Learnersourcing Personalized Hints

Elena L. Glassman, Aaron Lin, Carrie J. Cai, Robert C. Miller
MIT CSAIL
Cambridge, MA USA
{elg,aaronlin,cjcai,rcm}@mit.edu

ABSTRACT

Personalized support for students is a costly gold standard in education, but it scales poorly with the number of students. Prior work on *learnersourcing* presented an approach for learners to engage in human computation tasks while trying to learn a new skill. Our key insight is that students, through their own experience struggling with a particular problem, can become experts on the particular optimizations they implement or bugs they resolve. These students can then generate hints for fellow students based on their new expertise. We present workflows that harvest and organize students' collective knowledge and advice for helping fellow novices through design problems in engineering. Systems embodying each workflow were evaluated in the context of a college-level computer architecture class with an enrollment of more than two hundred students each semester. We show that, given our design choices, students can create helpful hints for their peers that augment or even replace teachers' personalized assistance.

Author Keywords
intelligent tutoring systems; crowdsourcing; learning at scale

ACM Classification Keywords
H.5.m. Information Interfaces and Presentation (e.g. HCI): Miscellaneous

INTRODUCTION

One-on-one human tutoring is a costly gold standard in education. Mastery-based instruction with corrective feedback can offer a substantial improvement in learning outcomes over conventional classroom teaching [1]. However, personalized support does not scale well with the number of students enrolled. In large classes, it is often not feasible for students to get personalized hints from a teacher in a timely manner. Massive open online courses (MOOCs) have teacher-to-student ratios that are smaller than large residential classes by orders of magnitude. Intelligent tutoring systems have strived to simulate the type of personalized support received in one-on-one tutoring, but they are expensive and time-consuming to build.

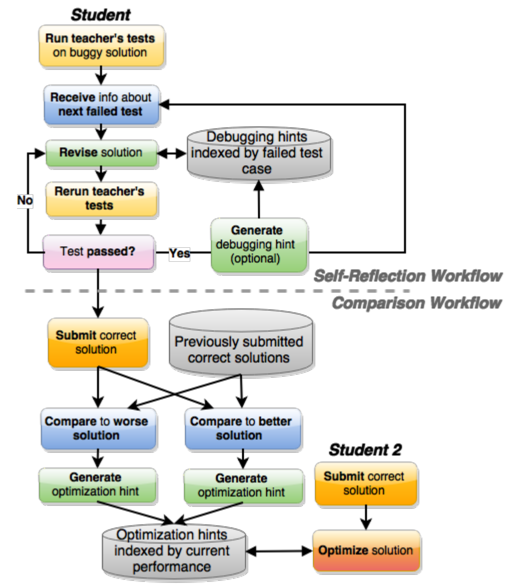

Student
Run teacher's tests on buggy solution
Receive info about next failed test
Revise solution
Debugging hints indexed by failed test case
Rerun teacher's tests
No
Test passed?
Yes
Generate debugging hint (optional)
Self-Reflection Workflow
Comparison Workflow
Submit correct solution
Previously submitted correct solutions
Compare to worse solution
Compare to better solution
Student 2
Generate optimization hint
Generate optimization hint
Submit correct solution
Optimization hints indexed by current performance
Optimize solution

Figure 1. In the *self-reflection* workflow, students generate hints by reflecting on an obstacle they themselves have recently overcome. In the *comparison* workflow, students compare their own solutions to that of other students, generating a hint as a byproduct of explaining how one might get from one solution to the other.

In this paper, we turn to learners for generating personalized hints. Prior work on *learnersourcing* presented an approach for learners to collectively generate educational content for future learners, such as video outlines and exam questions, while engaging in a meaningful learning experience themselves [14, 22]. The proposed benefit of learnersourcing is that learners are not only more intrinsically motivated to engage with the learning content to begin with, but may also benefit pedagogically from the task itself.

Our work builds upon learnersourcing by exploring how it can be applied to the generation of personalized hints during more complex problem solving. Whereas prior work determined which task to present to the learner depending on what information was still needed [22], many educational topics like digital circuit design require more domain expertise, raising the question of which learners should be assigned to generate which hints. Thus, beyond learnersourcing subgoals for how-to videos, we tackle the core challenge of generating content that is tailored to both the *hint-receiver*'s current progress and the *hint-author*'s likely level of understanding.

We present two workflows for learnersourcing hints that assign hint-generating tasks to learners based on what problems the learner has recently solved. In the *self-reflection* workflow, students generate hints by reflecting on an obstacle they themselves have recently overcome. In the *comparison* workflow, students compare their own solutions to those of other students, generating a hint as a byproduct of explaining how one might get from one solution to the other. The second workflow operates on the output of the first, as shown in Figure 1. In both workflows, the key insight is that, through their own experience struggling with a particular problem, learners can become experts on the particular optimizations they implement or bugs they resolve. The workflows can take pressure off teaching staff while giving students the valuable educational experiences of reflection and generating explanations.

While such workflows could have many applications, this paper presents a specific application within a college-level computer architecture class. In this course, students implement, debug, and optimize simulated processors by constructing digital circuits. During both the debugging and optimization process, hints are one mechanism for teachers to help students fix and optimize their circuits. This paper applies our learnersourcing workflows to two kinds of hints: *debugging hints* and *optimization hints*. A debugging hint is a student's attempt to help a future student change their solution so it generates the expected output for a particular input. An optimization hint is a student's attempt to help a future student get from one correct solution to another, more optimal solution. When hint-receivers encounter that particular situation during their problem-solving process, the hints can be shown to them as if they are the personalized hints an intelligent tutoring system might generate, or that a teacher might provide during a one-on-one interaction.

This paper makes the following contributions:

- A design space for learnersourcing personalized hints, drawing from experiences during early prototyping.
- Two learnersourcing processes for generating hints by matching hint-author tasks to the needs of hint-receivers.
- The results of deployment in a 200-student class, and an in-depth lab study with 9 participants. Results from these studies show that personalized hints can be viably learnersourced, and that these hints serve as helpful guides to fellow students encountering the same obstacle or attempting to reach the same goal.

RELATED WORK

Our work builds on prior research on delivering personalized support to students. It is also informed by existing research on the pedagogical benefits of reflection and explanation.

Personalized Support

Several types of solutions have been deployed to help students get the personalized attention they need. These solutions span the spectrum from recruiting more teaching assistants from the ranks of previous students [15] to automating hints using intelligent tutoring systems.

Intelligent tutoring systems provide personalized hints and other assistance to each student based on a pre-programmed student model. For example, previous systems sought to provide support through the use of adaptive scripts [12], or cues from the student's problem-solving actions [6]. Despite the advantage of automated support, intelligent tutoring systems often require domain experts to design and build them, making them expensive to develop. Furthermore, domain experts who generate these hints may also suffer from the "curse of knowledge": the difficulty experts have when trying to see something from a novice's point of view [16].

Discussion forums derive their value from the content produced by the teachers and students who use them. These systems can harness the benefits of peer learning, where students can benefit from generating and receiving help from each other. However, as the system has no student model, the information is available to all students whether or not it is ultimately relevant. Students can receive personalized attention only if they post a question and receive a response.

Reflection and Explanation

In this work, we aim to design opportunities for students to help others while simultaneously reflecting on their own solutions. Existing theories indicate that reflection is a critical method for triggering the transformation from conflict and doubt into clarity and coherence [5]. Turning that reflection into a self-explanation further improves understanding [2]. According to Turns et al. [19], the absence of reflection in traditional engineering education is a significant shortcoming.

One challenge inherent in learnersourcing hints is the possibility that novices could become confused if asked to reflect on their solution and compare it to a fellow student's solution. Piaget theorized that cognitive disequilibrium, experienced as confusion, could trigger learning due to the creation or restructuring of knowledge schema [10]. D'Mello et al. maintain that confusion can be productive, as long as it is both appropriately injected and resolved [7].

Similarly, reflecting on a peer's conceptual development or alternative solution may bring about cognitive conflict that prompts reevaluation of the student's own beliefs and understanding [9]. As such, peer instruction [13] and peer assessment [18] have not only been integrated into many classroom activities, but have also formed the basis of several online systems for peer-learning. For example, Talkabout organizes students into discussion groups based on characteristics such as gender or geographic balance [11]. HelpMeOut assists programmers debugging errors by suggesting solutions that peers have applied in the past [8]. Recent work on learnersourcing proposes that learners can collectively generate educational content for future learners while engaging in a meaningful learning experience themselves [14, 22]. Beyond existing work, we investigate alternatives for what support students should be prompted to provide, based on their own work as well as the needs of their peers. We also explore several ways to trigger productive reflection as a byproduct of hint creating, by prompting students to either self-reflect or compare their own solutions to those produced by peers.

DESIGN SPACE

We faced a number of decisions when designing interventions to collect and deliver hints. Here we discuss these decisions and some alternatives.

When should learners be asked to provide hints? As soon as a student has resolved a bug, they may have some expertise about that bug that they can share with other students. If too much time has passed between resolving the bug and writing a hint, the student may forget necessary details and context, or forget the bug altogether. A previous learnersourcing system also prompted students to contribute content immediately after having experienced it themselves, for similar reasons [22].

How should learners access hints? A "push" model of hint distribution might display all student-generated hints as a constantly updating resource, following the standard model of a course-wide discussion forum. Alternatively, a "pull" model would dispense hints to individual students just-in-time, when a student needs help. The hints could be algorithmically selected based on the student's work so far and the hints they have already received. We explored both of these models, using the push model for distributing debugging hints and the pull model for distributing optimization hints. Generating optimization hints was a required reflection activity, so the volume and redundancy of these hints made a push model potentially overwhelming.

What hints can we ask, or allow, a student to generate? In cases where the student's *start state* (prior to overcoming an obstacle) and *end state* (after overcoming an obstacle) are known, such as when a student fixes a bug, we ask the student to create a hint helping other students encountering a similar bug or start state. For example, in the case of circuit design, we consider a student who has recently fixed a bug resolving a particular verification error to be capable of writing a debugging hint associated with that verification error.

In many cases, however, a student might not face any explicit obstacle, or their start state may not be known. For example, a student might naturally arrive at a highly optimal circuit design without having first tried a less optimal design. Regardless of the path to their solution, the student could generate hints by comparing their own solution to a more optimal solution, or to a less optimal solution. In this paper, we explored both of these directions by asking each student to do both comparisons. To keep hint generation relevant to the learner's current task and to minimize cognitive load, we did not ask students to generate hints between pairs of solutions when they were familiar with neither solution.

How should hints be indexed? Indexing hints by a meaningful feature of student solutions allows students to more easily find relevant hints in a push model of hint distribution and allows the system to deliver more relevant hints to each student in a pull model of hint distribution. Optimization hints could be indexed by the learner's start state, end state, or start-and-end states with respect to some metric of optimality. Debugging hints could be indexed by verification errors.

During debugging, students' solutions are run through sequences of teacher-designed test inputs. A *verification error* occurs when a student's solution does not return the expected values. Because test cases have a specification of actual and expected outputs for each input, we decided to index debugging hints by the tests for which the solution deviates from the expected output. In other words, debugging hints are associated with the verification error that disappears if the bug is resolved.

During optimization, the goal is not simply to attain a correct solution, but rather to arrive at a more optimal correct solution. We decided to index optimization hints by start-and-end state: the leap from a less optimal solution to a more optimal solution that the hint is intended to inspire. These states, the solutions themselves, are complex circuit objects; we use the number of transistors in a solution as a metric of its optimality. In this indexing scheme, all hints written with the intent of helping a student with a 114-transistor solution create a 96-transistor solution are binned together.

Which hints should a student receive? In the push model of hint distribution, this question is not relevant, but in the pull model of hint distribution, it may be critical. Ideally, students would receive a progression of increasingly specific hints, following patterns of adaptive scaffolding established in intelligent tutoring system (ITS) literature [20].

Each hint should also be selected to help a student reach a more optimal solution on the spectrum. This can be done in several ways. Considering the zone of proximal development, a student with a 144-transistor solution could be provided with hints to get to a 132-transistor solution, which is the *next optimal solution* found by other students (see Figure 2). Alternatively, if the most common solution—the mode of the student solution distribution—is more optimal than the student's current solution, the student could be given hints to reach this *common solution*. As a final alternative, the student could receive hints to get them from a 144-transistor solution to the *most optimal solution* (96 transistors) found by other students. This may require a more significant restructuring of their solution.

Should hint-creation be a required task? As discussed in the Related Work, generating hints can be a valuable part of the learning process. We required all students to generate optimization hints as part of a reflection activity immediately after submitting their first correct circuit. We did not require students to generate debugging hints. This is because the number of bugs encountered could be large, and unlike optimizations, many bugs also may not lead to significant conceptual gain upon reflection. Because debugging hints are immediately pushed out to all students, we keep both hint creation and upvotes voluntary to minimize the signal-to-noise ratio in hint quality.

How should the variation in hint quality be handled? In the push model of hint distribution, we used users' upvotes to sort hints by quality. In the pull model of hint distribution, we took advantage of the redundancy of the hints, and presented five hints at once. If one or more redundant hints were of poor quality, their aggregate message might still be helpful for a student. If most of the hints were about an feature that

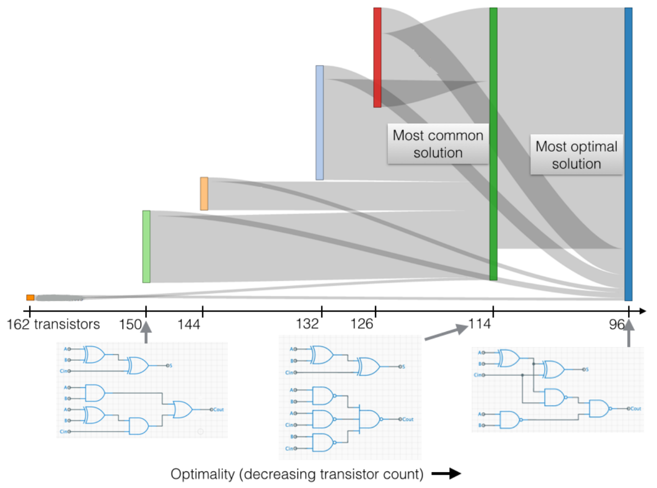
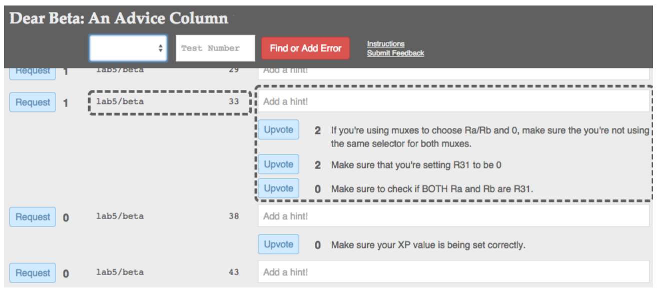


Figure 2. Sankey diagram of hints composed between types of correct solutions, binned by the number of transistors they contain. The optimal solution has only 21 gates and 96 transistors while the most common solution generated by students has 24 gates and 114 transistors.


Dear Beta: An Advice Column

Test Number | Find or Add Error | Instructions | Submit Feedback

Request 1 lab5/beta 29 Add a hint!

Request 1 lab5/beta 33 Add a hint!

Upvote 2 If you're using muxes to choose Ra/Rb and 0, make sure the you're not using the same selector for both muxes.

Upvote 2 Make sure that you're setting R31 to be 0

Upvote 0 Make sure to check if BOTH Ra and Rb are R31.

Request 0 lab5/beta 38 Add a hint!

Upvote 0 Make sure your XP value is being set correctly.

Request 0 lab5/beta 43 Add a hint!

Figure 3. ***Dear Beta*** **serves as a central repository of debugging advice for and by students, indexed by verification errors. In this figure, there are three learnersourced hints, sorted by upvotes, for a verification error on test no. 33 in the 'lab5/beta' checkoff file.**

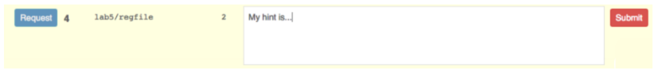

Request 4 lab5/regfile 2 My hint is...| Submit

Figure 4. After fixing a bug, students can add a hint for others, addressing what mistake prevented their own solution from passing this particular verification test.

is irrelevant to the student receiving the hints, the remaining hint(s) might be about something different and more relevant. We limited each set of hints to a size of five to avoid overwhelming the learner with too many hints.

How public should the hint-author be? Many systems for question-answering have a reputation system, where the author is known and recognized for contributing answers. Previous work at CSCW has examined whether reputation would improve class forum participation [3]. For simplicity, we chose to leave student identities hidden.

USER INTERFACE

We designed two user interfaces, one for each type of hint. To learnersource debugging hints, we built and deployed *Dear Beta*, a Meteor web application that serves as a central repository of debugging advice for and by students in the class. The name is an allusion to both the "Dear Abby" advice column and the Beta processor that students create in the class. To learnersource optimization hints, we built and deployed *Dear Gamma*, a web interface students were required to fill out as a reflection activity after submitting their final correct circuit for a class assignment.

Dear Beta

We applied the self-reflection learnersourcing workflow to Dear Beta. After fixing a bug, a student can post a hint on Dear Beta about how to resolve the bug, along with the verification error it caused. Other students encountering that verification error can look up these hints, upvote helpful hints, and contributing new hints.

Consider a student working on their Beta within the digital circuit development environment provided by the computer architecture class. The student runs the staff-designed test file x on their circuit. The development environment alerts them to a verification error: for a particular input (test number n), y was the expected output and the student's circuit returned z. After further examining the circuit with no success, they pull up the Dear Beta website in order to get help (Figure 3).

Dear Beta displays all errors and hints, sorted by test file name x then test number n, on a single page, based on student and teacher feedback on a previously deployed prototype. The student can either scroll to find hints for their verification error or, to quickly navigate to the hints for their verification error, a student can alternatively enter x and n into the bar pinned to the top of the page and click "Find or Add Error." This will scroll into view the hints for that error and highlight them. If the error was not yet in the Dear Beta system, the error will be added and scrolled into view.

To add a new hint, they can click in the text box labeled "Add a hint!" so that it expands into a larger textbox sufficient for typing out a paragraph of hint text (Figure 4). If there are hints already available for the verification error of interest, these hints are sorted by the number of upvotes they have already received. Students can click the "Upvote" button next to any hint if they find it helpful.

If there are no hints yet, or if the hints are unhelpful, the student can click the "Request" button to the left of the error, which increments a counter. This button helps communicate the need for hints for a particular verification error to potential hint writers. This is analogous to the "Want Answers" button and counter on Quora, a popular question-and-answer site.

Dear Gamma

In order to learnersource optimization hints, we caught students at a different stage in their learning process: right after they passed all verification tests for a particular digital circuit, the Full Adder. Because students may have arrived at their solution without encountering any particular optimization obstacles, Dear Gamma uses the comparison workflow for learnersourcing rather than the self-reflection workflow.

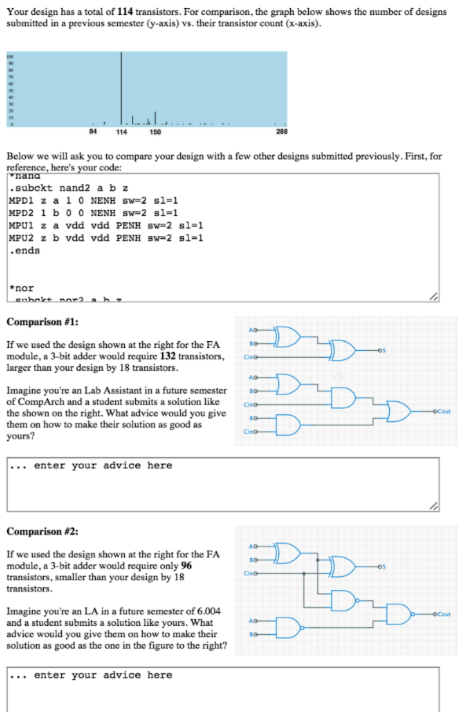
The comparison workflow is modified slightly, to accommodate the requirements of the course lecturer, who wanted to make sure that all students get a chance to consider both the most common and the most optimal solutions. The collection of previous student solutions in Figure 1 was also curated by the lecturer. If a student's solution is larger than the most common solution, they are not shown solutions larger than their own; instead, they are asked to consider both the most common and the most optimal solutions, so they benefit from seeing both without doing extra work overall. Students with the most optimal solution only consider alternative solutions that are worse than theirs. Figure 5 shows an example of the page for a student with a 114 transistor solution.

In this activity, students are given a pair of solutions and asked to give a hint to future students about how to improve from the less optimal solution to the more optimal solution. Students write hints for two such pairs of solutions. In each pair, one of the solutions is always their own. When the student's own solution is the better solution in the pair, then the student can hint at what the peer might have 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, they are challenged to first understand how the better solution uses fewer transistors, and then write a hint about the insight for a peer. To aid the student in comparing solutions, the Dear Gamma interface displays the student's own solution as a reminder of their design, as well as an alternative picked from among other students' solutions.

Specifically, if a student's solution *S* is just as or more optimal than the most common solution, they are asked to (1) write a hint to help a future student with a less optimal solution reach solution *S* and (2) write a hint to help a student with solution *S* reach the most optimal solution. If a student's solution *S* is less optimal than the most common solution, they are asked to write a hint to help a student with solution *S* reach (1) the most common solution and (2) the most optimal solution. This scheme ensures that all students are familiar with the most common solution and the most optimal solution and have written two hints to help future students improve the optimality of their solutions.

EVALUATION
To evaluate the extent to which learnersourced hints can support problem solving, we deployed Dear Beta and Dear Gamma to the computer architecture class, which had an enrollment of more than two hundred students. Dear Beta was deployed for 6 weeks, during which we collected student-generated debugging hints and observed the simultaneous usage of those hints in a real-world setting. Dear Gamma's optimization hint collection interface was released to students as part of a particular lab. We then conducted a lab study with nine students to understand how they solve a typical engineering problem using these learnersourced optimization hints.

The questions our evaluation sought to answer are: (1) *What are the characteristics of student-generated hints?* and (2) *Can learners solve problems using those hints?*


Your design has a total of **114** transistors. For comparison, the graph below shows the number of designs submitted in a previous semester (y-axis) vs. their transistor count (x-axis).

Below we will ask you to compare your design with a few other designs submitted previously. First, for reference, here's your code:

```
*nand
.subckt nand2 a b z
MPD1 z a 1 0 NENH sw=2 sl=1
MPD2 1 b 0 0 NENH sw=2 sl=1
MPU1 z a vdd vdd PENH sw=2 sl=1
MPU2 z b vdd vdd PENH sw=2 sl=1
.ends


*nor
.subckt nor2 a b z
```

Comparison #1:

If we used the design shown at the right for the FA module, a 3-bit adder would require **132** transistors, larger than your design by 18 transistors.

Imagine you're an Lab Assistant in a future semester of CompArch and a student submits a solution like the shown on the right. What advice would you give them on how to make their solution as good as yours?

```
... enter your advice here
```

Comparison #2:

If we used the design shown at the right for the FA module, a 3-bit adder would require only **96** transistors, smaller than your design by 18 transistors.

Imagine you're an LA in a future semester of 6.004 and a student submits a solution like yours. What advice would you give them on how to make their solution as good as the one in the figure to the right?

```
... enter your advice here
```

Figure 5. This is the Dear Gamma interface for a student with a solution containing 114 transistors. In the first comparison, they are asked to write a hint for a future student with a larger (less optimal) correct solution. In the second comparison, they are asked to write a hint for a future student with a solution similar to their own so that they may reach the smallest (most optimal) correct solution.

DEAR BETA
The Dear Beta website was released as a stand-alone additional resource for students one week prior to the due date for the final circuit design lab. Students were made aware of its existence through a class forum announcement and signs on chalkboards in the course's computer lab. It was left up for the remainder of the semester for students to refer to, if completing work late. We tracked student logins and engagement with the site's features. An initial prototype of Dear Beta was deployed for two consecutive semesters prior to this final system design and study, as well. Teachers' feedback on those initial deployments is summarized in the Results section.

DEAR GAMMA
Hint Succession and Categorization
While Dear Beta makes all hints available at all times, Dear Gamma is modeled on the hint-giving mechanism of an ITS. In prior work, sequences of hints have been posited to facilitate learning due to their similarity with sequences used in expert human tutoring, as well as their support of human memory processes [17]. Therefore, we further decomposed the hints collected with Dear Gamma into the three kinds of hints that typically comprise a hint sequence: 1) *pointing hints* direct the student's attention to the location of error in case the student understood the general principle but did not know to apply it; 2) *teaching hints* explain why a better solution exists by stating the relevant principle or concept; 3) *bottom-out hints* indicate concretely what the student should do [20].

Two researchers independently categorized the 435 collected Dear Gamma hints into six different categories: pure pointing hints (*p*), pointing and teaching hints (*pt*), pure teaching hints (*t*), teaching and bottom-out hints (*tb*), pure bottom-out hints (*b*), and hints that are irrelevant or clearly not helpful. They first independently labeled the first 30 hints. After discussing disagreements and iterating on their understanding of the hint categories, the coders then categorized the remaining 405 hints.

If one coder labeled a hint as a hybrid between two categories (i.e. teaching and pointing) while the other coder labeled it with only one category (i.e. pointing), we assigned the hint to the pure category (i.e. pointing) that was in common between the two coders' labels. If there was no shared category across the two coders, the hint was discarded. We also excluded the minority of hints (3.2%) that were labeled as irrelevant or unhelpful.

Lab Study

Nine out of the 226 current students in the computer architecture course participated in the study. These students were recruited through a course forum post. Participants were given $30 for the study, which lasted one hour.

Students were informed that we were studying the effectiveness of hints for optimizing circuits so that they use fewer transistors. Students were not told who generated these hints, nor that the hints were generated by students. During the study, three batches of hints were shown in the order of pointing, teaching, and bottom-out, but randomly selected within each category. Students began by opening up their previously completed lab and reviewing their solution. The experimenter noted down the number of transistors in their solution, and randomly selected five pointing-type hints for a solution of that size from the Dear Gamma collection. For example, if the student had 114 transistors in their solution, they received five hints previously generated by students who had written a hint to help improve a 114-transistor solution. Because hints may be of variable quality, the researcher presented hints in batches of five to increase the chances that one of them may be helpful.

The experimenter then asked the student to reduce the number of transistors in their solution. The experimenter explained that there were two more batches of five hints ready for them if they became stuck. These two batches were teaching hints and bottom-out hints. Students could consult outside resources like the course website and Google as well.

After receiving each batch of hints, participants answered the following 7-scale Likert scale questions about each hint (1:strongly disagree, 7:strongly agree): (1) *"This hint taught me something."* (2) *"This hint helped me get to a more optimal circuit."* and (3) *"I feel more confident that I could solve a similar problem in the future."* We placed these questions immediately after each batch of hints to capture user perception of hints at the time they occurred. However, to avoid slowing down the problem-solving process, participants were asked to explain their answers in writing only after the study, in the post-study questionnaire. This rating process was repeated for the teaching hints and bottom-out hints, even if students were able to solve the problem without asking for these hints.

After the study, users completed a post-study questionnaire regarding their overall impressions. Because users were shown a batch of hints at a time, all of which were student-generated, in the post-study questionnaire we added additional Likert-scale items, "I was able to find the most helpful hints and ignore the rest" and "Many hints felt repetitive," to understand whether users felt they could adequately ignore irrelevant hints.

LIMITATIONS

Because these studies do not have control groups, we cannot conclude on the magnitude of the effect on student learning. We can only report qualitative and quantitative measures of teachers' and students' engagement with the system. Some of those observed behaviors and opinions may be derived from the participants' sense of novelty, rather than the underlying value of the system. However, we deployed Dear Beta in a real classroom setting, and in the context of a real assignment, for the purpose of observing natural interaction with the system.

RESULTS

Teacher Feedback From Early Prototypes

After deploying initial prototypes of Dear Beta for two semesters, we invited Teaching Assistants to share their complaints, requests, and experiences with us. Four TAs were interviewed, in person or by email, and their feedback and experiences informed Dear Beta's final design.

TA_A adapted to Dear Beta's deployment by first asking each help-seeking student if they had already consulted Dear Beta. If they had not, he came back to them after visiting everyone else in the lab help queue. When he came back to the original help-seeking student, they had often already resolved their problem with Dear Beta's hints, and had a new bug they wanted help debugging. This TA also strongly supported its existing design as a single scannable sorted list for quickly finding hints, rather than a purely search-based hint retrieval mechanism or the more general class forum.

TA_B described Dear Beta as a "starting point" for students, many of whom used it diligently during debugging. TA_B appreciated that students who did ask for her help no longer said, "My Beta isn't working. Tell me why." Instead, they used Dear Beta as a starting point, to help them identify potential locations of a bug in many pages of code.

TA_B wondered if the extra hints were making it too easy to complete the lab, possibly letting students pass without understanding. TA_C echoed this concern, but he made sure each student actually understood the Dear Beta hints whenever he personally guided them through the debugging process. TA_C was able to describe with specific examples how Dear Beta helped him help students quickly resolve common bugs.

Like TA_A, TA_C asked every help-seeking student if they had already consulted Dear Beta. TA_D was absent during most

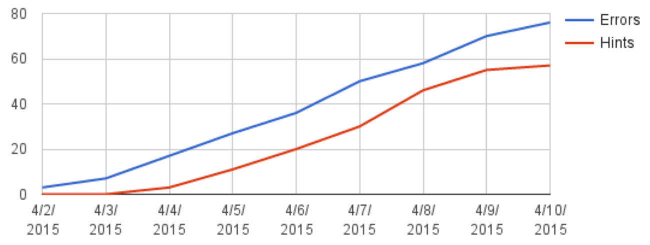


Figure 6. Between Dear Beta's release (4/2) and the lab's due date (4/10), verification errors were consistently being entered into the system. The addition of hints followed close behind.

of Dear Beta's deployment but still regularly recommended Dear Beta to students who asked for her help over email. A fifth TA declined to be interviewed on account that she did feel she had interacted with Dear Beta enough.

TA_B was concerned that the level of student involvement in producing hints might be too low. The affordances for contributing new hints in this initial prototype were not obvious and rarely visible on small screens. The final Dear Beta design is more externally consistent with other participatory Q&A systems, has more salient buttons for contributing new hints, and a responsive design that accommodates screens as small as a cell phone.

Dear Beta Study

For the week prior to the lab assignment due date, the number of registered unique users in the Dear Beta system rose linearly from 20 to 166. It plateaued at 180 by one week after the lab was due. For comparison, the total number of students in the class was 226. 119 students logged in more than once and many students logged in repeatedly.

In the 9 days between Dear Beta's release and the lab's due date, users added 76 verification errors and 57 hints as a response to those errors. Half of the errors received at least one hint. Seven errors received as many as three hints. Figure 6 shows users' engagement with the system over time. As soon as the initial stock of hints were available, students began upvoting them.

Users contributed 61 upvotes and 10 downvotes on the hints during the same period. The highest number of upvotes (10) was given to the hint *"When entering constants, 1#4 is 1111 and 1'4 is the 4-bit representation of 1."* Remember that, while this is a teaching-type hint, it is provided as a targeted troubleshooting hint for students whose solution fails to pass a specific test case. The second most upvoted hint (5 upvotes) was *"Make sure your ASEL logic is correct - don't allow the supervisor bit to be illegally set."*

None of the hints appear to be incorrect, though this is difficult to verify, since the teachers do not have copies of the solutions from which these hints were generated. Even within a collection of hints for the same error, not all will be relevant to any particular solution.

Dear Gamma Study

With the Dear Gamma hints, six of the nine laboratory subjects were able to improve the optimality of their circuits within the hour that the study took place. Figure 7 illustrates the subjects' revisions. One student only needed one set of pointing hints. Five students successfully revised their circuits after one set of pointing and one set of teaching hints. Four students received a set of final bottom-out hints as well. Three of those four students (S_2, S_5, and S_9) were still unable to successfully revise their circuits.

Hint Distribution Figure 2 is a Sankey diagram of the optimization hints collected by Dear Gamma. The number of hints between certain key transitions, such as between the most common and the most optimal solutions, had far more hints because of the lecturer's requests for pedagogically valuable hint prompts that introduced hint-writers to the common and optimal solutions.

The most common solution size was 114 transistors. Students with that common solution were randomly assigned to generate hints from one of the many different larger solutions down to theirs. These hints are pooled together with the hints written by students with solutions larger than 114 transistors who are seeing the common 114 transistor solution for the first time. Less than five percent of students' solutions were the most optimal (96 transistors), but, at the request of the lecturer, every student was asked to consider that most optimal solution and write a hint for a fellow student on how to optimize their solution into that most optimal solution.

The number of hints produced by students appear to correlate with subjects' degree of optimization success, but this may be just coincidental. Students in the study were drawn from the same population as the hint generating students, and all study subjects were offered the same number of hints (5 pointing, 5 teaching, and 5 bottom-out) over the course of the hour long session, regardless of the solution they started with.

Hint Types Table 1 shows the breakdown of hints by type, along with representative examples. The Cohen's κ [4] interrater reliability was 0.54, which indicated that the two coders had moderate agreement across the six hint categories [21].

Hint Prompt Hint-authors interpreted the prompt to create a hint in different ways. Some addressed the hint-receiver directly (*"Keep in mind that..."*), while others addressed the teaching staff (*"I would mention [to the student]..."*). Some hint-authors did not directly write a hint, but instead wrote about how they'd approach the situation of being a lab assistant for the hint-receiver: *"I think first I'd ask to make sure they knew what a NAND3 was, because I think a solution like this might come from not totally understanding how it works."* Still others took a conversational approach, as if they were having an unfolding conversation with the hint-receiver. Interestingly, a number of hint-authors referred to "here" or "my circuit" in their hints, as if the hint-receiver would be looking at the Dear Gamma interface, with all its examples, rather than just the text generated by the hint-author. This particular assumption on the part of the hint-author was confusing for hint-receivers.

Optimization Issues S_5 was the only student who had a standard, optimizable solution, received hints, and had no insights about how to optimize the circuit within the allotted hour.

Hint type	Count	(%)	Representative examples
Pointing (*p*)	62	14%	"You don't have to keep S and Cout as two separate/independent CMOS gates."
Pointing and teaching (*pt*)	81	19%	"Instead of making the S and Cout components individual, combine them together to save computation power."
Teaching (*t*)	111	26%	"Instead of recalculating values, save computation results to save time!"
Teaching and bottom-out (*tb*)	19	4%	"Via application of demorgan's theorem, NAND2 (XOR A B) Cin is equivalent to NAND3(NAND2 A B) Cin."
Bottom-out (*b*)	78	18%	"Use the output of a xor b for one of the nand2 gates."
Unhelpful/irrelevant	14	3%	"Use the hints provided by the lab, but try to improve on them."
No coder agreement	70	16%	

Table 1. Breakdown of Dear Gamma hints by type. Students in the Dear Gamma lab study initially received 5 pointing hints (*p*), followed by 5 pure teaching hints (*t*), and finally 5 pure bottom-out hints (*b*), delivered whenever the student was stuck and asked for more help.

S_1, S_2, and S_9's forward progress was confounded by having near-optimal top-level architecture and very large (sub-optimal) implementations of the underlying modules. Dear Gamma only shows hint-authors the top level architecture, not the underlying gate implementations, for the alternative solutions they compare their own solutions to. They therefore found the hints, which were often about fixing high-level architecture, irrelevant and unhelpful. Even so, S_1 was still able to revisit the hints and correctly extract the lesson that only inverting gates should be used. As a result, S_1 successfully optimized their circuit.

While working through their optimizations and hints, S_6 was the one student who significantly deviated from the correct line of thought by removing all inverting gates.[1] As soon as S_6 saw that their transistor count had increased rather than decreased, they revisited the hints, realized their mistake, and correctly optimized their circuit. None of the hints themselves were incorrect, though some were deemed irrelevant or unhelpful.

Hint Progression One student successfully optimized their solution from 150 transistors to the most common solution, 114 transistors, using only pointing and teaching hints. With some time left in the hour-long session, the student opted to optimizing their circuit further. The experimenter gave the student one last set of hints, for the transition from 114 to the optimal 96 transistor solution. However, the experimenter did not restart the progression for this next transition; the student was given a set of bottom-out hints. Based on these hints, the student got the final optimization step without understanding, and appeared to feel cheated from the satisfaction of figuring it out himself.

Student Reactions The six subjects without suboptimal gates agreed with the statement *"Overall, these hints helped me get to a more optimal circuit"* (μ=6, σ=1.1 on a 7-point Likert scale). The remaining three subjects with suboptimal gates disagreed with the same statement (μ=2.6, σ=2.1 on a 7-point Likert scale). Regardless of whether a subject's solution included suboptimal gates, subjects on average agreed with the statement, *"Hints helped me think differently about the problem, even if they didn't directly help me solve the problem"* (μ=5.4, σ=1.6).

[1]Optimal solutions have *only* inverting gates.

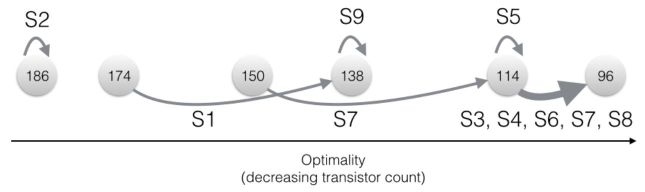


Figure 7. Six of the nine lab study subjects were able to improve the optimality of their circuits with the help of the Dear Gamma hints. Subject S7 was able to make two leaps–one to a common solution with 114 transistors and another from the common solution to the most optimal solution at 96 transistors.

Some subjects commented on the redundancy within each set of five hints of a particular type. This was sometimes expressed as a negative, as in *"These are all hinting at the same thing but I want new information,"* and sometimes expressed as a positive, as in *"Several hints are mentioning X.... I should look into it."* One student told the experimenter that, while the individual hints were hard to understand, together they formed a clearer picture in her mind about what to do.

DISCUSSION

In this section, we first address the research questions the evaluation was intended to answer. We then explain that, of the design decisions made during the design of Dear Beta and Dear Gamma, the critical factors for success included prompt clarity, the index chosen for hints, how alternative solutions are represented, and the use of hint progressions.

Answers to Research Questions

Our study sought to evaluate the characteristics of student generated hints. We can see from Table 1 that students, without coaching, can naturally produced hints that point, teach, give a bottom-out direction, or provide some combination of those elements. However, the number of pointing hints labeled by *both* coders as purely pointing-type (22) was much smaller than the number of such hints in the teaching (75) and bottom-out (64) categories. Because students were not informed that their hint would be slotted into a progression, it is possible they may have felt that if they were going to give a future student one hint, it would need to be more substantial than just pointing to a particular location in the solution and hoping the hint-receiver would see the possibility of optimization.

Secondly, the studies sought to evaluate whether learners can solve problems using these hints. Both studies suggest that these student-written hints are helpful. The aggregate activity of students and teachers on Dear Beta indicate that the resource was populated with helpful hints. The Dear Gamma lab study was set up based on the observed sub-optimality of students' circuits at the level of choosing and arranging gates. Students whose solutions were suboptimal in that anticipated way rated the hints as helpful. Students whose solutions were suboptimal in unanticipated ways, i.e., at the level of the gates themselves, were not well-served by the hints. Future optimization hint workflows will need both (1) an optimality metric that accounts for multiple common types of suboptimality and (2) a representation of solutions with an appropriate level of detail about the difference between any two solutions. Regardless, the Dear Gamma study suggests that students are helped by the hints when the optimality metric and representation are appropriate.

Lessons for Self-Reflection and Comparison Workflows

Prompt clarity appears to be critical for soliciting the highest possible quality of hints from students. In Dear Gamma, hint collection and delivery were separate processes. Some students misunderstood the prompt and wrote hints as if their audience was still the teacher, not a fellow student. Others did not understand that the hint-receiver would only see the text of their hint, not the diagrams it was based on. In Dear Beta, hint collection and delivery were all mediated through the same, constantly updated interface. The appropriate audience for a hint was clear. Future instantiations of the self-reflection and comparison workflows should clarify who the audience is for hint-authors, perhaps by displaying what learners will see.

The selection of an index for hints in the self-reflection workflow matters. In Dear Beta, the choice of test file name and test number as an index for hints worked well for a class of hundreds of students. In a MOOC-sized course, the index may need to include an indicator that specifies how the test failed as well. int indices in future systems should have sufficient information to group related hints into clusters of a manageable size.

The success of the comparison workflow depends not only the index for solutions, but also on how these solutions are represented in the workflow's prompt for hints. In the comparison workflow, we found that transistor counts sometimes did not account for lower-level reasons for a suboptimal circuit, resulting in hints that were unhelpful for solutions with lower-level suboptimality. Students will not generate hints that account for what has been abstracted away in the representation of solutions in the hint prompt. Likewise, if the definition of optimality used to index solutions does not account for a certain kind of suboptimality, the hints generated will be unlikely to help future students with that kind of suboptimality.

Lastly, when students request hints as they did in the Dear Gamma lab study, conforming to the ITS model of providing progressively specific hints is recommended. To automatically create hint progressions in the future, we could apply machine learning methods to estimate a given hint's type. Alternatively, we could learnersource hint classification.

Generalization

Although we applied these workflows to computer architecture problems, the self-reflection and comparison workflows could be extended to other domains. The workflows can be most readily applied to solutions that can be objectively tested for satisfying a set of requirements, e.g. passing unit tests, or whose optimality can be objectively measured. In domains without objective test cases or definitions for optimality, it may be more difficult to establish indices for clustering hints. In these cases, students could be asked to write what challenge they overcame or select from a growing list, enabling others to search for those terms. The comparison workflow could be modified to simply pair students with solutions different than their own, letting them judge for themselves which they think is better, the alternative solution or their own, and write hints based on that judgement.

CONCLUSIONS AND FUTURE WORK

This paper enriches learnersourcing by shaping the design space for learnersourcing personalized hints, and presenting two workflows that engage learners in hint creation while reflecting on their own work as well as that of peers. We built Dear Beta and Dear Gamma, which apply these workflows to the creation of debugging and optimization hints, matching students to the appropriate hint-creation task given their current progress. Results from our deployment study and subsequent lab study demonstrate the feasibility of these workflows, and indicate that learner-generated hints are helpful to learners. They also shed light on critical factors that may impact the quality of learnersourced hints, laying the groundwork for future systems in this area.

Next academic year, we plan to expand our deployment of Dear Beta to two new classes: a MOOC version of the computer architecture class in this paper and a residential college-level software engineering course taken by several hundred students each term. Solutions in the software engineering course are written in Java and tested against teacher-designed test suites. Each error could be specified by the problem set number, test name, and line number. The implementation of Dear Beta has been written generally so that it can simultaneously support both classes, each with their own schema for describing an error.

Future work for Dear Gamma will be focused on improving the metric for optimality. We will also explore what Dear Gamma in the context of the software engineering course could look like, e.g., prompting students to write hints based on fellow students' code that are either concise and clear or excessively verbose. We plan to continue deploying iterations of these workflows in classes for students' immediate benefit, and to demonstrate that it can enrich the learning experience across multiple engineering domains.

REFERENCES

1. Benjamin S Bloom. 1984. The 2 sigma problem: The search for methods of group instruction as effective as

one-to-one tutoring. *Educational Researcher* (1984), 4–16.

2. Michelene T.H. Chi, Nicholas De Leeuw, Mei-Hung Chiu, and Christian Lavancher. 1994. Eliciting Self-Explanations Improves Understanding. *Cognitive Science* 18, 3 (1994), pp. 439–477.
3. Derrick Coetzee, Armando Fox, Marti A. Hearst, and Björn Hartmann. 2014. Should Your MOOC Forum Use a Reputation System?. In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work and Social Computing (CSCW '14)*. ACM, New York, NY, USA, 1176–1187.
4. JA Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement* 20, 1 (1960), 37–46.
5. John Dewey. 1933. *How we think: A restatement of the relation of reflective thinking to the educational process*. D.C. Heath and Company.
6. Dejana Diziol, Erin Walker, Nikol Rummel, and Kenneth R Koedinger. 2010. Using intelligent tutor technology to implement adaptive support for student collaboration. *Educational Psychology Review* 22, 1 (2010), 89–102.
7. Sidney D'Mello, Blair Lehman, Reinhard Pekrun, and Art Graesser. 2014. Confusion can be beneficial for learning. *Learning and Instruction* 29 (2014), pp. 153–170.
8. Björn Hartmann, Daniel MacDougall, Joel Brandt, and Scott R Klemmer. 2010. What would other programmers do: suggesting solutions to error messages. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, pp. 1019–1028.
9. Lydia Kavanagh and Liza O'Moore. 2008. Reflecting on Reflection - 10 Years, Engineering, and UQ. In *Proceedings of the Conf. of the Australasian Assoc. for Engineering Education: To Industry and Beyond*. Institution of Engineers, Australia.
10. Jackie Kibler. 2011. Cognitive Disequilibrium. In *Encyclopedia of Child Behavior and Development*, Sam Goldstein and Jack A. Naglieri (Eds.). Springer US, p. 380.
11. Chinmay Kulkarni, Julia Cambre, Yasmine Kotturi, Michael S. Bernstein, and Scott R. Klemmer. 2015. Talkabout: Making Distance Matter with Small Groups in Massive Classes. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work and Social Computing (CSCW '15)*. ACM, New York, NY, USA, 1116–1128.
12. Rohit Kumar, Carolyn Penstein Rosé, Yi-Chia Wang, Mahesh Joshi, and Allen Robinson. 2007. Tutorial dialogue as adaptive collaborative learning support. *Frontiers in Artificial Intelligence and Applications* 158 (2007), 383.
13. E. Mazur. 1997. *Peer Instruction: A User's Manual*. Prentice Hall.
14. Piotr Mitros. 2015. Learnersourcing of Complex Assessments. In *Proceedings of the Second (2015) ACM Conference on Learning @ Scale (L@S '15)*. ACM, New York, NY, USA, 317–320.
15. Kathryn Papadopoulos, Lalida Sritanyaratana, and Scott R. Klemmer. 2014. Community TAs Scale High-touch Learning, Provide Student-staff Brokering, and Build Esprit De Corps. In *Proceedings of the First ACM Conference on Learning @ Scale (L@S '14)*. ACM, New York, NY, USA, 163–164.
16. N.J. Salkind. 2008. *Encyclopedia of Educational Psychology*. Number v. 1. SAGE Publications. 327–328 pages.
17. Robert A Sottilare, Arthur Graesser, Xiangen Hu, and Benjamin Goldberg. 2014. *Design Recommendations for Intelligent Tutoring Systems: Volume 2-Instructional Management*. Vol. 2. US Army Research Laboratory.
18. Keith Topping. 1998. Peer assessment between students in colleges and universities. *Review of Educational Research* 68, 3 (1998), pp. 249–276.
19. J. Turns, B. Sattler, K. Yasuhara, J. Borgford-Parnell, and C.J. Atman. 2014. Integrating Reflection into Engineering Education.. In *Proceedings of the ASEE Annual Conference and Exposition*. ACM.
20. Kurt Vanlehn, Collin Lynch, Kay Schulze, Joel A Shapiro, Robert Shelby, Linwood Taylor, Don Treacy, Anders Weinstein, and Mary Wintersgill. 2005. The Andes physics tutoring system: Lessons learned. *International Journal of Artificial Intelligence in Education* 15, 3 (2005), 147–204.
21. Anthony J Viera and Joanne M Garrett. 2005. Understanding interobserver agreement: the kappa statistic. *Fam Med* 37, 5 (2005), 360–363.
22. Sarah Weir, Juho Kim, Krzysztof Z. Gajos, and Robert C. Miller. 2015. Learnersourcing Subgoal Labels for How-to Videos. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work and Social Computing (CSCW '15)*. ACM, New York, NY, USA, 405–416.