Wiese, E.S., Rafferty, A.N., Kopta, D.M., & Anderson, J.M. (2019, May) Replicating Novices' Struggles with Coding Style. To appear in Proceedings of the 29th International Conference on Program Comprehension. Short paper to be presented at ICPC '19, Montréal, Canada

# Replicating Novices' Struggles with Coding Style

Eliane S. Wiese
*School of Computing*
*University of Utah*
Salt Lake City, USA
eliane.wiese@utah.edu

Anna N. Rafferty
*Computer Science Department*
*Carleton College*
Northfield, USA
arafferty@carleton.edu

Daniel M. Kopta
*School of Computing*
*University of Utah*
Salt Lake City, USA
dkopta@cs.utah.edu

Jacqulyn M. Anderson
*School of Computing*
*University of Utah*
Salt Lake City, USA
macharjk@gmail.com

***Abstract*—Good style makes code easier for others to read and modify. Control flow is one element of style where experts expect particular structure, such as conjoining conditions rather than nesting `if` statements. Empirical work is necessary to understand why novices use poor style, so they can be taught to use good style. Previous work shows that many students know what control flows experts prefer, but may say that novice-styled code is more readable. Yet, these same students showed similarly high comprehension across both expert- and novice-styled code. We propose a replication of that work that more fully assesses students' code comprehension and code writing. Our replication focuses on students who are earlier in their computer science courses and are less likely to be majors, to determine whether the pattern of results is particular to students who are relatively attuned to style concerns. Our pilot of the proposed replication finds that: students in this new population are less able to identify expert code; expert style may reduce comprehension for some control flows; and writing with good style does not always predict a preference for reading code with good style.**

***Index Terms*—Computer science education, Novice code comprehension, Programming style**

## I. Introduction

To become expert programmers, students must learn not only how to write code with desired functionality but also how to structure that code so it's readable and maintainable by others [1]. These latter skills are part of good style: writing code that follows conventions that are understood by the broader community [2]. Style encompasses a variety of different programming practices [3], including helpful comments [4], understandable variable names [5], and appropriate syntactic constructions [6]. Style is notoriously difficult to teach [1], [7], [8]. Designing effective style instruction requires empirical work on which style problems are common and determining if poor style is a symptom of conceptual misunderstanding [9]. Wiese, Rafferty, and Fox [10] examined these issues in seven different code patterns involving control flow and syntactic structure, and we replicate and extend this work.

### A. Original Study

The original study [10] surveyed students at the University of Utah on seven code patterns. Each code pattern had an expert version (*experts* here are instructors; *expert code* is how they want students to write: e.g., directly returning a Boolean condition) and a novice version, (e.g., checking a Boolean condition using an `if`-statement and then explicitly returning `true` or `false`). Students were asked which version of each pattern was most readable and which experts would label as having the best style. Students were also given code samples with each version, and asked what the output would be for some input, to ascertain whether students actually had trouble understanding different versions of the code or simply preferred one version. Students also wrote code that required using a Boolean condition, measuring how their code writing style compared to their preferences and comprehension.

In the original study [10], students found four novice patterns more readable than their expert-styled counterparts: checking a Boolean condition using an `if`-statement and then explicitly returning `true` or `false` (for conditions with and without relational operators); preceding a general solution with extraneous special cases; and repeating code within an `if`-block and its corresponding `else`. However, across all patterns, there was no significant effect of style on comprehension scores, nor was there an effect of which style students preferred. Writing style was highly correlated with readability preference for the one pattern with a writing question, suggesting that multiple-choice readability questions may be an efficient proxy for assessing writing style. The study did not find support for the hypotheses that students use poor style because (1) they don't know what expert style is, or (2) they don't understand code written with good style [10].

### B. Replication in Different Context

Our replication was conducted at a small liberal arts college. This kind of replication is important in determining whether the original findings [10] are typical of undergraduate students broadly. Student preparation across the two institutions differs, both in terms of curriculum and intention to major in CS. At the time of our survey, at the liberal arts college, most students had completed one ten-week introductory computing course (taught in Python), and were near the end of their second ten-week programming course, which focused on data structures and was taught in Java. While the coding patterns in the survey are applicable in both languages, the survey items were in Java. Wiese, Rafferty, and Fox [10] surveyed 231 undergraduate students in two courses: 75 in a data structures course and 156 in a software engineering course (the next course in the sequence). These students had more formal experience in Java - one or two 15-week semesters when they took the survey.

At the liberal arts college, style instruction emphasizes readability through variable naming and comments rather than

syntactic structure. While students at both schools had implicit exposure to the expert-style versions of these patterns through their textbook and in-class coding examples, the University of Utah particularly emphasized the types of style patterns that were on the survey, and even had an assignment in an introductory course specifically targeting one of the patterns. Further, 81% of students in the prior study [10] were majoring in CS or planned to, in contrast to 38% of our sample in this study. Most CS majors at the University of Utah are planning for a career that involves programming. The lower emphasis on the target style patterns, lower percentage of students intending to major in CS, and general pattern of liberal arts colleges having less emphasis on careers, may lead to the liberal arts college students being less focused on code style.

It is important to replicate the results with students earlier in their study of computer science and who are enrolled in a liberal arts college. First, the original study suggested that students would benefit simply from having their style errors flagged, since students mostly recognized and understood expert-styled code. However, if students with less experience with the programming language have more trouble comprehending expert-styled code, reminders to use good style may be asking students to use code that they don't understand. And, if students do not recognize expert-style code based on less emphasis on it in instruction, style guidelines may feel arbitrary. Thus, this pilot replication can address whether formal instruction on the expert style is necessary for students to recognize expert style, or if exposure is sufficient. Additionally, since the original study [10] found some code patterns that students had more difficulty comprehending, replicating this finding would indicate that those patterns are particularly difficult, perhaps necessitating other types of interventions.

## II. Methods

### A. Participants and Materials

Students enrolled in Data Structures (the second programming course) at the liberal arts college were recruited via their course mailing list. The instructor for one section gave extra credit for completing the survey. Of 64 students across two sections, 32 students began the survey. We analyze results from the 18 students who answered all questions (one skipped one part of a multi-part comprehension question).

We adapted the initial survey [10], examining the same seven control flow patterns where there are characteristic novice versus expert style differences (the original survey is available at https://tinyurl.com/RICE-Survey-pdf). That survey included three types of questions: code writing (write a function to accomplish a task), readability and style (from a set of code blocks with the same functionality, judge which is most readable and which is best styled), and code comprehension (given a function and input, determine the output).

Our survey used the same code blocks as the original survey [10] for the readability/style questions for three of the seven patterns (returning Boolean values with and without operators, and `for` vs. `while` loops). For the three patterns where the experts in the original study disagreed on which block was best styled (repeating code within an `if` and `else`, multiple cases, and `else-if`s vs. `if`s) we re-wrote the code blocks. Also, the prior work [10] may have found high comprehension for both novice-styled nested `if`s and expert-styled `&&` because the code did not require short-circuit evaluation. This may have overestimated students' understanding of the expert-style pattern, so we wrote new snippets where the expert version relied on short-circuit evaluation to avoid exceptions. Surveying three experts found unanimous agreement on the best-styled blocks for all new code.

The original study [10] found that code writing style was a stronger predictor than code comprehension of what style of code a student would prefer, based on a writing task for one code pattern. To investigate this relationship further, we included five writing tasks: (1) the original task (returning a Boolean with an operator); (2) returning a Boolean without an operator; (3) Nested `if` vs. `&&`; (4) one task targeting both Repeating code within `if` and `else` and `if` statements vs. `if-else`; and (5) one task targeting both Multiple Cases and `for` vs. `while` loops. Each task required only a few lines of code (e.g., "Write a function that takes an arrays of `int`s as input and returns an array of `int`s. The returned array should have the same content as the input array, but in reverse order."), and example inputs and outputs were given.

Code comprehension in the original study was assessed via multiple choice questions (given an input and a code block, select the output) [10]. Students performed very well, with over 70% correct on 12 of the 14 code samples. Those results [10] found a ceiling effect for comprehension questions on Boolean returns, so we dropped those items. Since we added writing and editing tasks, we trimmed some comprehension tasks to keep the survey under one hour. Instead of presenting the same inputs for different code blocks, we presented code samples that followed expert and novice patterns, and asked which had the same functionality (some were different). We added items on the number of comparisons performed in blocks of code that used a sequence of `if`s versus `if-else if` structure, to assess students' understanding of differences in code execution even when overall functionality is the same. We used the same comprehension questions as before [10] for the *Multiple Cases* pattern.

We compiled and tested all code samples for correctness; a few of the original study's code samples would have failed to compile due to missing semi-colons. To encourage students to focus on functionality instead of searching for syntax errors, we also removed the option "the code does not compile" from the comprehension questions, though we left the option that the code would throw an exception.

Finally, code editing questions were added to the survey in order to assess whether recognition of expert style is sufficient for improving the style of novice code. We describe the editing tasks in detail in their corresponding Results sections.

### B. Procedure

Students in the target course received a link to the Qualtrics survey via an email. For the section receiving extra credit, stu-

dents could choose to complete the survey without consenting to the research. Students completed the survey at their own pace, beginning with informed consent. Survey instructions indicated that the survey should take about an hour.

In the survey, students first completed the five code-writing questions, then fourteen questions about readability and style, then the comprehension questions, and finally the code-editing questions. For all sections but the code editing section, students were randomly counterbalanced to complete the questions in forward or reverse order. Questions in the code-editing section were ordered based expected difficulty, with easier questions first. After all code-related questions, students were asked for demographic information, including if they were majoring or intended to major in computer science.

### C. Threats to Validity

This study is subject to similar threats as the original [10]. Further, the sample may skew towards students who enjoy CS (participants were not compensated), perhaps over-estimating student abilities. The small number of students who responded and completed all questions lowers power to detect effects.

## III. Results

We report on four of the style patterns examined in the survey. Our full replication will analyze all seven patterns.

### A. Boolean Returns (with and without relational operators)

We examined two types of Boolean return items: with operators (e.g., ==, <,) and without operators (e.g., calling a method, such as `.equals()`). Most students produced functional code for the code-writing items for these patterns (17/18 and 13/18 students with and without an operator, respectively). On the writing task without an operator, students were asked to write a function that returned `true` when the input String started with "A". Non-functional solutions typically used incorrect syntax for the comparison or compared to a letter other than "A". Most students used the novice pattern: an `if` statement that explicitly returned `true` or `false` (17/18 with operator, 12/18 without operator), supporting the pre-registered hypothesis of the prior and current study: over 20% of our target population (second semester CS students) will use the novice pattern for Boolean returns ($z$-tests, $p < .0001$, with and without operators; $z = 7.9$ and $4.9$, respectively). The true prevalence in this population may be much higher.

Students' readability opinions were consistent with their writing style: a minority of students thought expert style was most readable (four with operator, six without). $z$-tests on this sample support the prior work's hypothesis that over 20% of our target population will select the novice pattern as more readable ($p < .0001$, both with and without operators; Table I). However, most students correctly *identified* expert style (72% with operator, 88% without). For Boolean returns without operators, students are much more likely to agree that expert code is best styled rather than most readable (McNemar test, Bonferroni-corrected $p = .031$; odds ratio is 8).

While most students could recognize expert style, that did not lead to correct editing for style. Code for the editing task *between* checked if the input was less than one number and greater than another. It used two nested `if`-statements and then returned `true` or `false`, using novice patterns for Boolean returns and nested `if`'s instead of `&&`. 13 students did some editing, but only two students combined the two conditions into one and returned it directly (without an `if`). Of these two students who edited correctly, neither used expert style on the corresponding writing task, but both correctly recognized expert style on the corresponding opinion question. It is notable that 11 additional students recognized what expert style was, but did not edit correctly for this topic (including the one student who used expert style on the writing task).

The editing task without operators checked if an input `String` ended with "ed" or "ing" (using a single `if` with an "or" (`||`) conjunction). Five students copied and pasted the original code without any editing; nine changed brackets or whitespace; and four returned the Boolean expression directly. Of the four students who edited correctly, only two used expert style on the corresponding writing task; all recognized expert style on the corresponding opinion question. As in the editing task with an operator, many students (here, 12) recognized expert style and yet did not edit correctly (including four students who used expert style on the writing task).

One unexpected result was that one of the two students who edited correctly on both tasks seemed to over-generalize the pattern. On the editing task targeted at removing special cases, this student also re-wrote all return statements to avoid returning `true` or `false`. However, in this task explicitly returning `true` and `false` was stylistically appropriate, and the change obscured the meaning of the code.

### B. Nested if-statements vs. &&

The writing, readability/style, and comprehension questions addressing nested `if` statements and `&&` required conditions to be ordered in a particular way to avoid exceptions (e.g., checking if a position was within the bounds of an array before examining the element at that position). The writing task asked for a function that takes two `int`s as input and returns the `String` "Ok" if the first input divided by the second is 5 or larger, and the first input is bigger than 10. Otherwise, the function should return the `String` "Not Ok". While an example input showed the function should return "Not Ok" if the second input was 0, students were not explicitly told to avoid dividing by 0, and only one student checked if the second input was 0 before using it as a divisor. Yet, that code was incorrect - it checked if the quotient was larger than 2.

We classified 14 students as using expert style for this task because they used at least two conditions (out of a necessary three for correct functionality) and no more than one `if`-statement; some answers avoided `if`s by returning Booleans with a conjunction instead of a String, and these were marked as expert style. Two students used nested `if` statements, including the one student who checked if the divisor was 0. Two students checked only one condition (in a single `if`).

TABLE I
SURVEY RESULTS: WRITING AND SELECTION OF EXPERT CODE AS MOST READABLE AND BEST STYLED

| Topic | Wrote with Good Style | Wrote Functional Solutions: | | Agreed Expert Code Was: | | | $z$-test |
|---|---|---|---|---|---|---|---|
| | | Good Style | Novice Style | Most Readable | Best Styled | Both | |
| Boolean Returns with Operator | 1 | 1 | 16 | 22% | 72% | 17% | $p < .0001*$ |
| Boolean Returns without Operator | 6 | 4 | 9 | 33% | 88% | 33% | $p < .0001*$ |
| Multiple Cases with General Solution | 18 | 5 | 0 | 61% | 50% | 28% | $p = .055$ |
| Nested `if` statements vs. `&&` | 14 | 0 | 0 | 61% | 61% | 44% | $p = .9$ |

*Bonferroni-corrected $p < .05$ (raw $p$-value $< .007$). $p$-values given above are raw.
$z$-tests (topics above the line): $\geq 20\%$ of the population thinks novice code is more readable than expert code. Below the line: $\leq 20\%$.

While 14/18 students demonstrated that they could use the conjunction `&&` rather than nested `if`s, these students only checked the two conditions where the order didn't matter. These results are consistent with results on the editing task, which presented nested `if` statements where the ordering of the conditions did not matter: all 13 students who edited the code used `&&` to combine the two conditions. Thus, it is unclear whether students could have produced expert style or modified novice code correctly in a situation requiring short-circuit evaluation.

Results from the comprehension questions suggest that students do not understand how the order of conjoined conditions affects execution. Both comprehension questions targeting this pattern required students to identify the output of code blocks based either on inputs they provided or inputs that were given to them. All code blocks did the same task: checking if a letter at a given position in a `String` was greater than some other letter. This task necessitated checking if the given position was in bounds. Two code blocks used nested `if` statements, one in an order that would never throw an exception and one that would (checking the position in the `String` before checking that the position was in bounds), and two code blocks used `&&`, with conditions in the same two orders as for the nested `if` versions. When giving an output for an input that was out of the bounds of the String, the majority of students gave the correct output when the conditions were in an order that wouldn't lead to an exception (12/18 for `&&`, 13/18 for nested `if`s). However, code that would throw an exception led to far more comprehension errors: only 6/18 students answered correctly for the nested `if` version, but this was actually higher than the 1/18 students who responded correctly to the `&&` version of the same code. This suggests that students struggle with understanding ordered conditionals overall, but that they have particular difficulty comprehending expert-style code for conditionals with multiple conditions where the order of the evaluation of these conditions matters.

The code in the readability/style questions included three conditions, two checking that a position was within the bounds of an array and one examining the element at that position. 8/18 students chose the single `if`-statement with conjunctions as both best styled and most readable. Only 11/18 students recognized this expert solutions as best styled, suggesting that recognizing expert style is more difficult for this pattern than for the Boolean returns patterns, although readability and style opinions tend to be more aligned for this pattern.

### C. Multiple Cases vs. General Solution

All students used a single general case for the writing task. However, no student was completely successful on the editing task, which included one necessary special case and two unnecessary special cases. Of the 12/18 students who made at least some edits, only four edited the number of cases: two removed one but not both unnecessary cases, and two removed all special cases, changing the code's functionality.

While no students wrote with multiple cases, 7/18 students thought that unnecessary cases made the sample code more readable. This was the only topic where students were *less* likely to agree with expert opinions for style than readability: 11/18 thought a single general case was most readable, but only 9/18 thought it had the best style. Of the four students who deleted at least one unnecessary case on the editing task, none thought that the expert style was the most readable *and* recognized it as the expert style.

Overall, comprehension was higher on the code written with multiple cases compared to the general solution (92% and 77%, respectively). The comprehension tasks gave three different inputs and asked what the outputs would be. Two inputs targeted special cases in the novice-styled code, and, perhaps unsurprisingly, for these inputs students were more likely to be correct on novice-styled code than expert code (94% vs. 77%). However, students were *also* more likely to be correct with the novice-styled code with the input that was not targeted to a special case (88% vs. 77%). A logistic regression on response variable item correctness, with explanatory variables of input type, code style, and an interaction term for input type * style found a marginal effect for style ($t(102) = 1.94, p = .056$). This trend also held within the 11 students who thought the expert-styled code was most readable.

## IV. DISCUSSION

### A. Comparing Results: Original Study and Pilot Replication

The prior study [10] found that students were in significantly more agreement with expert choices for style than for readability. Our pilot study replicated this finding, with a logistic regression on agreement with expert choices, and explanatory variables for style vs. readability ($t(250) = 4.34, p < .0001$), for topic, and random effect for student. For individual patterns, we replicated some of the prior results but not others. For *Boolean Returns*, we replicated the finding that at least 20% of

the target population would agree that expert-styled code was less readable than novice code and at least 20% would write with novice style on the task with an operator; we also extend these findings to a *Boolean Returns* writing task without an operator. A similar percentage of students across both studies recognized expert style for these patterns (71%-88%).

For the *Multiple Cases* pattern, they [10] also found that at least 20% of the target population would see expert-styled code as less readable than novice code. Our results do not contradict those findings (39% of our sample agreed that extraneous cases improved readability), but they did not reach statistical significance. In the original study [10], comprehension scores were higher when the code included extraneous cases compared to a single general solution (81% vs. 72%). Our results also show a marginal trend in this direction ($p = .055$). We expect the full replication to replicate both results from the original study.

For *Nested ifs vs. &&*, students in our survey were less likely to find the expert-style more readable (61% here versus 90% in the prior study, [10]). However, our survey used more complex code for this pattern, relying on short-circuit evaluation to avoid exceptions. Future work should compare expert and novice styles for this more complex code versus code where the ordering of the comparisons doesn't matter.

Wiese, Rafferty, and Fox [10] found that readability preferences were predictive of writing style for the one writing task on the survey. We did not replicate that finding on either the *Boolean Returns* tasks or the *Nested ifs vs &&* task. For *Boolean Returns*, this non-replication is likely due to our small sample size and the small percentage of students writing with expert style. For the *Nested ifs vs &&* writing tasks, 17/18 students neglected a key condition in the prompt, writing code that did not rely on short-circuit evaluation to avoid an exception while judging readability on code that did. Further, our results suggest that this pattern will not hold for *Multiple Cases*, where no students used novice style on the writing task, but 39% preferred it for readability.

### B. Lessons Learned

Recognizing good style was not sufficient for making correct style edits. Exposure to good style and learning to recognize it may not be enough for students to actually code well. We must explore instruction to address this gap. Further, style is not a binary skill that students either have or don't: it is comprised of sub-skills, and students' mastery of each can vary. Consider the student who avoided all `return true/false` statements in editing tasks. This student used novice style on the corresponding writing task, but edited correctly on both *Boolean Returns* editing tasks, suggesting the student learned from the survey. While encouraging that students may improve coding style from a brief style exercise, this shows that using good style is not simple. Novices may need to be taught the specific contexts in which an expert style pattern applies.

### C. Plan for Full Replication

We plan to do a full replication and extension of the original work [10]. Some instructors at the liberal arts college do not allow extra credit, so in our full replication we will pay participants to ensure a large enough sample size.

While the prior study [10] suggested a strong correlation between how students write and what style they find most readable, this relationship was based on one writing task. Our full replication, like our pilot, will include writing tasks for more patterns. Our initial results indicate that writing choices don't always predict readability preferences. No students wrote with multiple cases, but 39% preferred to read it. However, students may have been reacting to the different functionalities of the code they were writing vs. reading. To examine this relationship more closely, our full replication will counterbalance the task (writing, opinions on style/readability, comprehension, and editing) with the particular coding problem. This way, we can be sure that overall differences in students' responses are due to the task and the code style, not the functionality. Further, while 14/18 used `&&` to conjoin Booleans (instead of using nested `if`s), none of these students checked the inputs to make sure their code did not divide by 0. Our full replication will include a reminder to avoid throwing this exception, so we can examine writing style when the order of the conditions matters.

Finally, Wiese, Rafferty, and Fox [10] proposed that since most novices could recognize expert style, they should be able to improve their novice-styled code when prompted. However, in this study, many students who recognized expert style for a given pattern did not edit correctly for that pattern. It may be harder for students to edit someone else's code than to edit their own. In our full replication, we will automatically check for some style errors, and ask students to revise if they make those errors on the writing tasks. To investigate how much help students need to edit their own code, we will offer a hint if students cannot revise correctly on their own, and we will offer a worked example if students do not understand the hint.

### D. Conclusion

A full replication of the initial study [10] with our additional writing, editing, and revision items will empirically show which novice style errors are the most common, and which indicate problems with comprehension. This work will help instructors design effective lessons and assignments, including those that focus on code reading and editing (not just code writing). In our population of students with less Java experience than those in the original study [10], and who are less likely to be as attuned to style concerns, we found that exposure to good style in code examples is not sufficient for many students to distinguish good style from poor style for all style issues. As in the prior study, [10], we also find that when students can identify expert style code, they do not necessarily believe it is more readable. Our range of tasks show that style knowledge is multi-faceted, and that students do not acquire the skills of recognition, usage, comprehension, and editing at the same time. Untangling the relationships between these skills is important for creating effective style instruction.

## REFERENCES


[1] S.-N. A. Joni and E. Soloway, "But My Program Runs! Discourse Rules for Novice Programmers," *Journal of Educational Computing Research*, vol. 2, no. 1, pp. 95–125, 1986.

[2] E. Soloway and K. Ehrlich, "Empirical Studies of Programming Knowledge," *IEEE Transactions on Software Engineering*, vol. SE-10, no. 5, pp. 595–609, 1984.

[3] P. W. Oman and C. R. Cook, "A Taxonomy for Programming Style," in *Proceedings of the 1990 ACM Annual Conference on Cooperation CSC '90*, Washington, DC, 1990, pp. 244–250.

[4] S. C. B. de Souza, N. Anquetil, and K. M. de Oliveira, "A study of the documentation essential to software maintenance," in *Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting & Designing for Pervasive Information*, ser. SIGDOC '05. New York, NY, USA: ACM, 2005, pp. 68–75. [Online]. Available: http://doi.acm.org/10.1145/1085313.1085331

[5] E. L. Glassman, L. Fischer, J. Scott, and R. C. Miller, "Foobaz: Variable Name Feedback for Student Code at Scale," in *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology - UIST '15*, 2015, pp. 609–617.

[6] E. S. Wiese, M. Yen, A. Chen, L. A. Santos, and A. Fox, "Teaching Students to Recognize and Implement Good Coding Style," in *Proceedings of the 4th ACM conference on Learning at Scale*. Cambridge, MA: ACM, 2017, pp. 41–50.

[7] C. Hundhausen, A. Agrawal, D. Fairbrother, and M. Trevisan, "Integrating pedagogical code reviews into a cs 1 course: An empirical study," in *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '09. New York, NY, USA: ACM, 2009, pp. 291–295. [Online]. Available: http://doi.acm.org/10.1145/1508865.1508972

[8] M. Woodley and S. N. Kamin, "Programming studio: A course for improving programming skills in undergraduates," in *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '07. New York, NY, USA: ACM, 2007, pp. 531–535. [Online]. Available: http://doi.acm.org/10.1145/1227310.1227490

[9] J. Whalley, T. Clear, P. Robbins, and E. Thompson, "Salient elements in novice solutions to code writing problems," *Conferences in Research and Practice in Information Technology Series*, vol. 114, pp. 37–45, 2011.

[10] E. S. Wiese, A. N. Rafferty, and A. Fox, "Linking Code Readability, Structure, and Comprehension among Novices: It's Complicated," in *Proceedings of the 41st ACM/IEEE International Conference on Software Engineering*, 2019.