, number/nature of activities, at-home or in-class, with or without instructor/TA help, etc.), the population setting (e.g., number of students, age, gender, ethnicity, etc.), and/or the outcome measured (e.g., struggle-causing errors using time-to-fix average, etc.). The lack of such data made it difficult to compare between the publications. Also, such a lack of details made it difficult to assess the confidence in the publications' results such as how the authors decide something is an error and when it counts as fixed.

Another limitation is the definition of CS1. We defined, early in section "Literature review method", the CS1 topics that we covered in this study, but some researchers might not agree with our definition of CS1, and thus might not agree with the coverage of the errors in this study based on that definition. For example, if there is a publication about OOP challenges in CS1, we just picked up the non-OOP related errors from that publication. Also, our error categorization did not use a systematic approach based on clearly defined guidelines, but was subjective and based on the authors' own judgments. Some researchers might not agree with such a classification. Also, the errors mentioned in this study might not apply to all programming languages and might be language-dependent or/and language-specific. Furthermore, the time-to-fix parameter, which we used in this study to highlight errors that might cause struggle, might not be accurate due to multiple factors such as students stepping out rather than working on solving the bug.

Moreover, we did a Google Scholar search for publication in the period from 1985 to 2018 (3 decades) since the search was easy to do using Google search. However, the manual database search covered only

2008 to 2018 because it was difficult to do the search manually in 8 different databases for the last 3 decades. Another limitation is the keywords that we used in the search for related publications as explained in Section "Literature review method". We used a limited number of 11 keywords that might not be inclusive for all related publications in the last 3 decades. Also, some researchers might disagree with the keywords that we used. Lastly, we manually searched only 8 databases for related publications, so omissions of some other relevant databases might be a limitation and some researchers might disagree with having such limited database search. Even with the above limitations, we believe this survey is still helpful and more efficient than reading 47 publications.

Conclusion

We highlighted various errors that were indeed problematic and thus instructors, developers, and authors can focus on reducing those as well as the others too. As the data showed, the literature focused mainly on frequent errors but not on errors that caused struggle. For example, out of 47 literature materials, only 2 (4%) papers focused on errors that are difficult to fix. A frequent error is not necessarily problematic if easily detected and fixed by students, and in fact some would argue that such detecting/fixing is an important part of the learning process. In contrast, an error that causes struggle may lead to frustration and de-motivation without justifiable learning benefit. Detecting struggle was harder in the past due to a lack of online logging of student activity, but is more possible today with newer tools being used in CS1 classes. Thus, we encourage future work that increases focus on errors that cause struggle, and remedies to reduce such struggle.

Acknowledgements

This work was supported in part by the National Science Foundation (grant number 1563652).

References

[1] Stuart Zweben and Betsy Bizot. 2018. "2017 CRA Taulbee survey," Computing Research News 30, 5 (2018), 1–47.

[2] Tracy Camp, W. Richards Adrion, Betsy Bizot, Susan Davidson, Mary Hall, Susanne Hambrusch, Ellen Walker, and Stuart Zweben. 2017. "Generation CS: the growth of computer science," ACM Inroads 8, 2 (2017), 44–50.

[3] Christopher Watson and Frederick WB Li. 2014. "Failure rates in introductory programming revisited," In Proceedings of the 2014 conference on Innovation & technology in computer science education, 39–44.

[4] Maria Hristova, Ananya Misra, Megan Rutter, and Rebecca Mercuri. 2003. "Identifying and correcting Java programming errors for introductory computer science students," ACM SIGCSE Bulletin 35, 1 (2003), 153–156.

[5] Paul Denny, Andrew Luxton-Reilly, and Ewan Tempero. 2012. “All syntax errors are not equal” In Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education, 75–80.

[6] Andrew J. Ko and Brad A. Myers. 2005. “A framework and methodology for studying the causes of software errors in programming systems,” Journal of Visual Languages & Computing 16, 1–2 (2005), 41–84.

[7] Amjad Altadmri and Neil CC Brown. 2015. “37 million compilations: Investigating novice programming mistakes in large-scale student data,” In Proceedings of the 46th ACM Technical Symposium on Computer Science Education, 522–527.

[8] Andrew Ettles, Andrew Luxton-Reilly, and Paul Denny. 2018. “Common logic errors made by novice programmers,” In Proceedings of the 20th Australasian Computing Education Conference, 83–89.

[9] Linda Grandell, Mia Peltomäki, and Tapio Salakoski. 2005. “High school programming—a beyond-syntax analysis of novice programmers’ difficulties,” In Proceedings of the Koli Calling 2005 Conference on Computer Science Education, 17–24.

[10] Greg C. Lee and Jackie C. Wu. 1999. “Debug It: A debugging practicing system,” Computers & Education 32, 2 (1999), 165–179.

[11] Vassilios Efopoulos, Vassilios Dagdilelis, Georgios Evangelidis, and Maya Satratzemi. 2005. “WIPE: a programming environment for novices,” In Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education, 113–117.

[12] Nabeel Alzahrani, Frank Vahid, A. Edgcomb, R. Lysecky, and Susan Lysecky. 2018. “An Analysis of Common Errors Leading to Excessive Student Struggle on Homework Problems in an Introductory Programming Course,” In Proceedings of ASEE Annual Conference.

[13] Beth Simon, Sue Fitzgerald, Renée McCauley, Susan Haller, John Hamer, Brian Hanks, Michael T. Helmick, Jan Erik Moström, Judy Sheard, and Lynda Thomas. 2007. “Debugging assistance for novices: a video repository,” ACM SIGCSE Bulletin 39, 4 (2007), 137–151.

[14] Renée C. Bryce, Alison Cooley, Amy Hansen, and Nare Hayrapetyan. 2010. “A one year empirical study of student programming bugs,” In 2010 IEEE Frontiers in Education Conference (FIE), IEEE, F1G-1-F1G-7.

[15] Sandy Garner, Patricia Haden, and Anthony Robins. 2005. “My program is correct but it doesn’t run: a preliminary investigation of novice programmers’ problems,” In Proceedings of the 7th Australasian conference on Computing education-Volume 42, 173–180.

[16] J. C. Spohrer, E. Pope, M. Lipman, W. Sack, S. Freiman, D. Littman, W. L. Johnson, and E. Soloway. 1985. “Bug Catalogue: II, III, IV,” Yale University, Department of Computer Science.

[17] Ricardo Caceffo, Steve Wolfman, Kellogg S. Booth, and Rodolfo Azevedo. 2016. “Developing a computer science concept inventory for introductory programming,” In Proceedings of the 47th ACM Technical Symposium on Computing Science Education, 364–369.

[18] Brian Hanks. 2008. “Problems encountered by novice pair programmers,” Journal on Educational Resources in Computing (JERIC) 7, 4 (2008), 1–13.

[19] Yizhou Qian and James Lehman. 2017. “Students’ misconceptions and other difficulties in introductory programming: A literature review,” ACM Transactions on Computing Education (TOCE) 18, 1 (2017), 1–24.

[20] Anthony Robins, Patricia Haden, and Sandy Garner. 2006. “Problem distributions in a CS1 course,” In Proceedings of the 8th Australasian Conference on Computing Education-Volume 52, 165–173.

[21] Morgan Hall, Keri Laughter, Jessica Brown, Chelynn Day, Christopher Thatcher, and Renee Bryce. 2012. “An empirical study of programming bugs in CS1, CS2, and CS3 homework submissions,” Journal of Computing Sciences in Colleges 28, 2 (2012), 87–94.

[22] Sue Fitzgerald, Gary Lewandowski, Renee McCauley, Laurie Murphy, Beth Simon, Lynda Thomas, and Carol Zander. 2008. “Debugging: finding, fixing and flailing, a multi-institutional study of novice debuggers,” Computer Science Education 18, 2 (2008), 93–116.

[23] Laurie Murphy, Gary Lewandowski, Renée McCauley, Beth Simon, Lynda Thomas, and Carol Zander. 2008. “Debugging: the good, the bad, and the quirky--a qualitative analysis of novices’ strategies,” ACM SIGCSE Bulletin 40, 1 (2008), 163–167.

[24] Marzieh Ahmadzadeh, Dave Elliman, and Colin Higgins. 2005. “An analysis of patterns of debugging among novice computer science students,” In Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education, 84–88.

[25] A. Raana, M. A. Azam, M. A. Ghazanfar, A. Javed, Y. Amin, and U. Naeem. 2016. “C++ BUG CUB: Logical Bug Detection for C++ Code,” Nucleus 53, 1 (2016), 56–63.

[26] Nghi Truong, Paul Roe, and Peter Bancroft. 2004. “Static analysis of students’ Java programs,” In Proceedings of the Sixth Australasian Conference on Computing Education-Volume 30, Citeseer, 317–325.

[27] V. Vipindeep and Pankaj Jalote. 2005. “List of common bugs and programming practices to avoid them,” Electronic, March (2005).

[28] Hyunmin Seo, Caitlin Sadowski, Sebastian Elbaum, Edward Aftandilian, and Robert Bowdidge. 2014. “Programmers’ build errors: a case study (at google),” In Proceedings of the 36th International Conference on Software Engineering, 724–734.

[29] Alexander Hoem Rosbach. 2013. “Novice difficulties with language constructs,” The University of Bergen.

[30] Ioana Tuugalei Chan Mow. 2012. “Analyses of student programming errors in Java programming courses,” Journal of Emerging Trends in Computing and Information Sciences 3, 5 (2012), 739–749.

[31] Basma S. Alqadi and Jonathan I. Maletic. 2017. “An empirical study of debugging patterns among novices programmers,” In Proceedings of the 2017 ACM SIGCSE technical symposium on computer science education, 15–20.

[32] Yuliya Cherenkova, Daniel Zingaro, and Andrew Petersen. 2014. “Identifying challenging CS1 concepts in a large problem dataset,” In Proceedings of the 45th ACM technical symposium on Computer science education, 695–700.

[33] David Pritchard. 2015. “Frequency distribution of error messages,” In Proceedings of the 6th Workshop on Evaluation and Usability of Programming Languages and Tools, 1–8.

[34] Toby J. Teorey and Ann R. Ford. 2001. “Practical DeBugging C++,” Prentice Hall Professional Technical Reference.

[35] Draylson Micael de Souza, Michael Kölling, and Ellen Francine Barbosa. 2017. “Most common fixes students use to improve the correctness of their programs,” In 2017 IEEE Frontiers in Education Conference (FIE), IEEE, 1–9.

[36] R. Paul Wiegand, Anthony Bucci, Amruth N. Kumar, Jennifer L. Albert, and Alessio Gaspar. 2016. “A data-driven analysis of informatively hard concepts in introductory programming,” In Proceedings of the 47th ACM Technical Symposium on Computing Science Education, 370–375.

[37] Alireza Ebrahimi. 1994. “Novice programmer errors: Language constructs and plan composition,” International Journal of Human-Computer Studies 41, 4 (1994), 457–480.

[38] Teemu Sirkiä and Juha Sorva. 2012. “Exploring programming misconceptions: an analysis of student mistakes in visual program simulation exercises,” In Proceedings of the 12th Koli Calling International Conference on Computing Education Research, 19–28.

[39] James C. Spohrer and Elliot Soloway. 1986. “Novice mistakes: Are the folk wisdoms correct?,” Communications of the ACM 29, 7 (1986), 624–632.

[40] Anthony Robins, Janet Rountree, and Nathan Rountree. 2003. “Learning and teaching programming: A review and discussion,” Computer science education 13, 2 (2003), 137–172.

[41] Einari Kurvinen, Niko Hellgren, Erkki Kaila, Mikko-Jussi Laakso, and Tapio Salakoski. 2016. “Programming misconceptions in an introductory level programming course exam,” In Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education, 308–313.

[42] Michael Winikoff. 2014. “Novice programmers’ faults & failures in GOAL programs,” In Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems, 301–308.

[43] Eranki LN Kiran and Kannan M. Moudgalya. 2015. “Evaluation of programming competency using student error patterns,” In 2015 International Conference on Learning and Teaching in Computing and Engineering, IEEE, 34–41.

[44] Neil CC Brown and Amjad Altadmri. 2014. “Investigating novice programming mistakes: Educator beliefs vs. student data,” In Proceedings of the tenth annual conference on International computing education research, 43–50.

[45] Cruz Izu, Amali Weerasinghe, and Cheryl Pope. 2016. “A study of code design skills in novice programmers using the SOLO taxonomy,” In Proceedings of the 2016 ACM Conference on International Computing Education Research, 251–259.

[46] Brian Hanks and Matt Brandt. 2009. “Successful and unsuccessful problem solving approaches of novice programmers,” ACM SIGCSE Bulletin 41, 1 (2009), 24–28.

[47] Nelishia Pillay and Vikash R. Jugoo. 2006. “An analysis of the errors made by novice programmers in a first course in procedural programming in Java,” Preface of the Editors 84, (2006).

[48] Nancy Cunniff, Robert P. Taylor, and John B. Black. 1986. “Does programming language affect the type of conceptual bugs in beginners’ programs? A comparison of FPL and Pascal,” ACM SIGCHI Bulletin 17, 4.