2023 Annual Conference & Exposition
Baltimore Convention Center, MD | June 25 - 28, 2023
The Harbor of Engineering Education for 130 Years
ASEE
Paper ID #36963

Board 196: A Framework to Assess Debugging Skills for Computational Thinking in Science and Engineering

Derrick Hylton, Spelman College
Dr. Shannon Hsianghan-huang Sung, Institute for Future Intelligence

Shannon H. Sung is a Learning Scientist at Institute for Future Intelligence. Her research focuses on technology-enhanced learning and assessment, interdisciplinary STEM learning, and the cognitive learning processes.

Xiaotong Ding
Mary Johanna Van Vleet

A Framework to Assess Debugging Skills for Computational Thinking in Science and Engineering

Abstract

A rubric is presented to assess debugging skills for students particularly in the natural sciences and engineering. The three categories that are assessed for the cognitive processes in debugging skills are identification, isolation, and iteration. These are defined, and the characteristics of each process are listed. We discuss the method used to develop this rubric that was based on intentional errors in a programming assignment given to students in an introductory physics course. The programming in this assignment was in Python and a visual-based programming platform, called iFlow. We believe that visual-based programming will help elicit weaknesses in debugging because it removes students' familiarity with particular programming languages.

Our focus on debugging skills came from a survey of students to self-identify barriers in computational work in an introductory physics course that included engineering majors. This skill was the primary self-identified barrier along with abstraction skills, which will be the focus of another work. We also present the results of this survey. The Python assignment ($n_{text} = 9$) was used to create the rubric and the iFlow assignment ($n_{graphic} = 11$) was used to test the rubric. Scoring was based on a scale of six levels in each category. Although the sample size was too small to establish rigorous scoring reliability, we discussed how the two researchers attained agreement in scoring the assignments after iterative modifications of the rubric and rescoring. For the Python assignment, the average for identification was 2.75/5, for isolation 2.30/5, and for iteration 3.33/5. For the iFlow assignment, the average for identification was 2.63/5, for isolation 2.23/5, and for iterate 3.32/5. A consistent trend from these assignments showed that students' approach to debugging is mainly to identify and iterate without a full understanding of the error (i.e., isolation). The lack of a full understanding of the error implies that students are prone to repeat the error. Thus, the important outcome of debugging is to understand the source of error by systematically investigating different parts of the computational solution. Our preliminary results led to the hypothesis that students with weak debugging skills are mainly due the isolation process. This hypothesis will be tested in a future experiment. Results from such an experiment will be significant to those who are designing intervention strategies to integrate computational thinking in science and engineering curricula.

Background

In STEM education, computational thinking (CT) has become a critical component in preparing students for the technical workforce [1]. Computation is fundamental to science because it renders rich contexts for solving complex problems in the real world. The overall goal of this project is to equip practitioners with the ability to enhance students' computational skills in STEM courses, especially in introductory courses. In order to do this, we must identify barriers, develop specific assessments, and create intervention activities to improve CT skills.

Practitioners are less familiar with the integration and assessment of CT in STEM curricula [2]. Also, very few CT assessment studies have been applied to higher education, and the CT literature is especially lacking in the STEM field, or mostly focuses on assessing overall CT. Although focusing on assessing overall CT is beneficial, it does not allow a practitioner to pinpoint the specific weakness of a student. Therefore, just-in-time and strategic interventions may not be feasible.

The definition of CT in the literature, although divergent, entails common themes (e.g., [3], [4], [5]) which we have coalesced into five practices: *abstraction, decomposition, algorithmic thinking, debugging,* and *generalization*. For more detailed information of the aforementioned practices, see Martínez and his colleagues [6]. In this paper, we summarize our preliminary work in debugging.

Our focus on debugging skills came from a survey of students to self-identify barriers in computational work in an introductory physics course that included engineering majors. This skill was the primary self-identified barrier along with abstraction skills, which will be the focus of another work. Our objective is to identify cognitive processes and practices associated with debugging computational solutions in STEM and develop a framework using undergraduate students' artifacts.

Methods

Operational Definition of Debugging

We adopted Weintrop and his colleagues' definition of troubleshooting and debugging definition, which states "*Students who have mastered this practice will be able to identify, isolate, reproduce, and ultimately correct unexpected problems encountered when working on a problem, and do so in a systematic, efficient manner*." ([5], p.140) Our team operationally defined 3 cognitive processes of debugging that are most relevant in STEM education – identification (making sense of the solution), isolation (investigating the cause of an error), and iteration (repeatedly improving the solution); see the Table in the Appendix for complete characteristics for each process.

Key practices for each cognitive process were listed and student responses according to their complexity levels were categorized into 6 levels. This approach was inspired by the Knowledge Integration framework, where Lynn [7], Liu [8], and their colleagues listed essential science concepts and categorized student's conceptual understanding into various levels according to the number of connected key concepts. These characteristics were defined as distinct as possible to ease scoring purposes.

Context of Study

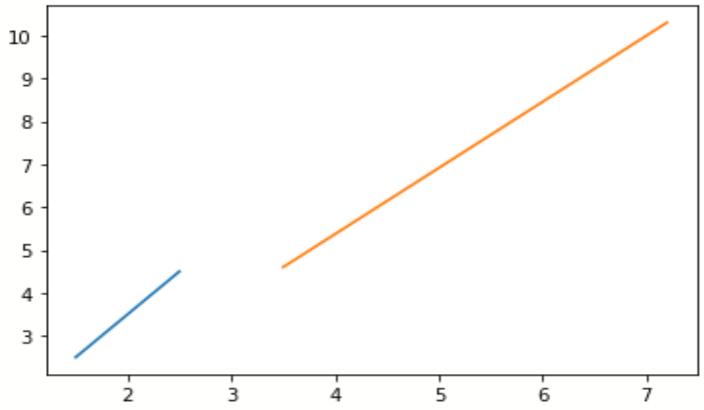
To test whether the assessment framework could be used in different programming platforms, we designed two parallel debugging assignments, one in text-based (see Box 1) and the other in graphic-based (see Box 2) formats. Both questions are program-based computational problem, where students have to identify that the output is not the most ideal solution, to isolate which input codes are needed to be corrected, and to iterate the investigation to fix the error. Twenty students (n_{text} = 9, $n_{graphic}$ = 11) from the laboratory component of a calculus-based introductory physics course consented to participate in this study. Four think-aloud interviews were conducted to ensure that the questions were eliciting the desirable debugging practices under study.

Box 1

Sample text-based debugging question.

We write a code to plot the points (1.5, 2.5), (2.5, 4.5), (3.5, 7.2) and (4.6, 10.3), as follows:

```
import matplotlib.pyplot as plt
point1 = (1.5, 2.5)
point2 = (2.5, 4.5)
point3 = (3.5, 7.2)
point4 = (4.6, 10.3)
plt.plot(point1, point2, point3, point4)
```



a. Is the output of the plot function as expected? Explain in the space below.

b. Python allows you to check the type of data for a particular variable by using the built-in function 'type'. The syntax is: type(x), where the argument x is the name of the variable you want to check. Use this to consider inputs and output of the plot function to isolate the problem. In the space below state what you did with the 'type' function, if anything.

c. Try to fix the problem. In the table below, write each thing that you tried along with your reasoning, even if it did not work.

d. What might have been the thought process of writing the code as shown?

Box 2

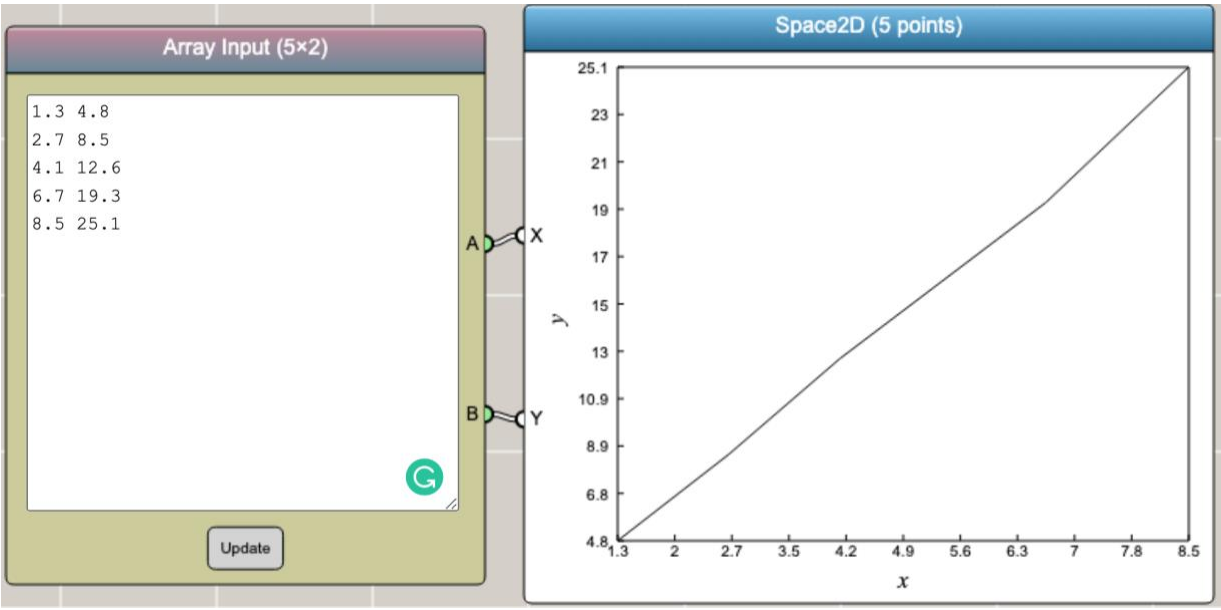
Sample graphic-based debugging question

Graphic-based prompt:

Assume that you have collected data of one quantity (called y) as you vary another quantity (called x). You want to plot this data so that you can visualize any trend in the data. The data points (x, y) are: (1.3, 4.8), (2.7, 8.5), (4.1, 12.6), (6.7, 19.3), and (8.5, 25.1) with appropriate units. This data has been entered into an Array Input block in iFlow for you. The link below will take you to this iFlow file. Click on the link or copy and paste it in a Chrome browser; sign in; and copy this file either in the Clouds or on your local computer as a .json file. (iFlow works better with Chrome.)

http://intofuture.org/iflow/index.html?userid=shannon%40intofuture.org&filename=ssplot1

To plot this data, we use a block called Space2D that can be found in the Inputs and Outputs category of blocks. Drag a Space2D block to the work area and resize it. Connect the output node of the Array Input block to both input nodes of the Space2D block as an attempt to plot the data.



a. Is there a problem with the final graphical output? Explain.

b. Fill in the table below in describing each action you did in trying to fix the problem:

List of things done	**Reasoning for each step**	**State what you learned from each step**

Refining Debugging Rubrics

Two of the researchers coded both Python and iFlow questions together to establish interrater agreement. Although the sample size was too small to establish rigorous scoring reliability, the two researchers obtained agreement in scoring the assignments after iterative modifications of the rubric and rescoring them after exhaustive discussions. We also took notes based on the recurring mistakes found in their responses as feedback to revise future questions.

Results

The final debugging rubrics are divided into three cognitive processes along with expected practices and 5 complexity levels for each process (see Appendix). The Python assignment was used to create the rubric and the iFlow assignment was used to test the rubric. Scoring was based on five levels in each category. For the Python assignment, the average for identification was 2.75/5, for isolation 2.30/5, and for iteration 3.33/5. For the iFlow assignment, the average for identification was 2.63/5, for isolation 2.23/5, and for iterate 3.32/5. A consistent trend from these assignments showed that students' approach to debugging is mainly to identify and iterate without a full understanding of the error (i.e., isolation). For example, one student's response to the text-based prompt in Box 1 as an example of intermediate identification:

> *I thought it would just produce a picture of one plot line but it also spit out a discontinuous line. (identification: 2).*

The student showed an attempt to examine the output, however, the response was incorrect because the problem asked to plot the points and the student was focusing on the line rather than the points as was prompted.

Although there was an attempt to identify an issue, there was no investigation as to why the line was discontinuous as the following response shows:

> *I put each position variable into the type function to make sure there were no errors. c)I tried to make a line of best fit by using outside resources for tips. I changed the x and y values to be listed. I then calculated a slope and made a plot line. I tried this because I was unsure of how to make the original code work. It was running perfectly, the line was just broken. It worked, and produced one single line of best fit. d) I think the code was written that way to put the x and y points together instead of having the code do it automatically. But, since the points were plotted individually with two lines of best fit, the line had a significant break. (isolation: 0)*

This student tried to artificially place a continuous line to replace the discontinuous line and did not attempt to investigate why the line was discontinuous in the first place. In order for students to receive a Level 5 score on isolation, they must complete all three practices: 1) Decide which part of the computational solution is a reasonable cause of the error; 2) Provide the correct reasoning as to what about the part in #1 could be the cause the error; 3) Systematically study how the reasoning given could have caused the error. Notice this score was not used to penalize the lack of identification in Question a), but to assess their practices for isolation.

Discussions

Our preliminary results led us to hypothesize that students with weak debugging skills were mainly due to the isolation process, since this process had the lowest score. The finding implies that, even if students can quickly identify and fix the error in the program, they are not as proficient in isolating and investigating possible causes of error, which is an essential practice to acquire better understanding of computational problems. Without a full understanding of the error, students are prone to repeat the error. Thus, the important outcome of debugging is not just to fix the error, but also to understand the source of error by systematically investigating different parts of the computational solution.

Results from this study will be significant to those who are designing intervention strategies to integrate computational thinking in science and engineering curricula. For instance, it shows that practitioners should focus most of their efforts on teaching the isolation process instead of spending a lot of time on identifying or iterating; the focus should be on investigating the source of error. Even though this study focused on the program-level debugging, the process can be readily applied to problem-level debugging. Problem-level debugging is an analysis of the solution based on the non-programming parts, as not all errors occur in the programming part. For example, some errors may occur in the assumptions and modelling parts of the solution.

Future Plans

A similar approach will be taken in the future to other practices in CT, such as for algorithmic thinking and abstraction. We would also like to research interconnectedness of the different CT practices, such as the relationship among debugging, abstraction, and algorithmic thinking, and what it means in terms of assessment. Our hypothesis that students are weak in isolation will be tested in future studies that include problem-level CT.

Acknowledgements

This work is funded under the NSF HBCU-UP Broadening Participation Research Program in STEM Education under award number 2107104

References

[1] "Dear Colleague Letter: Advancing Educational Innovations that Motivate and Prepare PreK-12 Learners for Computationally-Intensive Industries of the Future (nsf20101) *National Science Foundation*." https://www.nsf.gov/pubs/2020/nsf20101/nsf20101.jsp (accessed Feb. 13, 2022).

[2] C. Wang, J. Shen, and J. Chao, "Integrating Computational Thinking in STEM Education: A Literature Review," *International Journal of Science & Mathematics Education*, vol. 20, no. 8, pp. 1949–1972, Dec. 2022, doi: 10.1007/s10763-021-10227-5.

[3] X. Tang, Y. Yin, Q. Lin, R. Hadad, and X. Zhai, "Assessing computational thinking: A systematic review of empirical studies," *Computers & Education*, vol. 148, p. 103798, Apr. 2020, doi: 10.1016/j.compedu.2019.103798.

[4] Y. Yin, R. Hadad, X. Tang, and Q. Lin, “Improving and Assessing Computational Thinking in Maker Activities: the Integration with Physics and Engineering Learning,” *J Sci Educ Technol*, vol. 29, no. 2, pp. 189–214, Apr. 2020, doi: 10.1007/s10956-019-09794-8.
[5] D. Weintrop *et al.*, “Defining computational thinking for mathematics and science classrooms,” *J Sci Educ Technol*, vol. 25, no. 1, pp. 127–147, Feb. 2016, doi: 10.1007/s10956-015-9581-5.
[6] M. L. Martínez, O. Lévêque, I. Benítez, C. Hardebolle, and J. D. Zufferey, “Assessing Computational Thinking: Development and Validation of the Algorithmic Thinking Test for Adults,” *Journal of Educational Computing Research*, vol. 60, no. 6, pp. 1436–1463, Oct. 2022, doi: 10.1177/07356331211057819.
[7] M. C. Linn, H.-S. Lee, R. Tinker, F. Husic, and J. L. Chiu, “Teaching and Assessing Knowledge Integration in Science,” *Science*, vol. 313, no. 5790, pp. 1049–1050, 2006, Accessed: Feb. 13, 2023. [Online]. Available: https://www.jstor.org/stable/3847060
[8] O. L. Liu, H.-S. Lee, C. Hofstetter, and M. C. Linn, “Assessing Knowledge Integration in Science: Construct, Measures, and Evidence,” *Educational Assessment*, vol. 13, no. 1, pp. 33–55, Mar. 2008, doi: 10.1080/10627190801968224.

Appendix

Table

Assessment Framework for Debugging of Computational Solutions

Categories	**Response Characteristics/Practices**	**Level 0**	**Level 1**	**Level 2**	**Level 3**	**Level 4**	**Level 5**
Identification process: making sense of the solution	Compare the output with existing schema to make sense. 1. Identify a correct issue based on the solution/output. 2. explain why is it an issue.	no/ irrelevant response	show some attempt to examine the output but conclude that there is *no issue*	conclude that there is a problem but the issue identified is irrelevant or incorrect w/ or w/o explanation	correctly identify the issue w/ *incorrect or w/o explanation*	correctly identify the issue with *partially* correct explanation	correctly identify the issue with correct explanation
Isolation process: investigating the cause of an error	Understand the source of an error by systematically investigating different parts of the computational solution. 1. Decide which part of the computational solution is a reasonable cause of the error. 2. Provide the correct reasoning as to what about the part in #1 could be the cause the error. 3. Systematically study how the reasoning given could have caused the error.	no/ irrelevant response	Attempted isolation: Attempt(s) of the practice(es) is/are incorrect.	Simple isolation: Attempt practices with only 1 correct practice.	Partial isolation: Engage in 2 out of the 3 practices correctly.	Systematic isolation: Engage in #1 and #2 correctly and an attempt to #3	Complete isolation: Engage in all 3 practices correctly and a complete undertanding of the error.
Iteration process: repeatedly improving the solution	Employ strategies repeatedly to improve the solution. 1. Examine a change that affects the solution. 2. Change to get an improved solution. 3. **Iterate** (if necessary) to address ALL issues with the solution to get the **best solution**	no/ irrelevant response	Attempt(s) at the practice(es) is/are incorrect.	Attempt practices with only 1 correct practice.	Engage in #1 and #2 but did NOT get the best solution for any issue as stated in #3.	Engage in #1 and #2 correctly and solve SOME of the issues stated in #3 or *did not complete all the iterations.*	Engage in all 3 practices correctly and follow through #3 iterations completely.