Learner-Sourcing in an Engineering Class at Scale

Elena L. Glassman
MIT CSAIL
elg@mit.edu

Chris Terman
MIT CSAIL
cjt@mit.edu

Robert C. Miller
MIT CSAIL
rcm@mit.edu

ABSTRACT

Teaching computer architecture as a hands-on engineering course to approximately 250 MIT students per semester requires a large, dedicated teaching staff. This spring, a shortened version of the course will be deployed on edX to a potentially far larger cohort of students, without additional teaching staff. To better support students, we have deployed developmental versions of three learner-sourcing systems to as many as 500 students. In this work, learner-sourcing refers to crowd-sourcing within the community of students enrolled in a course.

Our three systems have distinct objectives: (1) learner-source hints for debugging, (2) learner-source hints for optimization, and (3) prompt students to reflect on other students' solutions. These systems harvest and organize students' collective knowledge about designing and debugging solutions to assignments. The key design principle is that students write hints immediately after completing a task or fixing a bug. Later, other students can use these hints to help guide them to a correct solution or better design. We plan to deploy and study the next iteration of these systems on edX this Spring.

Author Keywords

engineering education; crowd-sourcing; learner-sourcing

ACM Classification Keywords

H.5.m. Information Interfaces and Presentation (e.g., HCI): Miscellaneous

INTRODUCTION

One-on-one tutoring has been established as a costly gold standard in education [3]. Engineering education is no exception. By opening up the virtual classroom to thousands of students, MOOCs make the teacher-to-student ratio drastically lower. Teaching staff become even more distant from students' solutions because they can no longer walk around the lab or classroom interacting with students one-on-one to get a sense of common success and failure modes. They also suffer from the "curse of knowledge," which is the difficulty experienced by experts when trying to see some concept or artifact from the perspective of a novice [4].

We are developing systems which harvest and organize students' collective knowledge about designing and debugging solutions to assignments. Students themselves are each becoming experts on their own bugs and solutions. The key design principle is that students write hints immediately after completing a task or fixing a bug. Later, other students can use these hints to help guide them to a correct solution or better design. While students do not have the pedagogical content knowledge to necessarily generate the optimal explanations, they also do not suffer from the curse of knowledge. These systems relieve some of the pressure on the relatively small teaching staff, and give students the valuable educational experiences of reflection and generating explanations [5].

We present on-going case-studies of three systems deployed in an undergraduate computer architecture course at MIT. Around two hundred and fifty students enroll each semester. To pass, students must each build an entire simulated processor out of logic gates. The three deployed systems have distinct objectives: (1) learner-source hints for debugging, (2) learner-source hints for optimization, and (3) prompt students to reflect on other students' solutions. In this work, learner-sourcing refers to crowd-sourcing within the community of students enrolled in a course.

These systems have already been deployed at scale to as many as five hundred students over multiple semesters. Our residential case-studies prepare us to design, deploy, and study the next iteration of these systems on edX this Spring. Throughout the process, we continue to develop design principles that can guide future learner-sourcing systems for engineering classes.

RELATED WORK

In order to generate hints for others, students must reflect on their bug, solution, or optimization, and then communicate it to others. We review some relevant literature at the intersection of learning theory and engineering education.

Reflection

Reflection and confusion are both treated at length in learning theory literature. Piaget theorized that cognitive disequilibrium, experienced as confusion, could trigger learning: the creation or restructuring of knowledge schema [11]. However, D'Mello et al. point out that, for this learning to take place, it is important for confusion to be both appropriately injected and resolved [7]. Dewey theorized that reflection is a critical method for triggering that transformation from conflict and doubt into clarity and coherence [6]. Turning that reflection into a self-explanation also improves understanding [5].

Given the established value of reflection, Turns et al. [16] argue that the absence of reflection in traditional engineer-

L@S'14, March 4–5, 2014, Atlanta, GA.
ACM 978-1-4503-2669-8/14/03
((DOI HERE))

ing education scholarship is a significant gap. Professor Susan Ambrose, Northeastern's Vice Provost for Teaching and Learning has called on engineering curriculum designers to incorporate reflection [1]. In this work, we aim to design scalable automated opportunities for students to reflect.

Peer Instruction and Assessment

While reflection is valuable in its own right, it is also a building block of larger frameworks, like Peer Instruction [12] and Peer Assessment [15]. Peer instruction requires students to form educated guesses, and then discuss and reflect on their choices with peers. Peer Assessment replaces peer discussion with peer evaluation. For example, in Peer Assessment Learning Sessions [13], students assessed each other's written calculations and sketches in numerical problem solving courses, i.e., Hydraulic Engineering and Reinforced Concrete Design.

Reflecting on a peer's conceptual development or alternative solution may bring about cognitive conflict that prompts re-evaluation of the student's own beliefs and understanding [10]. We aim to trigger productive cognitive conflict that students can attempt to resolve through written reflection.

Generating Hints for Students

With a mixture of automation and human input, helpful hints have been delivered students in multiple problem domains. Some of these domains can be solved by students taking one of a family of sequential steps toward the final correct answer, such as logic proofs [2] and physics problems [8]. Other domains are less constrained, e.g., introductory programming assignments [14]. Finally, some domains, such as general software development, are complex enough that hints are based on errors and local information about a student's work, rather than a global understanding of that students' work [9]. Our students' processors are complex enough that they fall into this third, most complex problem domain.

Hartmann et al.'s HelpMeOut [9] system handles arbitrarily complex Java code by tracking and storing the changes programmers make when fixing a compilation error or runtime exception. Users, presumably students and teachers, can write helpful messages to accompany these automatically extracted bug fixes. Like HelpMeOut for hardware design, one of our systems also asks students to generate hints for peers struggling with the same bug and indexes those hints according to their associated verification failure. Another system we have deployed allows students to learner-source design optimizations, which are larger transitions from one working solution to another.

DEBUGGING: CROWD-SOURCING HINTS FROM LEARNERS

In our course, students design entire simulated processors composed of logic gates. These processors, expressed as pages' worth of an in-house declarative hardware description language, can be challenging to debug even with the one-on-one help of seasoned teaching staff. In fact, students who have recently resolved a particular bug can be in a better position than available staff members to help a fellow student with the same bug.

Bugs are revealed by voltage verification failures. A testbed monitors a set of simulated wires within the student's processor during execution of a staff-provided sequence of instructions. During each simulation, the testbed reports the first wire that fails verification and the time of that failure. There are hundreds of possible unique verification failures, some more common than others. The course has a Piazza forum, but the generic forum structure made it awkward and time-consuming to search for other students' questions about resolving particular verification failures.

We built *Dear Beta*, a system that serves as a central repository of debugging advice for and by students, indexed according to verification failures. The processor in this class is called the Beta, and the system's name comes from the fact that it looks like a spreadsheet with an advice column, as shown in Figure 1. As soon as a student resolves a bug that was causing a particular verification failure, they can post an explanation of their bug on Dear Beta along with the verification failure it caused. Providing the bug-resolving explanation is pedagogically useful, and students struggling with a particular verification failure can easily look up advice for their particular failure, if another student or staff member has added a hint. Students can upvote hints they found helpful.

A particular verification failure may be caused by one of several possible bugs. As the number of students and/or cumulative semesters becomes sufficiently large, crowd-sourcing students' own bug-fixes may generate hints about all the possible causes of a particular verification failure.

System usage statistics, interviews with teaching staff, and anecdotal evidence, including a student's unprompted class forum thank you note, suggest that many students find the system to be a consistently helpful tool. As one teaching assistant said, "Whenever I came to help a student, I first asked them if they'd checked Dear Beta."

We hope to create a Meteor or Django-backed version of Dear Beta for both the residential and potentially much larger online cohort of students who take the course this Spring. We anticipate that students can cover a larger fraction of verification failures with hints that are just as helpful as the original staff-provided debugging hints.

OPTIMIZATION: CROWD-SOURCING HINTS AND RESULTS FROM LEARNERS

As a final exercise in our residential course, students optimize their own processors for additional points, which are a function of the processor's size and speed. Students also get a long list of optimization hints written by staff, which students consult when choosing optimizations to implement. Students can take hours to act on a single optimization hint. Some students hear stories of the form, "I tried x and got y points!" One student heard that two classmates got significantly different outcomes from implementing the same two optimizations in opposite orders. This does not make sense, technically, but

B	C	D	F	G	H	I
Node(s)	Time (ns)	Hint	# Upvotes	Upvote Here	Give A Hint	
		If you were a lab assistant helping a student with this problem, what would you say to help them fix it?		*Did this hint help you?*	*Do you have your own advice to add about*	
ma[31:0]	399	Look carefully on how the WDSEL mux works, pay attention to the ordering of its inputs.	8	upvote	give a new hint for this error	
ma[31:0]	399	it could also be that your bsel is wrong	1	upvote	give a new hint for this error	
ma[31:0]	399	Check that your ALU is functioning correctly - it's possible to pass Lab 3's checkoff without actually having a fully functional ALU	0	upvote	give a new hint for this error	
mwd[31:0]	1499	It's most likely a problem with your REGFILE. Make sure you're handling R31 correctly both for radata and rbdata.	3	upvote	give a new hint for this error	
mwd[31:0]	1499	make sure d0 in your mux4 for wdsel is connected to gnd, not ia[31:0].	1	upvote	give a new hint for this error	
mwd[31:0]	1499	Remember that wmd should be connected to one of the output of the regfile, and not the wd of the regfile itself	1	upvote	give a new hint for this error	
mwd[31:0]	1499	mwd is the memory write address. It is not the same thing as the memory you want to write in the registers.	1	upvote	give a new hint for this error	
mwd[31:0]	1499	Don't forget about BSEL!	0	upvote	give a new hint for this error	

Figure 1. *Dear Beta* **serves as a central repository of debugging advice for and by students, indexed by verification failures. For example, there are three separate crowd-sourced hints for the first verification failure ($ma[31:0], 399ns$), ordered from greatest to least upvotes. The left two columns, wire name and time, uniquely characterize the verification failure. The central pale yellow column is for the hint. The right-most three columns display the number of upvotes, a link that will add an upvote to the hint, and a link to add a different hint for the same verification error.** *Dear Beta* **is implemented with Google Spreadsheets, Forms, and Apps Scripts.**

these kinds of optimization tales are all students have had to go on.

We deployed a system for learner-sourcing optimization hints and results, with the aim of giving students more transparency and assistance. Rather than indexing hints based on verification failures, we indexed hints based on whether they were intended to reduce their processor's size, increase its speed, or both. We also gave students the option of submitting the magnitude of the speed and size improvement they got from acting on a hint, so that future students could gauge which optimization hints were most beneficial on average.

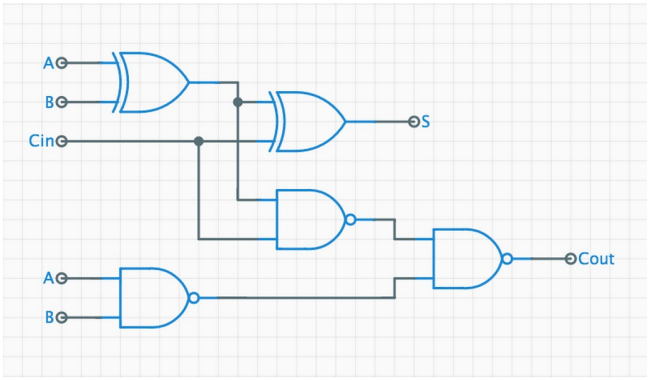
This system was not as successful as Dear Beta. We are still diagnosing why it failed. For example, it did not collect hints about verification failures created *during* the optimization process, which were common. We are, however, confident that students are ultimately capable of generating high quality processor optimization hints, based on results shown in the next section on reflection and comparison.

REFLECTION: PROMPTED REFLECTIONS ON OTHER STUDENTS' SOLUTIONS

In our course, one of the early digital design assignments students complete is the construction of a Full Adder. A Full Adder adds two bits of information, plus an optional 'carry-in' bit, to produce a 'sum' bit and 'carry out' bit. Two possible solutions are shown in Figure 2. Since the assignment only requires a working Full Adder circuit and there is no pressure to optimize their solution, students create a variety of solutions that fit the behavior specification. Some are bloated, and some make use of one or more 'tricks' to use as few transistors as possible. In this scenario, fewer transistors translates to better performance.

Through exploration of hundreds of previous students' solutions, we found that alternative correct solutions are nearly completely distinguishable by the number of transistors they contain. We picked a set of representative student solutions and rendered them graphically, rather than as raw student code. Each student was asked to compare their own solution to (1) a worse solution in the representative set and (2) to

gates: 21, fets: 96



gates: 24, fets: 114 (expected solution)

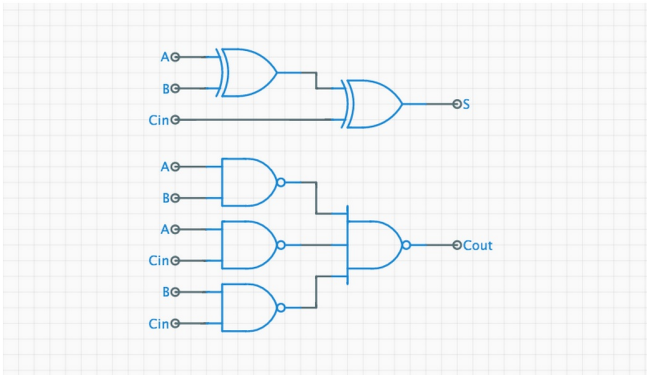


Figure 2. Two correct student solutions (Full Adders), composed of different numbers and arrangements of XOR and NAND gates. The more optimal solution has only 21 gates and 96 transistors (FETs) while the less optimal solution has 24 gates and 114 transistors.

the best solution in the representative set. Figure 2 illustrates one possible pair of alternative solutions.

Students were asked to give a hint to future students about how to improve the poorer solution in each pairing. When the student's own solution is the better solution in the pair,

Learner-sourced Advice
"Do not try to be too clever with C_{out}—design your schematic as the expression is written. This way you will achieve the [standard] schematic."
"Keep in mind that although expressions like XOR and OR are not equivalent in isolation, when combined in a larger expression, they may serve the same purpose. Because you've already found the XOR of A and B for example, you can then use that as the OR of A and B when you know A AND B will be ANDed with A XOR B later on."
"Mutate the boolean function for C_{out} such that all OR and AND operations are being NOT'ed. This allows you to design a circuit using only naturally inverting CMOS gates."
"I would ask: is there a way for you to use some intermediate node in one circuit to bypass a CMOS gate in the other, leading to a reduction of used mosfets?"

Table 1. Examples of learner-sourced advice for optimizing fellow students' Full Adders.

then the student can diagnose and describe, as a hint, what the poorer solution's creator had conceptually missed. For example, *Remember DeMorgan's Law: you could replace the 'OR' of 'ANDs' with a 'NAND' of 'NANDs.'* When the students' own solution is the poorer solution in the pair, then they are challenged to understand how the better solution uses fewer transistors to achieve the same functionality, and then explain that to the future student that did not yet have that insight.

This reflection and explanation process is pedagogically valuable on its own. In addition, students' explanations give a rich window into their understanding, while serving as strikingly cogent potential advice to future students. Table 1 shows examples of student-generated hints collected during our deployment. The online version of this course may afford us the opportunity to deliver these optimization hints back to students, when they complete a new, online-only adder optimization lab.

CONCLUSIONS AND FUTURE WORK

This work in progress is the accumulation of several semesters of system development and deployment in a large undergraduate engineering course. We have found that students can write high quality optimization advice for simple digital circuits when their solution is paired with a solution that is different from theirs. We have also found that crowdsourcing students' debugging hints, indexed according to the verification failures they are associated with, is an effective way to help students help each other. We are preparing to deploy the next iteration of these systems on edX this Spring, while we actively consider how best to measure their impact on the student learning experience.

ACKNOWLEDGMENTS

We appreciate the support of the NSF Graduate Research Fellowship, Quanta Computer, and the Amar Bose Teaching Fellowship for funding this work.

REFERENCES

1. Ambrose, S. A. Undergraduate engineering curriculum: The ultimate design challenge. *The Bridge 43*, 2 (2013), pp. 16–23.
2. Barnes, T., Stamper, J. C., Lehmann, L., and Croy, M. J. A pilot study on logic proof tutoring using hints generated from historical student data. In *EDM* (2008), pp. 197–201.
3. Bloom, B. S. The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational researcher* (1984), pp. 4–16.
4. Camerer, C., Loewenstein, G., and Weber, M. The curse of knowledge in economic settings: An experimental analysis. *Journal of Political Economy 97*, 5 (1989), pp. 1232–1254.
5. Chi, M. T., De Leeuw, N., Chiu, M.-H., and Lavancher, C. Eliciting self-explanations improves understanding. *Cognitive Science 18*, 3 (1994), pp. 439–477.
6. Dewey, J. How we think: A restatement of the relation of reflective thinking to the educational process. *Lexington, MA: Heath* (1933).
7. D'Mello, S., Lehman, B., Pekrun, R., and Graesser, A. Confusion can be beneficial for learning. *Learning and Instruction 29*, 0 (2014), pp. 153–170.
8. Gertner, A. S., Conati, C., and VanLehn, K. Procedural help in andes: Generating hints using a bayesian network student model. In *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, AAAI '98/IAAI '98, American Association for Artificial Intelligence (Menlo Park, CA, USA, 1998), pp. 106–111.
9. Hartmann, B., MacDougall, D., Brandt, J., and Klemmer, S. R. What would other programmers do: suggesting solutions to error messages. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2010), pp. 1019–1028.
10. Kavanagh, L., and O'Moore, L. Reflecting on reflection-10 years, engineering, and uq. In *Proceedings of the 19th Annual Conference of the Australasian Association for Engineering Education: To Industry and Beyond*, Institution of Engineers, Australia (2008).
11. Kibler, J. Cognitive disequilibrium. In *Encyclopedia of Child Behavior and Development*, S. Goldstein and J. Naglieri, Eds. Springer US, 2011, pp. 380–380.
12. Mazur, E. *Peer Instruction: A User's Manual*. Series in Educational Innovation. Prentice Hall, 1997.
13. O'Moore, L. M., and Baldock, T. E. Peer assessment learning sessions (pals): an innovative feedback technique for large engineering classes. *European Journal of Engineering Education 32*, 1 (2007), pp. 43–55.

14. Singh, R., Gulwani, S., and Solar-Lezama, A. Automated feedback generation for introductory programming assignments. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, ACM (New York, NY, USA, 2013), 15–26.

15. Topping, K. Peer assessment between students in colleges and universities. *Review of Educational Research 68*, 3 (1998), pp. 249–276.

16. Turns, J., Sattler, B., Yasuhara, K., Borgford-Parnell, J., and Atman, C. Integrating reflection into engineering education. In *Proceedings of the ASEE Annual Conference and Exposition*, ACM (2014).