Using a Visual-Based Coding Platform to Assess Computational Thinking Skills in Introductory Physics

Derrick Hylton, Spelman College, dhylton@spelman.edu
Shannon Sung, Institute for Future Intelligence, shannon@intofuture.org
Xiaotong Ding, Institute for Future Intelligence, xiaotong@intofuture.org

Abstract: Developing assessment tools for computational thinking (CT) in STEM education is a precursor for science teachers to effectively integrate intervention strategies for CT practices. One problem to assessing CT skills is students' varying familiarity with different programming languages and platforms. A text-neutral, open-source platform called iFlow, is capable of addressing this issue. Specifically, this innovative technology has been adopted to elicit underrepresented undergraduate students' debugging skills. We present how the visual-based coding platform can be applied to bypass programming language bias in assessing CT. In this preliminary study, we discuss design principles of a visual-based platform to effectively assess debugging practices – identification, isolation, and iteration – with the use of iFlow assignments. Our findings suggest how the ability of iFlow to test parts of a program independently, dataflow connectivity, and equity in removing biases from students' various backgrounds are advantageous over text-based platforms.

Introduction

In STEM education, computational thinking (CT) has become a critical component in preparing students for the technical workforce (NSF, 2020). Practitioners are, however, facing problems integrating and assessing CT in the STEM curriculum (Wang et al., 2021). CT presents an additional stumbling block for underrepresented groups in building the pipeline for computational-related careers (Thomas et al., 2018). Our central proposed intervention strategy is for students to create computational solutions to science problems without, at first, requiring them to master intimidating syntax and semantics. This can be accomplished with the aid of a visual-based programming tool, so that we can make students' computational thinking visible while they are shaping it (Ainsworth et al., 2011). We collaborated with the developers of one such tool, called iFlow, where we can create a beta version suitable for our needs. The goal is to design assessment tools for evaluating the effectiveness of visual-based programming in facilitating students' CT skills. We report on one of the assignment questions for eliciting CT skills of debugging, which students self-identified as a barrier in our preliminary study (Hylton et al., 2021).

Innovative visual-based programming platform – iFlow

Visual-based programming was implemented by the application of iFlow. Inspired by the Unified Modeling Language and dataflow programming paradigm, this constructionist environment (Papert, 1991), named iFlow, models a program as an executable directed graph depicting the structure of a computational solution and the interactions among its constituents. The results emerge as data flow through these interconnected elements (see Figure 1 as an example). While there exist successful dataflow programming products such as Grasshopper, LabVIEW, and Simulink, most of them are tailor-made for specific applications that may not be appropriate for introductory courses. For example, Grasshopper only works within the Rhinoceros 3D computer-aided design software, LabVIEW is mostly used in data acquisition and instrument control, and Simulink focuses on modeling multi-domain dynamic systems. By comparison, iFlow is a general-purpose, Web-based, and integrated computational environment designed for students to solve common problems encountered in the science curriculum, with an objective to meet diverse educational needs of students with various backgrounds and interests in science. To clarify, although it bears some resemblance to system dynamics software such as Stella and Vensim, iFlow is not a system dynamics modeler. In iFlow, the representational blocks are directly manipulable (e.g., pulling a slider changes the variable it represents). Their changes can be immediately transmitted across the connector networks, updating the linked nodes on their way and making the entire diagram interactive

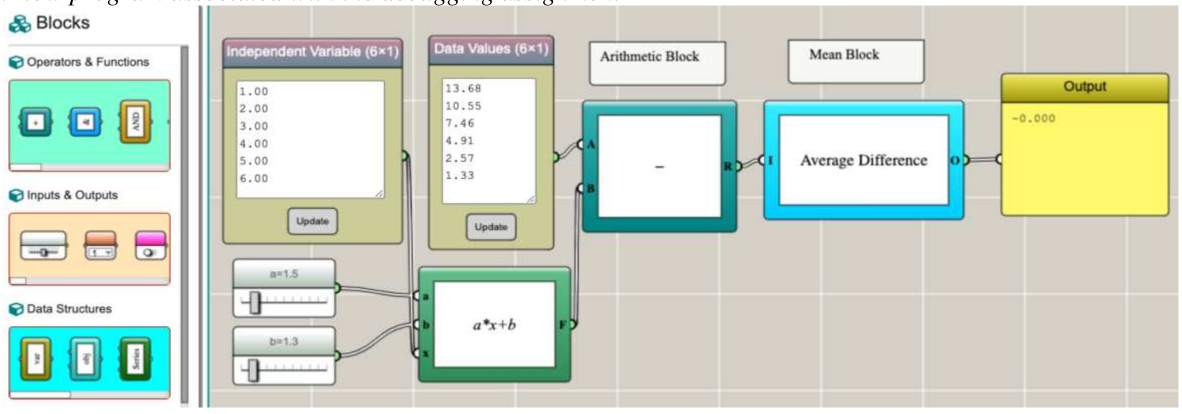
iFlow debugging assignment

We describe an assignment intended to elicit and assess users' debugging skills. The assignment prompt is:

Your lab partner devised a method to test whether the function shown in the *Multivariate Function block* (the green block in Figure 1) is a good fit to the data. S/he decided to subtract the data values from the function values (using an *Arithmetic block* [see Figure 1]) to see how close they are as a test of goodness of fit. S/he also decided to average the difference of each data point (using the *Mean block*), and since the average is 0 as shown in the *Output block*, s/he decided that the fit is very good. Your job is to decide whether you agree or disagree with your lab partner's work, and if you disagree to improve on the work.

The assignment in Figure 1 illustrates some of the blocks where we can store arrays, define functions, assign variables, do arithmetic computations and statistics, and display outputs. The assignment program was constructed by dragging appropriate blocks from the left "Blocks" palette to the working canvas. The function of each block was described to the student via a manual and training exercise. The blocks are connected via the nodes on the left and/or right sides of the blocks, which represent the input(s) to and output(s) from the blocks. Each block has properties that are listed in table form (by right clicking on the block) that can control each block. The assignment was given to 13 STEM majors in an Introductory Physics lab. A post-activity survey was implemented to collect students' reflection and feedback.

Figure 1
iFlow program associated with the debugging assignment



```
Independent Variable (6×1): 1.00 2.00 3.00 4.00 5.00 6.00
Data Values (6×1): 13.68 10.55 7.46 4.91 2.57 1.33
Output: -0.000
```

We hypothesize that this visual-based programming can enhance students' cognitive processing of data flow, which would lead to a stronger performance in more traditional computing tools, such as text-based Python and visual-based LabView. This question can be used to assess debugging skills, because it purposefully introduces an error in testing the goodness of fit, which is an essential part of common data analysis in introductory physics courses. The error introduced is averaging the deviations to gauge goodness of fit. This will not work since some deviations are positive and some are negative. Students with more advanced debugging skills should be able to graph the data and fit function with iFlow (see Figure 2) to realize that the fit is not good.

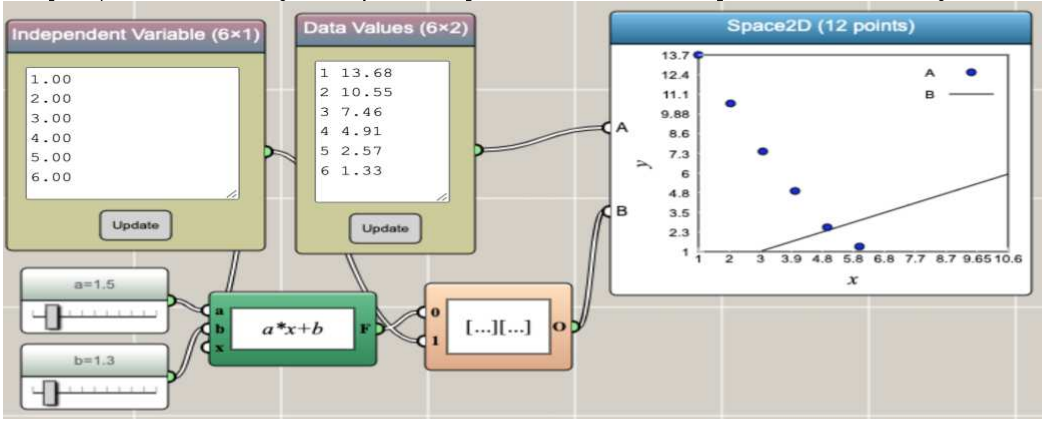
Assessing cognitive processes of debugging in iFlow

iFlow program makes the assessment of debugging practices easier, because it could not only help assess the processes involved in fixing errors, but also resurface the cognitive processes of completely understanding the errors. Such affordance could mitigate the problem, where errors are often fixed without systematically investigating them, and thus, learners are prone to repeat the errors (Li et al., 2019).

We classify the debugging practices dealing with errors into three cognitive processes—identification, isolation, and iteration – adapted from the work of Weintrop and his colleagues (2016). *Identification* focuses on how one makes sense of the solution, *isolation* focuses on one's systematic investigation of the issue, and *iteration* focuses on how one reproduces and fixes the error. To illustrate each cognitive process, we prompted the students with the following questions as a protocol in eliciting thought processes: 1) *Do you AGREE with the given solution? 2) If Yes, explain your reasoning in the space below. 3) If No, you should improve on your solution. 4) Record ALL the things you did to justify your decision and to improve the solution. These things should include anything you did to acquire information needed to understand ALL the issues you identified. For example, if you tried to understand the properties of a block outside of the computational problem, then it should be included in the list of things done.*

For numbers 1-3, we expect students to spot some issues in the solution shown in Figure 1. For example, the deviation (computed by the arithmetic block) can average to 0 as shown in the output block, but with further investigation, a graph generated by an exemplar student (see the Space2D block in Figure 2) reveal that the solution proposed in Figure 1 is inadequate. For number 4, students are prompted to list their actions, so that we can evaluate whether they engaged in the investigative processes. For instance, students should research the error enough to eliminate the canceling of the deviations by comparing the data values (shown as the blue points in Figure 2) to the line fitted to the data (shown as B in the legend of Space2D block in Figure 2). Another issue is the type of function, and students should realize by the practices involved in *isolation* and *iteration* that the linear function is not the best solution.

Figure 2
Sample systematic investigation of an exemplar student on the computational errors in goodness of fit.



```
1.00
2.00
3.00
4.00
5.00
6.00
```

```
1 13.68
2 10.55
3 7.46
4 4.91
5 2.57
6 1.33
```

Design principles of iFlow in assessing debugging

Assessing cognitive processes of debugging can be done in common text-based programming, but visual-based programming has some advantages over text-based programming (Navarro-prieto & Cañas, 2001; Saito et al., 2017; Weintrop & Wilensky, 2017) and assessment. First, iFlow helps its users to investigate blocks independent of the rest of the program, which usually cannot be readily done in text-based programming (e.g., usually, one has to disable the rest of the program in order to check one part). For assessment purposes, iFlow makes it easier for students to isolate and trace back the steps without breaking the entirety of the programming solution (e.g., Figure 2). For researchers, iFlow is advantageous over text-based programming, because we can evaluate what the tendencies of users are when they engage in debugging, such as: 1) identifying an issue based on the visual computational solution, 2) deciding which part of the visual-based computational solution is a reasonable cause of the error, and 3) systematically studying how the cause of error. Therefore, iFlow better helps the practitioners and researchers evaluate investigative processes pertaining to how students isolate errors and fix them.

Second, we believe that iFlow encourages debugging because it visually shows connectivity and relationship among different components on the same page. The hierarchical linearity or non-connectivity involved in the text-based programming usually overwhelms novice students, preventing them from even attempting to identify the issue (Mosemann & Wiedenbeck, 2001), which deters any assessment endeavor. In visual-based programming, students can trace the source of error and test various ideas in a more systematic manner. Correspondingly, the inputs of text-based codes could be assigned hundreds of lines back in the program before being called into computing. Such a gap and lack of connectivity may demotivate students in science classes to engage in the practices of debugging. As a result, iFlow is more suitable than text-based platforms in assessing debugging, since the interface is more welcoming for users to engage in debugging practices.

Third, all of the students can be provided with an equal background in iFlow, which may not be possible if the same platform, such as Python or Excel, is used in the science courses for assessment purposes as well as coursework. The coursework platform for assessment may impose bias since students would have varying backgrounds and familiarity. For instance, we may not be able to assess the cognitive processes in *isolation* correctly if some of the students already know how to deal with the purposeful errors. These students can bypass some of the debugging sub-practices, which makes their prior knowledge interfere with the assessment. Thus, the validity of the assessment is jeopardized.

Discussions and implications

In this technology innovation paper, we only presented the application of iFlow to the assessment of debugging skills. Based on students' positive feedback, such as "*The biggest thing is that iFlow is the most straightforward [compared with Python]. If there's anything that you're not understanding, you are able to just see it.*" we conclude that visual-based programming would be efficient in assessing debugging skills, such as the ability to see the various connections shown on iFlow interface. A few of the advantages are: ability to test parts of a program independently, dataflow connectivity, and equity in removing biases from underrepresented students' various backgrounds. Visual-based programming, such as iFlow (Hylton et al., 2021), is an innovative technology to teach computational thinking in STEM courses. It is a powerful platform not only for students to be able to solve computational problems in science, but also for researchers to assess pupils' learning in various aspects.

The assignment developed for assessing debugging skills can be applied to both text-based and visual-based programming. In our future studies, we plan on building debugging skills of students in STEM courses by an appropriate intervention whose effectiveness can be assessed and refined using appropriate debugging rubrics.

Nevertheless, with the promising affordance in assessing CT in STEM education, there are some limitations to this innovative technology. For instance, iFlow requires more development, as is planned, so it is more responsive to the various needs and provides a smoother interface, such as comparing similar computational solutions side by side without the need to open multiple cloud files. In addition, because the interface is different from conventional text-based programming, it requires constant customization of different manuals and user training materials that are tailored for instructional use.

References

Ainsworth, S., Prain, V., & Tytler, R. (2011). Drawing to Learn in Science. *Science, 333*(6046), 1096-1097. doi:10.1126/science.1204153

Hylton, D., Sung, S., & Xie, C. (2021) Adopting no-code methods to visualize computational thinking. Rodrigo, M. M. T. et al. (Eds.) (2021). *Proceedings of the 29th International Conference on Computers in Education. Asia-Pacific Society for Computers in Education*, 79-84.

Li, C., Chan, E., Denny, P., Luxton-Reilly, A., & Tempero, E. (2019). Towards a Framework for Teaching Debugging. *Proceedings of the Twenty-First Australasian Computing Education Conference*, 79–86. https://doi.org/10.1145/3286960.3286970

Mosemann, R., & Wiedenbeck, S. (2001). Navigation and comprehension of programs by novice programmers. In *Proceedings 9th International Workshop on Program Comprehension. IWPC 2001* (pp. 79–88). IEEE.

Navarro-Prieto, R., & Cañas, J. J. (2001). Are visual programming languages better? The role of imagery in program comprehension. *International Journal of Human-Computer Studies*, *54*(6), 799–829. https://doi.org/10.1006/ijhc.2000.0465

National Science Foundation (NSF) 2020. Dear Colleague Letter retrieved from https://www.nsf.gov/pubs/2020/nsf20101/nsf20101.jsp

Papert, S. (1991). Situating Constructionism. In I. Harel & S. Papert (Eds.), Constructionism. Norwood, NJ: Ablex Publishing Corporation.

Saito, D., Washizaki, H., & Fukazawa, Y. (2017). Comparison of Text-Based and Visual-Based Programming Input Methods for First-Time Learners. *Journal of Information Technology Education: Research*, *16*, 209–226. https://www.informingscience.org/Publications/3775

Schmidgall, S. P., Eitel, A., & Scheiter, K. (2019). Why do learners who draw perform well? Investigating the role of visualization, generation and externalization in learner-generated drawing. Learning and Instruction, 60, 138-153. doi:https://doi.org/10.1016/j.learninstruc.2018.01.006

Thomas, J. O., Joseph, N., Williams, A., Crum, C., & Burge, J. (2018). Speaking Truth to Power: Exploring the Intersectional Experiences of Black Women in Computing. 2018 Research on Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT), 1–8. https://doi.org/10.1109/RESPECT.2018.8491718

Wang, C., Shen, J., & Chao, J. (2021). Integrating Computational Thinking in STEM Education: A Literature Review. *International Journal of Science and Mathematics Education*. https://doi.org/10.1007/s10763-021-10227-5

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology*, *25*(1), 127–147. https://doi.org/10.1007/s10956-015-9581-5

Weintrop, D., & Wilensky, U. (2017). Comparing Block-Based and Text-Based Programming in High School Computer Science Classrooms. *ACM Transactions on Computing Education (TOCE)*, *18*(1), 1–25. https://doi.org/10.1145/3089799

Acknowledgments

This work is funded under the NSF HBCU-UP Broadening Participation Research Program in STEM Education under award number 2107104. We thank Charles Xie for developing iFlow platform.