A Misconception Driven Student Model to Author Feedback

Luke Gusukuma
Computer Science
Virginia Tech
Blacksburg, Virginia
lukesg08@vt.edu

ABSTRACT

Getting novice programmers over initial misconceptions is difficult because learning programming is difficult. Practice is one of the best ways for novices to learn. However, in the absence of feedback contextualized to instruction and focused on misconceptions, misconceptions become a difficult hurdle. To improve feedback, I present the Misconception-Driven Student Model (MDSM). MDSM is a cognitive model that lends itself to a framework to scalably deliver Misconception-Driven Feedback (MDF). I show MDF's impact through a quasi-experimental study that indicates that MDF significantly supports programming skill development. I plan on verifying these results by running another experimental study.

CCS CONCEPTS

• **Applied computing** → **Education**; Learning management systems; • **Social and professional topics** → **Computational thinking**; **CS1**; **Student assessment**;

KEYWORDS

CS Education; Immediate Feedback; Student Model; Misconception

ACM Reference format:
Luke Gusukuma. 2018. A Misconception Driven Student Model to Author Feedback. In *Proceedings of 2018 International Computing Education Research Conference, Espoo, Finland, August 13–15, 2018 (ICER '18),* 2 pages.
https://doi.org/10.1145/3230977.3231015

1 PROGRAM CONTEXT

I am a computer science PhD student. My current research project started in Fall of 2016. I am analyzing a cognitive model I have developed, the Misconception Driven Student Model (MDSM). I have run one of two planned quasi-experimental studies to analyze the impact of MDSM on student learning. My work includes two published papers on MDSM and the experimental study's results. I plan on publishing additional results from the firs experiment and data from the second experiment.

2 CONTEXT AND MOTIVATION

Programming is difficult to learn; increasing programming experience is one of the most effective ways to learn programming[9]. To increase the efficiency of knowledge acquisition, students need frequent practice with plentiful immediate feedback, grounded in instruction [8]. However, delivering well-designed feedback for numerous programming problems to build that experience can be a time-consuming task.

3 BACKGROUND & RELATED WORK

Creating immediate feedback for programming problems has two popular approaches: hint generation and unit testing (e.g. [7] and [3]). Hint generation suggests code edits to students while unit tests deliver feedback regarding program output. Both approaches require little instructor effort, but frequently fail to give feedback about program solutions [4] and contextualize feedback to instruction [3, 7]. While intelligent tutoring systems address this issue, they require significant extra effort and expertise[5]. In contrast, my approach intimately involves the instructor and mitigates instructor burden through technology and several reuse strategies by leveraging what current hint generation and unit testing techniques currently lack, a well articulated model to contextualize immediate feedback to instruction for several programming problems.

4 STATEMENT OF THESIS/PROBLEM

To contextualize automated feedback to instruction, I propose the following thesis: *authoring feedback using a cognitive student model supports student learning of programming.* This thesis requires confronting a number of challenges:

(1) What is an appropriate cognitive student model?
(2) How can this model be used practically by instructors to author feedback contextualized to their instruction?
(3) How can the impact of the feedback on learning be measured?

5 RESEARCH GOALS & METHODS

The cognitive model I propose is the Misconception-Driven Student Model (MDSM). MDSM models student knowledge by mapping observed programming mistakes to sets of inferred misconceptions; this model enables detection of misconceptions, linking of feedback to instruction, immediate generation of feedback, and finer grained evaluation of students and feedback. While applicable to the learning of programming by all novice learners, I explore the impact of MDSM on non-computing majors.

MDSM builds on the idea of knowledge components, "an acquired unit of cognitive function or structure that can be inferred from performance on a set of related tasks."[6]. Framing misconceptions as undesirable knowledge components and mistakes as student performance, I define two interrelated ideas, a (programming) misconception and a (programming) mistake, as follows:

ICER '18, August 13–15, 2018, Espoo, Finland

ACM ISBN 978-1-4503-5628-2/18/08.
https://doi.org/10.1145/3230977.3231015

- A programming misconception is a unit of cognitive function or structure that can be inferred from a mistake on a programming task.
- A programming mistake is an incorrect configuration of code elements.

The model is defined as follows: a programming mistake maps to an associated (inferred) set of programming misconceptions. This model's major implication is that automatically detecting programming mistakes inherently implicates the underlying misconceptions. This model relies on having a set of discovered misconceptions that we can map to a set of mistakes. This model and the challenges posed in section 4 leads to four research questions:

5.1 Research Question 1

What is an appropriate cognitive student model on which to base feedback for students learning to program? My tentative answer to this question is MDSM. This model may need to be refined as the research proceeds and new insights and evidence are gained.

5.2 Research Question 2

How can we discover misconceptions? Usage of MDSM, necessitates misconception discovery. As misconceptions are inferred from mistakes, experts (e.g. instructors) must discover these mistakes. Mistake discovery techniques are described in [1]; they include observing code, machine learning, and personal experience.

5.3 Research Question 3

How can we detect misconceptions in student code and deliver instructor-authored immediate feedback based on misconceptions? My proposal for authoring of contextualized feedback is a specification language for authoring mistakes' automatic detection and feedback delivery. I have outlined this specification in [2]. I refer to instructor-authored feedback contextualized in misconceptions as Misconception-Driven Feedback (MDF). The implementation of the specification involves implementing a modified tree-inclusion algorithm for ASTs and abstract interpretation.

5.4 Research Question 4

How does feedback grounded in instruction impact student learning? I plan on measuring the impact of MDF through experimental studies using multiple choice tests, programing problems, surveys, and log data. I use multiple choice tests to measure students' recall and understanding. I use programming problems to identify deficiencies in students' practical skills by viewing distributions of misconceptions detected by MDF. I use surveys to measure the students' perceptions of MDF on their learning.

6 DISSERTATION STATUS

I have used MDSM to develop an instructional design process (Instructional Design + Knowledge Components a.k.a. ID + KC)[1] and MDF. ID + KC is a misconception discovery-centric Instructional Design process used to develop an instructional unit on iteration that was deployed in classrooms.

I completed program analysis software implementation of MDF and deployed it to collect treatment data regarding the impact of MDF. This experiment was run on 290 students over one control semester and two treatment semesters; instructors, course content, number of TAs, etc. were controlled to isolate the MDF's effect. Summarily, the experiment's results suggests that MDF supports development of programming skills to a significant degree with an average score increase of 10%. These results help to answer research questions 3 and 4 and have been submitted for review.

The continuation of this research includes reviewing more log data, and applying the MDSM and MDF to a different programming context. None of my dissertation has been written yet. This is pending on first completing my research proposal document and an estimated dissertation completion date of Fall of 2019.

7 EXPECTED CONTRIBUTIONS

My expected contributions are as follows:

(1) The Misconception Driven Student Model, a cognitive model suitable for formulating immediate feedback
(2) Two Quasi-experimental studies analyzing the impact of Misconception Driven Feedback, a product of the MDSM
(3) A software implementation of program analysis techniques to detect mistakes and deliver Misconception Driven Feedback.

I anticipate the results of the second experiment will also provide evidence that MDF supports student learning.

ACKNOWLEDGMENTS

This work is supported in part by National Science Foundation grants DUE 1624320, DUE 1444094, and DGE 0822220.

REFERENCES

[1] Luke Gusukuma, Austin Cory Bart, Dennis Kafura, Jeremy Ernst, and Katherine Cennamo. 2018. Instructional Design+ Knowledge Components: A Systematic Method for Refining Instruction. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. ACM, 338–343.
[2] Luke Gusukuma, Dennis Kafura, and Austin Cory Bart. 2017. Authoring feedback for novice programmers in a block-based language. In *Blocks and Beyond Workshop (B&B), 2017 IEEE*. IEEE, 37–40.
[3] Georgiana Haldeman, Andrew Tjang, Monica Babeş-Vroman, Stephen Bartos, Jay Shah, Danielle Yucht, and Thu D Nguyen. 2018. Providing Meaningful Feedback for Autograding of Programming Assignments. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. ACM, 278–283.
[4] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. 2016. Towards a systematic review of automated feedback generation for programming exercises. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*. ACM, 41–46.
[5] Kenneth R Koedinger, Vincent Aleven, Neil Heffernan, Bruce McLaren, and Matthew Hockenberry. 2004. Opening the door to non-programmers: Authoring intelligent tutor behavior by demonstration. In *International Conference on Intelligent Tutoring Systems*. Springer, 162–174.
[6] Kenneth R Koedinger, Albert T Corbett, and Charles Perfetti. 2012. The Knowledge-Learning-Instruction framework: Bridging the science-practice chasm to enhance robust student learning. *Cognitive science* 36, 5 (2012), 757–798.
[7] Thomas W Price, Yihuan Dong, and Dragan Lipovac. 2017. iSnap: Towards Intelligent Tutoring in Novice Programming Environments. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. ACM, 483–488.
[8] Marieke Thurlings, Marjan Vermeulen, Theo Bastiaens, and Sjef Stijnen. 2013. Understanding feedback: A learning theory perspective. *Educational Research Review* 9 (2013), 1–15.
[9] Chris Wilcox and Albert Lionelle. 2018. Quantifying the Benefits of Prior Programming Experience in an Introductory Computer Science Course. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. ACM, 80–85.