

UC Berkeley

UC Berkeley Previously Published Works

Title

Debugging as a Context for Fostering Reflection on Critical Thinking and Emotion

Permalink

<https://escholarship.org/uc/item/1x47k4v1>

Authors

DeLiema, David

Dahn, Maggie

Flood, Virginia J

et al.

Publication Date

2019-09-12

DOI

10.4324/9780429323058-13

Peer reviewed

This is an accepted book chapter published by Routledge in the book, “Deeper Learning, Communicative Competence, and Critical Thinking: Innovative, Research-Based Strategies for Development in 21st Century Classrooms,” in 2020, available online: <https://www.routledge.com/Deeper-Learning-Dialogic-Learning-and-Critical-Thinking-Research-based/Manalo/p/book/9780367262259>

Reference: DeLiema, D., Dahn, M. Flood, V. J., Abrahamson, D., Enyedy, N., Steen, F. F. (2020). Debugging as a context for collaborative reflection on problem-solving processes. In E. Manolo (Ed.), *Deeper Learning, Communicative Competence, and Critical Thinking: Innovative, Research-Based Strategies for Development in 21st Century Classrooms* (part 4, chapter 12). New York: Routledge.

Debugging as a context for fostering reflection on critical thinking and emotion

David DeLiema¹, Maggie Dahn², Virginia J. Flood¹, Ana Asuncion³, Dor Abrahamson¹, Noel Enyedy⁴, Francis Steen²

University of California, Berkeley¹; University of California, Los Angeles²; 9 Dots³; Vanderbilt University⁴

Summary: The process of handling breakdowns in computer programming, a practice known as debugging, provides an auspicious context for fostering teacher-student communication about critical thinking. Toward this end, this chapter explores two practical classroom designs. The first design focuses on student journaling and art making about critical thinking processes and emotional experiences that undergird debugging. The second design focuses on instructors modeling and prompting for reflection on critical thinking strategies during debugging. These teaching strategies lead to growth in students' impressions of their skills for handling failure and their confidence during failure, both vital components of environments that promote deeper learning.

Introduction

When designing classroom activities to foster communicative competence, critical thinking, and deeper learning, educators should consider a common albeit challenging event in the learning process: the moment that a course of action breaks down, ushering in “a more reflective or deliberative stance toward ongoing activity” (Koschman, Kuutti, Hickman, 1998, p. 26; ; see also Schön, 1983). Because breakdowns in learning catalyze reflection and storytelling (Heider, 1958; Herman, 2009; Weiner, 1985), they naturally elicit *communication* about the learning process (DeLiema, 2017; Heyd-Metzunayim, 2015). In addition, the causes of failure are numerous, interconnected, and distributed across people, materials, and time (Hesslow, 1988; Suchman, 1987). Reasoning about failure thus warrants *critical thinking*: identifying facts about the breakdown, formulating alternative conjectures about possible causes, clarifying points of confusion, developing new knowledge about the problem, and presenting and weighing arguments for why an intervention might work (Facione, 1990; Greiff, Wüstenberg, Csapó, Demetriou, Hautamäki, Graesser, & Martin, 2014; Hmelo-Silver, 2004). Furthermore, moments of failure create a bedrock for *deeper content learning* when teachers provide responsive scaffolding (Kapur, 2008; Schwartz & Martin, 2004).

These observations raise a central question: How can educators take advantage of these opportunities in concert? In this chapter, we describe a pedagogical framework designed to promote deeper learning by *uniting* communication and critical thinking around moments of failure. In the proposed framework, students communicate about how they intend to address upcoming, as-yet-unknown breakdowns in learning, including by planning critical thinking strategies for moments of failure and approaches to negotiating the emotional components of failure. In turn, when breakdowns arise in learning, instructors respond to what students have communicated by modeling, prompting for, and reflecting on students' proposed strategies for handling failure. Afterwards, students reflect on past failures, evaluating the efficacy of their strategies and documenting their emotional experiences. This pedagogical approach establishes a connection between the application of critical thinking strategies *during* failure and communication about the critical thinking process *before* and *after* failure. In addition,

this approach acknowledges the inextricable relationship between thinking and emotion. Beyond providing examples of how instructors engage with this approach, this chapter covers preliminary evidence that these teaching strategies lead to growth in students' impressions of their skills for handling failure and their confidence during failure, both vital components of environments that promote deeper learning.

Description of teaching strategies

Computer programming as a context for communication about critical thinking

Practice-based documentation of these teaching strategies comes from the domain of computer science. In computer science, identifying and correcting errors, known as *debugging*, is part and parcel of the pursuit. For programmers, debugging is a routine part of coding, supported by specialized tools (e.g., syntax checkers and print statements) and critical thinking strategies (Murphy-Hill, Zimmermann, Bird, & Nagappan, 2013; Perscheid, Siegmund, Taeumel, & Hirschfeld, 2017). Although programmers develop debugging skills by coding, it is challenging to learn independently (Klahr & Carver, 1988). Current techniques for supporting debugging learning include providing students with resources to make debugging more tractable or efficient (Katz & Anderson, 1987; Ko & Myers, 2009), designing game-based contexts to facilitate debugging (Liu, Zhi, Hicks, & Barnes, 2017), and providing students with faulty artifacts to repair (Fields, Searle, & Kafai, 2016). Educational researchers have paid less attention to teaching strategies that promote and sustain student-driven communication about the critical thinking and emotional processes that surround debugging.

Planning and reflecting on critical thinking strategies for failure

Design principles

Our journaling and art making designs invited 5th – 10th grade students to communicate about critical thinking strategies and emotional experiences that surround debugging. To frame this work, instructors told stories about professionals' routine encounters with failure en route to progress, a practice found to normalize failure and motivate students (Lin-Siegler, Ahn, Chen, Fang, & Luna-Lucero, 2016). Students then envisioned “strategies and skills for dealing with everyday problems in school” (Oyserman, Terry, & Bybee, 2002, p. 316), specifically by planning critical thinking strategies for debugging. Students also created artwork about debugging that moved beyond conventional story archetypes about failure, examining how emotion shapes the process of building knowledge (Jaber & Hammer, 2016). Below, we outline the specifics of these instructional strategies.

Using journals for planning and reflecting

At the start of a coding session, students used personal coding journals to reflect on their past critical thinking strategies for debugging and set an intention to learn a new critical thinking strategy for debugging. Drawing on Twitter conventions, students wrote brief statements followed by hashtags: phrases that describe the topic of the message or “the tone of the message or the tweeter's emotions” (Mohammad & Kiritchenko, 2015, p. 302). Hashtags serve as “instrument(s) for creative self-expression and language play” (Heyd & Puschmann, 2017, p. 51). In our coding workshops, students wrote in personal journals responding to the following prompts:

- What debugging strategy worked well for you last time? #Hashtags
- Tweet your goal for when coding gets tough today. Choose one new debugging strategy to work on. #Hashtags.

The instructors framed students' goal setting by telling a story at the beginning of class about how someone outside of a coding context insightfully responded to failure, such as how a rock climber worked through a tough section or how an artist learned to draw an object that had previously stymied her. Instructors also gave students a chance to consult a visual map of the debugging-specific critical thinking strategies students surfaced at the workshop.

At the end of class, students evaluated the efficacy of their past debugging strategies and planned their approach to the next coding session. Students wrote specifically in response to the following prompts:

- Tweet a description of a bug you encountered. #Hashtags
- How well did your debugging strategy work? #Hashtags
- How do you think the bug got into your code in the first place? #Hashtags
- How will you tackle the next bug you encounter? #Hashtags

On some days, students wrote their goals and reflections on sticky notes, placing them on a large poster board (see Figure 1) or next to their laptop keyboards.

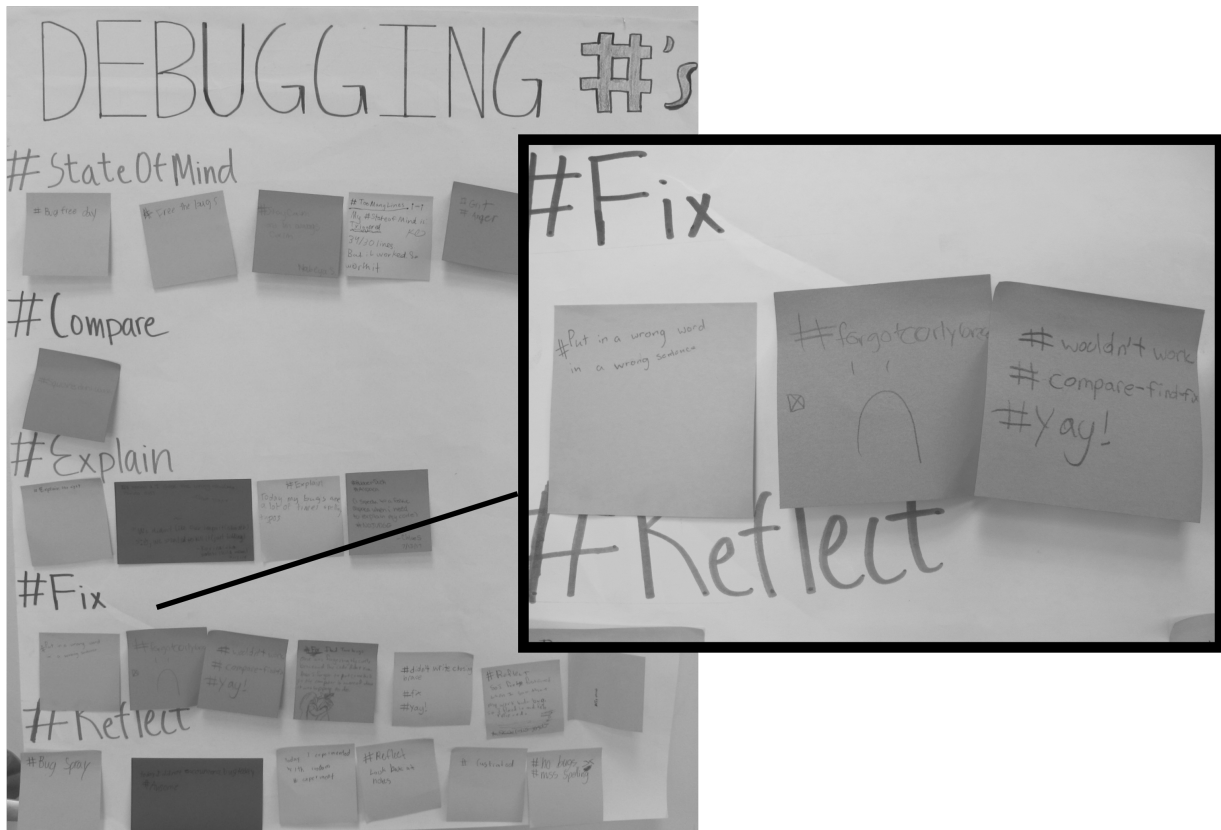


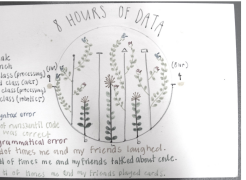


Figure 1. Students' sorted sticky note reflections; the overlaid image shows a zoomed-in portion.

Communicating about emotion and thinking processes through visual artwork

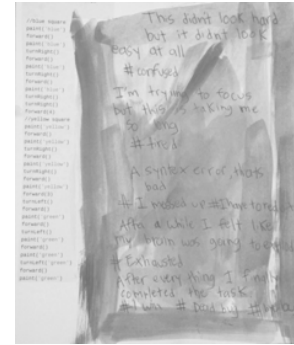
To make space for alternative forms of communication about the learning process, we capitalized on the centrality of emotion (Langer, 1953), metacognition (Goldberg, 2005), and transformation (Pelowski & Akiba, 2011) in art making. Students created abstract watercolor paintings, comic-strip-like panels, data visualizations, and code poems about coding and debugging (see examples and brief descriptions of each art project in Table 1 below). Each art class focused on an essential question (e.g., “How do artists use color, shape, texture, etc. to communicate different feelings?”) and began with a warm-up conversation about the topic. The warm-up invited storytelling about students’ recent experiences with coding. To familiarize themselves with the art materials and the topic, students then engaged in a quick, exploratory art making activity. After considering a broad range of works of art on the topic, students then used open-ended studio time (stretching over a few days) to develop their work of art. During art making, the instructor scaffolded students’ work by encouraging students to adopt flexible goals and remain open to uncertainty and surprise, allow the materials to guide exploration of the topic, ground the artwork in memories of coding experience, and embrace the challenge of producing art (Dahn, DeLiema, & Enyedy, in press). Students concluded by writing artist statements and sharing their artwork in whole class and small group settings. A central outcome of these art making activities was discovering *different ways of seeing, documenting, and showing experience*, in particular focused on the relationship between thinking and emotion during failure.

Table 1. Art making activities and example projects

Description of art making	Student example	
Abstract watercolor paintings: Students used watercolor, oil pastel, and colored pencils to create abstract depictions of emotions they experienced during moments of coding and debugging	In this piece I wanted to show a common emotion that I felt when I solve a bug. In this case that emotion was the feeling of awareness. To me being aware feels clear, bright, colorful, curious, and experimental. I used light and (cool) bright colors to represent brightness, and contrast.	
Comic-strip-like panels: Students created a simple coding and debugging story focused on an event unfolding over time	When I get a bug, I feel like a Rubiks Cube. Hard to solve, but looks easy. But then, an explosion becomes an answer.	
Data visualizations: Students collected data on a number of self-identified factors that were interesting to them during their coding and debugging process; students used data to create a visual representation of their experiences	<i>Symbols indicate progression of how time was spent, syntax errors, # of runs until code was correct, grammatical errors, # of times me and my friends laughed, # of times me and my friends talked about code, # of times me and my friends played cards</i>	

Code poems: Students printed out in-process code and then wrote free verse poems inspired by the lines in their code and memory of their experience

This didn't look hard but it didn't look easy at all
#confused
I'm trying to focus but this is taking me so long
#tired
A syntax error, that's bad
#I messed up #I have to redo it
After a while I feel like my brain is going to explode
#Exhausted
After everything I finally completed the task
#I will #Dead bug #bye bug



Engaging with critical thinking during failure

Design principles

The prior section focused on teaching strategies that encouraged students through journaling and art making to *plan before* and *reflect after* moments of failure. In this section, we describe how our instructors incorporated students' ideas about critical thinking *during* moments of failure. This teaching strategy helped form a close connection between planning, enacting, and reflecting, and ensured that students' prior communication about critical thinking strategies and emotion informed their coding practice. In particular, teachers and students focused on critical thinking strategies valuable to the process of navigating failure (Lewis, 2012), a practice programmers have long sought to support and understand (Ducassé & Emde, 1988; Freeman, 1964; Ripley & Druseikis, 1978). Below, we outline the specifics of these instructional strategies.

Modeling and prompting for critical thinking strategies during debugging

The foundation of this design is a set of critical thinking strategies for debugging that we amalgamated from research and from professional programmers' reflections on their practice (e.g., Zeller, 2009).

1. Pre debugging

- a. Get in the debugging state of mind: fearlessness, curiosity, thoughtfulness.
- b. Recall student's personally selected debugging goal/strategy for the day.

2. What is going wrong?

- c. Describe in granular detail what the program is doing when you run it.
- d. Describe in granular detail what you want to the program to do when you run it.

3. Propose an explanation for why the program is going wrong

- e. Propose a starting place to search for the bug.
- f. Explain what is happening in the code at that point.
- g. Explain how the code might be causing the problem when you run the program.

4. Attempt to fix the bug

- h. Form a plan to repair the code.
- i. Rewrite the code.

- j. Explain why this plan might work.
- k. Run the code.
- l. Return to step 2 if bug persists.

5. Reflect on the debugging process

- m. Reflect on the process taken during the debugging exchange.
- n. Talk about how the bug got there in the first place.
- o. Talk about goals for the next debugging exchange.

These debugging strategies correspond to recognized facets of critical thinking (Facione, 1990; Haynes, Lisic, Goltz, Stein, & Harris, 2016) by inviting students to *decode the significance* of observed outcomes of the program (c and d); *analyze the argument* or logic of their code (e and f); *evaluate* where that logic breaks down (g); *formulate alternative conjectures* about ways to fix the breakdown (h, i, and l); *justify* conjectures about the etiology and resolution for the breakdown (j and m); and *examine one's own* assumptions in iterative cycles of attempting a fix (l). In addition, the state of mind strategy aims to deliver on the goal of cultivating affective dispositions in critical thinking (Facione, 1990), such as self-confidence, open-mindedness, and alertness. Lastly, two strategies (b and o) incorporate students' *earlier* communication about critical thinking during debugging, uniting the processes of planning and enacting.

In their pedagogy, instructors participated in one-on-one and small-group debugging sessions with students. These sessions were informed by the tenets of reciprocal teaching (Palincsar & Brown, 1984). Instructors aimed to understand students' baseline debugging skills and looked for opportunities to model new and relevant critical thinking strategies, incorporating both the expert heuristics noted above and the strategies students planned in their journals. Modeling the strategy entailed carrying out the strategy and narrating what an expert might think when enacting it (Collins, Brown, & Newman, 1989). Moving forward, instructors then prompted students to use that strategy and reflect on how it works. Over the long run, instructors during debugging sessions with students aimed to listen to students articulate their strategies for debugging code.

Research evidence

This research took place in 2-week summer computer programming workshops (2 sessions; $n = 120$; 47 girls) and in 8-day weekend computer programming workshops (3 sessions; $n = 123$; 55 girls). The workshops served late elementary students, middle school students, and early high school students, all of whom either demonstrated financial need or attended schools with high proportions of students from low-income families. Undergraduate computer science majors at the beginning stages of developing their teaching practice worked as lead instructors. Students used four programming contexts -- OpenProcessing, PixelBots, Minecraft, and Lego Robotics -- to learn foundational computer science concepts in project-based environments. We documented classroom discourse during programming with multiple GoPro cameras and with screen recordings of students' coding activities, we photographed the artifacts students produced along the way (e.g., journals and artwork), and we conducted semi-structured interviews with students at the end of the workshop to gauge their thoughts about debugging. In addition, we collected survey measures of students' impressions of their own confidence and skill level with debugging.

Our analysis of artifacts and interviews involved iterative stages of looking at subsets of the data, writing memos about tentative themes emerging from the data, forming and reducing

categories/constructs, sharing inferences with the research team, and returning to the data, all features of the constant comparative method (Glaser, 1967). Our analysis of classroom discourse data focused on creating multimodal transcripts of moments of debugging and considering how participants worked together to accomplish debugging. This approach followed conventions of interaction analysis research (Jordan & Henderson, 1996; Goodwin, 2018). In our transcripts below, brackets signal overlapping talk, lines with arrows connect strips of talk to co-occurring changes in the environment and/or non-verbal actions of participants, words in italics describe observable action, numbers in parenthesis describe gaps in talk in seconds, and punctuation (e.g., a question mark) marks grammatical structure.

Journaling and art making about critical thinking and emotion

We found that asking students to journal about debugging generated reflection on a wide array of critical thinking strategies. Responding to the prompt to plan “one new debugging strategy” and then evaluate how well a past “debugging strategy work[ed],” students across two eight-session weekend workshops surfaced numerous strategies (see Table 2).

Table 2. Categorization of the debugging strategies students documented in their journals

Category	Strategy	Student quote
Myself	Work alone	Try to do it by myself
	Believe in myself	I want to be able to believe in myself
	Remind myself of past successes	When coding gets tough, I’m going to remind myself how far I’ve come
Social support	Teacher	If or when I have trouble...I would ask one of my mentors for help
	Peer	When it gets tough I will ask my peers for help
	Unspecified	When it gets tough ask someone
Emotion	Shyness	My goal is not to be shy
	Anger	Don’t get mad if I fail
	Disappointment	Don’t be disappointed
	Relaxation	#becalm
	Fear	When coding gets hard...don’t be afraid to be wrong #Don’t be Afraid
	Frustration	Don’t be frustrated when it doesn’t come out the way I wanted

Cognitive	Observe	My goal for when coding gets tough is to look very closely for my mistakes
	Memory	I'll try to not to forget which is left and right...
	Prior knowledge	My personal goal for the day when it gets tough is to put all of my knowledge I have learned before into one
	Think	#Think about it
Coding specific tools	Console	Joey helped me find the bug using console #thanksfam
	Stepper tool	I used the stepper to fix the problem
Miscellaneous	Effort	...I want to be able to fix it and not give up
	Novelty	Think of different ways to solve something #trydifferentstrategies
	Prior exemplars	We looked up the zene
	Experimentation	#fun experimenting (with code!)
	Play	Play with a lot then ask advice
	Focus	Consintrate on the question
	Find the cause	If coding gets tough today, I will...find whats wrong
	Make mistakes	Moreover, when we make mistakes it helps us learn
	Prepare	I will study to give me stuff that I need for the solution
	Faith	#believe
	Creativity	#be creative
	Define	Said what the x, y
	Interact	We worked with it...and got our answer
No strategy		My goal is to make it work after solving

Table 2 documents the set of actions that *students positioned as strategies* for debugging. Many of these strategies allude to domain-general critical thinking moves (Facione, 1990; Haynes et al., 2016): examining ideas (define, prior exemplars), analyzing arguments (find the cause, stepper tool), identifying new information (console), self-examining (prior knowledge), and drawing conclusions (experimentation). Other strategies, such as play, creativity, thought, and preparation, might encompass a number of critical thinking moves without explicitly labeling them. Yet other strategies implicate

resources pivotal to critical thinking: perception, memory, peers, teachers, and students themselves. Furthermore, in line with the recognition that there are dispositional elements to critical thinking (Facione, 1990), students described a range of strategies involving emotion, such as relaxing, disposing of fear, and curtailing frustration.

Separately, three art projects (abstract emotion drawings, code poems, and three-panel stories) offered a space for students to communicate about how affect surrounds moments of critical thinking during failure (Dahn, DeLiema, & Enyedy, under review). Students provided vibrant accounts of how it feels during failure, including feeling down (“when you gloomy, you really don’t see anything but nothing”), angry (“Mad is a fist that you are holding up”), and alert (“the feeling of awareness”). Moreover, students documented how emotions change or layer up, such as feeling “sad because I’m getting frustrated” or “how nervousness can take over, yet can become something beautiful if you change the perspective on it.” In particular, students described emotional states that arose during specific stages of the code writing process: “OH NO A BUG #MAD #feelingsmall,” “The red is supposed to represent the anger when I don’t get a bug,” and “Bug arises. Time to fix it. #Calm #Cool.” Because debugging is such a rich site for critical thinking, this artwork challenged our research-practice team to continue to grapple with how emotion, and interactions between emotions, co-occurred with and perhaps shaped how students worked through problem solving.

Importantly, students viewed making and communicating about art as capable of transforming their approach to coding. Students described a number of transformative potentials: self-understanding and awareness (“I learned how I got mad, how I got feelings”), setting expectations (“It made me understand how I feel and how I will when a bug comes”), shifting state of mind (“give me some hope and that I can fix it”), resting/relaxing/calming (“To keep you peace from overstress”), shifting emotion (“Art changed my way of feeling about coding”), and helping with confidence during problem solving (“And when I go to my other class after art, I feel like I can pass my challenge”). Similarly, students viewed their public-facing sticky note reflections as supports for thinking through and emotionally coping with debugging. Students discussed becoming aware that everyone debugs (“And when you look at other people’s sticky notes you’re like man I’m not the only person with this problem”), learning from the errors of their peers (“they can just go to the wall and learn from the other people’s mistakes”), recalling debugging strategies (“and it helps me know what I did so that next time I make a bug I can use that same process too”), returning oneself to a calm emotional state (“I feel like it let the stress out”), recognizing the classroom’s growth around debugging (“It’s just fun seeing your progress with your classmates like freaking out to calm”), and communicating one’s experiences (“I thought it was better to let out our feelings instead of just holding it in”).

Overall, these data show that given the opportunity to communicate about failure through journaling and art making in a supportive classroom environment, students generated a wide array of critical thinking strategies and developed rich insights into how emotion surrounds thinking during failure. Moreover, students believed that these reflective experiences transformed their experience of navigating the critical thinking and affective demands of failure, such as developing self-understanding, setting new expectations, reminding themselves of effective strategies, drawing on community knowledge, or simply honoring gradual progress.

Modeling, prompting, and reflecting on critical thinking during debugging

We now turn attention to critical thinking *during* debugging. In order to support students' more autonomous resolution of bugs, instructors often attuned their questioning and referring strategies to shape students' perception of the affordances of the programming environment for debugging (Flood, DeLiema, Harrer, & Abrahamson, 2018). This enskilment process (Ingold, 2000) incorporates conversation practices such as the use of *vague references* and *contracting and expanding question agendas*. Here we describe three other teaching strategies: *modeling* a new critical thinking strategy, *prompting* for a student to apply a critical thinking strategy, and *reflecting* after the fact on the critical thinking process.

In the first excerpt, an instructor *models* a critical thinking strategy for debugging. In this exchange, a student's repeat loop is missing its final parenthesis, a bug known as a syntax error. With syntax errors, instructors can directly point them out and offer a way to remember a fix (e.g., introducing a phrase like, "every parenthesis needs a friend"). When this happens, instructors privately enact critical thinking strategies used both to find and fix the bug. In other debugging interactions with students, instructors explicitly model critical thinking strategies used to locate the bug (see Figure 2). In the transcript below, the instructor, Ben, models for the student, Mav, *where* to look for correct syntax and *how* to compare correct and broken syntax.

GRID DIRECTIONS

API

```

1 repeat(3, () => {
2 | repeat(3, () => {
3 | paint('pink')
4 }

```

Error

unknown: Unexpected token, expected ,, (4:1) 2 | repeat(3, () => { 3 | paint('pink') > 4 | } | ^.^
Check the code at line -3.

OK

Mav's screen at start of debugging session

1 Mav runs her code; it generates an error message
2 Ben: Hmmmm. Let's see if it's exactly like we
3 want it to be. (0.2) So let's compare the
4 API. So repeat parenthesis 3 comma
5 parenthesis equal sign (indecipherable)
6 okay and then you have your code that you
7 repeat. Let's look here. Is there something
8 you're missing here?
9 Mav: Ohhh the
10 Ben: Mhmm. Nice. (0.4)
11 Mav adds the final parenthesis to her repeat loop
12 Ben: Yeah I know the syntax is pretty wonky.

Mav's code at the end of this debugging session.

```

2- repeat(3, () => {
3 | paint('pink')
4 | })

```

API

```

paint(color)

repeat(num, () => {
  // code to repeat
})

```

Ben points back and forth between repeat loop in API and repeat loop in Mav's code.

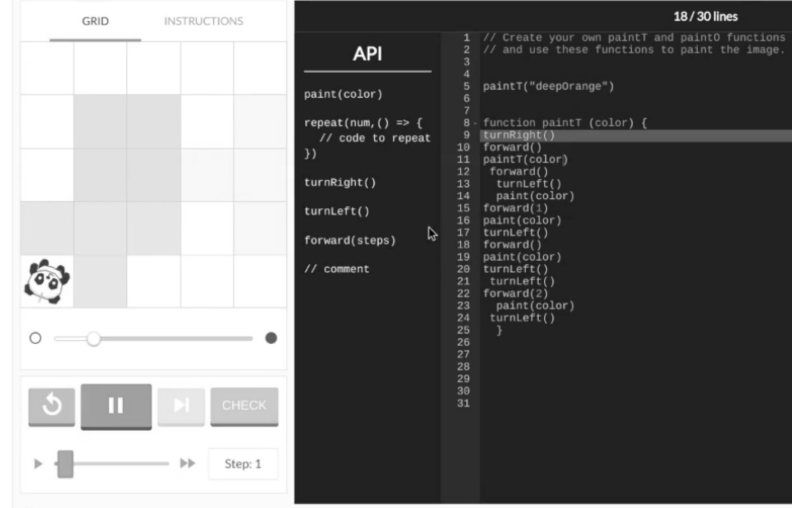
Mav points to the final parenthesis in the API repeat loop.

Figure 2. Modeling a critical thinking strategy: Comparing broken code with working code in the API

The instructor guides the student to a fix while introducing a critical thinking strategy for debugging: visually comparing correct syntax in the API with the student's broken syntax. In terms of critical thinking, this approach explicitly models how to *query* the coding environment for information, *systematically examine* competing approaches to writing code (the API syntax and the student's own syntax), *draw a conclusion* about a missing element, and ultimately *self-correct*. By publicly narrating and showing how to compare broken and accurate syntax, the instructor makes visible a set of critical thinking moves during failure that the student could independently apply to subsequent syntax errors.

Second, we examine how an instructor *prompts* for a student to use a tool in the programming environment that facilitates critical thinking about logic errors: situations in which the program runs but results in output the student does not intend or want. Instead of directly pointing out a flaw in the student's reasoning, the instructor, Jad, prompts the student, Zoa, to use a tool to discover the underlying

cause of the observable problem (see Figure 3). This example also illustrates how the goals students set in their coding journals can motivate the exploration of a debugging strategy.



The screenshot shows a coding environment with a grid on the left, an API section, a code editor with 31 lines of code, and an error message box on the right. The code includes functions like `paint(color)`, `repeat(num, () => {`, `turnRight()`, `turnLeft()`, and `forward(steps)`. The error message says "Error: Out of bounds. Check the code at line 10."

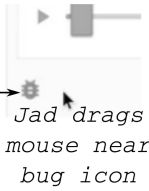
Zoa's code at the start of this debugging session.

1 Zoa runs her code; the Panda goes out of bounds; an error message pops up.
 2 Jad, who had been watching from behind, leans down next to Zoa.
 3 Jad: Zoa can I give you a tip?
 4 Zoa: Yeahh
 5 Jad: What's your debugging goal for today? (0.8)
 6 Zoa: Uhhhhh (3.0)
 8 Zoa starts looking through her journal




Experiment
 So I want to experiment & learn
 takes takes never new things.

9 Jad: Do you remember? (6.0) So - so many goals!
 10 Zoa: Dih (laughing) (1.5) Experiment.
 11 Jad: Experiment. Okay. Um so let's do some experimenting by (1.2)
 12 clicking that bug (1.2) Have you clicked on the bug yet?
 13 Zoa clicks bug icon which pops up her debugging statistics

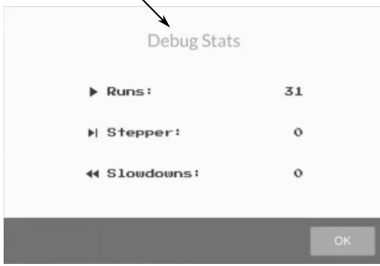


Jad drags mouse near bug icon

14 Jad: What do you think that's showing you? (6.0)
 15 So this shows you that you've run your code
 16 31 times (1.0) How many times have you used the
 17 stepper tool?
 18 Zoa: What's the stepper tool?
 19 Jen: [This thing]
 20 Jad: [Mmmmmmmmmmm] Interesting. That's what I'm
 21 going to show you.
 22 Jad moves the mouse to the stepper button



23 Jad: So let's experiment. (2.8) Go to the -- let's refresh. What you can
 24 do is click step and go one line at a time.
 25 Jad clicks the stepper button once. Zoa takes over and clicks once.
 26 Jad: It'll really help you see where things are going wrong.
 27 Zoa clicks the stepper button once more.
 28 Zoa: Mmmmm.
 29 Zoa stops stepping and starts to erase line 10.
 30 Zoa: I see now.



Debug Stats

- ▶ Runs: 31
- ▶ Stepper: 0
- ◀ Sloudowns: 0

Figure 3. Prompting for use of a critical thinking tool: The stepper

In this example, an instructor offers debugging support by asking the student about her debugging goal for the day and drawing attention to her debugging statistics. Using these two reflective practices as a point of departure, the instructor then prompts for exploration of a critical thinking tool for debugging. The tool, known as a stepper, promotes the integration of a number of critical thinking strategies: *analyzing the argument* or logic in the code line by line, *evaluating* the relationship between that logic and the output of the program (how the PixelBot moves and paints), and *self-correcting* by *drawing a conclusion* about a flaw in the code. Indeed, after prompting the student to try the stepper tool, the instructor in line 26 describes its value: helping the student to independently find (“help you see”) the source of the problem (“where things are going wrong”). Without the instructor describing what has caused the problem, the student is nonetheless able to use the stepper to locate the trouble spot and start to repair it (line 29 in Figure 3).

Our third example (see Figure 4) documents how *reflection* on the critical-thinking process can incorporate debugging goals from students’ journals and take place in conversation just after a bug fix. In the following exchange, the student, Lin, and the instructor, Teo, have already worked together to successfully repair her code when the instructor prompts for reflection.

In this exchange, the instructor prompts for post hoc analysis of prior critical thinking strategies immediately after the bug is fixed. The student provides a label (experiment) for a critical thinking strategy she used, and then the instructor and a peer synchronously highlight another critical thinking strategy (inspect). The group then discusses whether experimenting and inspecting advanced Lin’s debugging goal for the day. Lin describes not meeting one of her goals: focusing. The instructor follows up by describing goal setting as a progressive practice, something we’re “always working on.” This interaction demonstrates how instructors and peers can collaboratively reify a set of actions into overarching descriptions of critical thinking strategies, and then evaluate whether those strategies advance students’ debugging goals. This excerpt documents how collaborative reflection during coding can tie together, reify, and support the planning and enacting of critical thinking.



Lin's code. → *Lin sees that her Minecraft turtle builds a pillar.*

1 Lin: Oh there.

2 Teo: Niiice job. What debugging strategy did you use? (1.0)

3 *Lin stops working and looks up toward the ceiling*

4 Lin: Eh (0.8)

5 Teo: What do you think? (0.4)

6 Lin: Experiment?

7 Teo: Experiment? Yeah. You kind of insp[ected?]

8 Jul: [inspect]

9 *Lin's neighbor, Jul, had leaned over to participate*

10 Jul: [I was gonna say inspect]

11 Teo: [you inspected] And then you experimented, right?

12 Jul: Like

13 Lin: Mhmm

14 Jul: In- inspect [indecipherable]

15 Teo: What was your (0.2) goal today.

16 Ask? [indecipherable]

17 Lin: No this one.

18 *Lin points to different part of journal*

19 Teo: Oh this was today. Experiment? So you experimented. Nice. Did you remain focused?

20 Teo is referencing "focus," Lin's second goal in her journal

21 Lin: (whispering) Kind of.

22 Teo: (laughing) Kind of? It's okay, we're always working on it.



Teo leans in to look at Lin's open journal

Figure 4. Reflecting on critical thinking strategies after fixing a bug.

Overall results of instructional strategies

A number of interview and survey data sources used across our coding workshops suggest that the cumulative impact of our discourse and journaling/art making learning designs (core parts of all of our coding workshops) increased students' sense of their *debugging skill level*, their *confidence* with debugging, and their *awareness of debugging strategies*. With respect to skill level and confidence, students across three coding workshops (Tables 3 and 4) showed the same trend from pre to post survey data on a five-item Likert scale: fewer students reporting having low (terrible, bad, or fine) debugging skill/confidence and more students reporting having high (good, extremely good) debugging

skill/confidence (with average gains, calculated by subtracting the pre-test Likert score (1-5) from the post-test Likert score (1-5) for each student, of .59 in winter 2017; .76 in summer 2017; and .41 in Spring 2018).

Table 3. Survey results about students' debugging skill level

Question: How good are you at debugging code?

	Pre Winter 2017	Post Winter 2017	Pre Summer 2017	Post Summer 2017
Terrible	6.12%	4.08%	11.90%	0%
Bad	14.29%	8.16%	13.01%	3.57%
Fine	48.98%	14.29%	35.71%	21.43%
Good	26.53%	63.27%	30.96%	59.53%
Extremely Good	4.08%	10.20%	8.33%	15.48%
	n = 49		n = 84	

Table 4. Survey results about students' confidence with debugging

Question: How would you feel when you come across a problem with your code?

	Pre Spring 2018	Post Spring 2018
Not at all confident that I can fix it.	0%	0%
Slightly confident that I can fix it.	16.67%	6.25%
Moderately confident that I can fix it.	47.92%	29.17%
Very confident that I can fix it.	22.92%	50.00%
Extremely confident that I can fix it.	12.50%	14.58%
	n = 48	

In addition, we narrowed our focus in Spring 2018 and asked students whether they believed they could create a helpful strategy for debugging (see Table 5). A similar trend emerged in which fewer students reported disagreeing with the statement that they could create debugging strategies and more students reported agreeing with this statement (leading to an average Likert scale gain of .48).

Table 5. Survey results about students' capacity to create new debugging strategies

I can create a helpful strategy to find the problem with my code.

	Pre Spring 2018	Post Spring 2018
Not at all	4.17%	2.08%
Slightly	16.67%	2.08%
Moderately	37.50%	29.17%
Very	29.17%	50.00%
Extremely	12.50%	16.67%

n = 48

Apart from students reporting higher skill level/confidence in debugging and higher likelihood that they could invent debugging strategies, we also gathered evidence that students at the end of the summer 2017 2-week coding workshop could talk extemporaneously about a number of debugging strategies they had used throughout camp. During in-depth end-of-workshop interviews with 20 students, researchers asked: "What is your debugging process -- how do you get rid of bugs?" Every student listed multiple debugging strategies. Overall, students discussed the strategies in Table 6 below.

Table 6. Categorization of students' stated debugging strategies during an end of workshop interview

Percentage of students who described this strategy	List of strategies
5%	<ul style="list-style-type: none"> - Use what I learned - Check the most recent change - Stay calm - Slow down code - Figure it out myself - Classify the bug type

10%	- Comment out code - Focus
15%	- Use the stepper tool
30%	- Automatic Syntax Checker
40%	- Experiment - Look for common syntax errors
50%	- Ask for help - Look at/Re-read my code
60%	- Compare with working code

The table of strategies ranges from specific tools in the coding environment (e.g., stepper tool) and approaches to reviewing code (reread my code) to methods for tinkering with code (e.g., experiment) and cognitive/affective states (e.g., focus and stay calm). The table suggests that students new to coding can start to consider a relatively wide range of critical thinking and affective practices they enact to debug code.

Discussion

Because breakdowns in learning naturally motivate reflection on a complex set of causes and possible resolutions (DeLiema, 2017; Heider, 1958; Herman, 2009; Weiner, 1985), they provide opportunities for communication about the critical thinking process. In this chapter, we described two instructional strategies designed to support communication about critical thinking: (1) student journaling and art making that focused on planning and examining critical thinking strategies and emotional experiences that surround failure; and (2) instructors modeling, prompting for, and reflecting on critical thinking strategies with students during failure to explore both expert debugging heuristics and students' own goals. These designs prioritized public reflection on and communication about the critical thinking process (Collins, Brown, & Newman, 1989), both in stable artifacts (journals and artwork) and during coding. We argue that these teaching strategies are effective because they explicitly bridge *planning*, *reflecting*, and *enacting*, helping ensure that students and instructors actively pursue their plans for critical thinking during failure. Moreover, because this approach values reflection on the phenomenology of failure, or how it feels to encounter a breakdown (Jaber & Hammer, 2016; Sengupta, Dickes, & Farris, 2018), it makes possible conversations in the classroom about how emotion shapes students' selection of critical thinking strategies for failure, an insufficiently examined but important facet of problem solving.

Connections to deeper learning

Even though this chapter did not provide empirical evidence of the capacity of these teaching strategies to promote deeper learning, we would argue that fostering in students a capacity to think critically during failure, including by acknowledging affect and building students' confidence during failure, constitute key elements of pathways that foster high quality learning. To be more specific, pedagogical frameworks such as preparation for future learning (Schwartz & Martin, 2004) and productive failure (Kapur, 2008) have empirically documented the value of failure for long-term, robust learning. However, as Kapur (2008) notes, "learners' frustration thresholds and level of engagement in solving the problem, for example, may be particularly critical" (p. 414). Bringing experiences with frustration and other affective states out of the shadows and into public classroom discourse, whether through journaling or art making, invites *collective* approaches to understanding and supporting how emotion shapes critical thinking. In this way, deep learning may best emerge from failure when a community of teachers and students actively searches for ways to understand and communicate about emotion and critical thinking, rather than when students privately process breakdowns in learning. Moreover, explicitly working to build students' *authority* to debug (Engle & Conant, 2002), including by guaranteeing access to relevant resources, empowers *students themselves* to generate deeper learning from struggle.

Limitations and future work

There a number of limitations to this work at its current stage. First, the link between the proposed teaching strategies and deeper learning is only hypothesized; future work should investigate whether communication about affect and critical thinking surrounding failure promotes robust learning. In a similar way, this study is limited by its focus on select moments of teaching and coverage of aggregate outcomes. More granular, longitudinal case studies, and systematic experimental work, could more meticulously document *how* and *under what conditions* these teaching strategies promote learning. In addition, even though we argued that failure provides a useful point of departure for communication about critical thinking, this assumption warrants inquiry. In particular, future research could explore these teaching strategies around moments of success in the learning process. Lastly, the data used in this study explored a limited dimension of time with respect to communication about failure. Journal reflections took place at the start and end of class, and interviews took place at the end of a 2-week workshop. Future work could explore whether and how student-driven communication about critical thinking strategies for failure, positioned at different points in the learning process (e.g., just before or just after a moment of failure), might promote more relevant or finer-grained planning with implications for the quality of learning.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 1612660, 1612770, and 1607742.

References

- Collins, A., Brown, J. S., & Newman, S. E. (1989). Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics. In L. B. Resnick (Ed.), *Knowing, Learning, and Instruction: Essays in Honor of Robert Glaser* (pp. 453-494). Hillsdale, NJ: Lawrence Erlbaum.
- Dahn, M., DeLiema, D., & Enyedy, N. (in press). Art as a point of departure for understanding student experience in learning to code. *Teachers College Record*.
- DeLiema, D. (2017). Co-constructed failure narratives in mathematics tutoring. *Instructional Science*, 45(6), 709-735.
- Ducassé, M., & Emde, A. M. (1988). A review of automated debugging systems: Knowledge, strategies and techniques. In T. N. Nam (Ed.), *Proceedings of the 10th International Conference on Software Engineering* (pp. 162-171). Los Alamitos, CA: IEEE Computer Society Press.
- Engle, R. A., & Conant, F. R. (2002). Guiding principles for fostering productive disciplinary engagement: Explaining an emergent argument in a community of learners' classroom. *Cognition and Instruction*, 20(4), 399-483.
- Facione, P. A. (1990). *Critical thinking: A statement of expert consensus for purposes of educational assessment and instruction* (ERIC Document Reproduction Service No. ED 315 423). Retrieved from PhilArchive open access website: <https://philarchive.org/archive/FACCTA>.
- Fields, D. A., Searle, K. A., & Kafai, Y. B. (2016). Deconstruction kits for learning: Students' collaborative debugging of electronic textile designs. In P. Blikstein, M. Berland, & D. A. Fields (Eds.), *Proceedings of the 6th Annual Conference on Creativity and Fabrication in Education* (pp. 82-85). New York: ACM.
- Flood, V. J., DeLiema, D., Harrer, B. W., & Abrahamson, D. (2018). Enskilment in the digital age: The interactional work of learning to debug. In J. Kay & R. Luckin (Eds.), *Rethinking Learning in the Digital Age: Making the Learning Sciences Count, 13th International Conference of the Learning Sciences (ICLS) 2018, Volume 3* (pp. 1405-1406). London, UK: International Society of the Learning Sciences.
- Freeman, D. N. (1964). Error correction in CORC, the Cornell Computing Language. *AFIPS Conference Proceedings, Volume 26, Fall Joint Computer Conference* (pp. 15-34). Baltimore, MD: Spartan Books.
- Glaser, B. G. (1965). The constant comparative method of qualitative analysis. *Social Problems*, 12(4), 436-445.
- Goodwin, C. (2018). *Co-operative action*. New York: Cambridge University Press.
- Goldberg, P. D. (2005). Metacognition and art production as problem solving: A study of third grade students. *Visual Arts Research*, 31(2), 67-75.
- Greiff, S., Wüstenberg, S., Csapó, B., Demetriou, A., Hautamäki, J., Graesser, A. C., & Martin, R. (2014). Domain-general problem solving skills and education in the 21st century. *Educational Research Review*, 13, 74-83.
- Haynes, A., Lisic, E., Goltz, M., Stein, B., & Harris, K. (2016). Moving beyond assessment to improving students' critical thinking skills: A model for implementing change. *Journal of the Scholarship of Teaching and Learning*, 16(4), 44-61.
- Heider, F. (1958). *The psychology of interpersonal relations*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Herman, D. (2009). *Basic elements of narrative*. Malden, MA: Wiley-Blackwell.

- Hesslow, G. (1988). The problem of causal selection. In D. J. Hilton (Ed.), *Contemporary science and natural explanation: Commonsense conceptions of causality* (pp. 11–32). Brighton, England: Harvester Press.
- Heyd-Metzuyanin, E. (2015). Vicious cycles of identifying and mathematizing: A case study of the development of mathematical failure. *Journal of the Learning Sciences, 24*(4), 504-549.
- Heyd, T., & Puschmann, C. (2017). Hashtagging and functional shift: Adaptation and appropriation of the #. *Journal of Pragmatics, 116*, 51-63.
- Hmelo-Silver, C. E. (2004). Problem-based learning: What and how do students learn? *Educational Psychology Review, 16*(3), 235–266.
- Ingold, T. (2000). *The perception of the environment: Essays on livelihood, dwelling and skill*. Oxford, UK: Routledge.
- Jaber, L. Z., & Hammer, D. (2016). Engaging in science: A feeling for the discipline. *Journal of the Learning Sciences, 25*(2), 156-202.
- Jordan, B., & Henderson, A. (1995). Interaction analysis: Foundations and practice. *Journal of the Learning Sciences, 4*(1), 39-103.
- Kapur, M. (2008). Productive failure. *Cognition and Instruction, 26*(3), 379–424.
- Katz, I. R., & Anderson, J. R. (1987). Debugging: An analysis of bug-location strategies. *Human-Computer Interaction, 3*(4), 351-399.
- Klahr, D., & Carver, S. M. (1988). Cognitive objectives in a LOGO debugging curriculum: Instruction, learning, and transfer. *Cognitive Psychology, 20*, 362-404.
- Ko, A., & Myers, B. A. (2009). Finding causes of program output with the Java Whyline. In D. R. Olsen & R. B. Arthur (Eds.), *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 1569-1578). New York: ACM.
- Koschmann, T., Kuutti, K., & Hickman, L. (1998). The concept of breakdown in Heidegger, Leont'ev, and Dewey and its implications for education. *Mind, Culture, and Activity, 5*(1), 25-41.
- Langer, S. (1953). *Feeling and form: A theory of art*. New York: Charles Scribner's Sons.
- Lewis, C. M. (2012). The importance of students' attention to program state: A case study of debugging behavior. In A. Clear, K. Sanders, & B. Simon (Eds.), *Proceedings of the Ninth Annual International Conference on International Computing Education Research* (pp. 127-134). New York: ACM.
- Liu, Z., Zhi, R., Hicks, A., & Barnes, T. (2017). Understanding problem solving behavior of 6–8 graders in a debugging game. *Computer Science Education, 27*(1), 1-29.
- Lin-Siegler, X., Ahn, J. N., Chen, J., Fang, F. F. A., & Luna-Lucero, M. (2016). Even Einstein struggled: Effects of learning about great scientists' struggles on high school students' motivation to learn science. *Journal of Educational Psychology, 108*(3), 314.
- Metzger, R. (2004). *Debugging by thinking: A multidisciplinary approach*. Burlington, MA: Elsevier Digital Press.
- Mohammad, S. M., & Kiritchenko, S. (2015). Using hashtags to capture fine emotion categories from tweets. *Computational Intelligence, 31*(2), 301-326.
- Murphy-Hill, E., Zimmermann, T., Bird, C., & Nagappan, N. (2013). The design of bug fixes. In D. Notkin, B. H. C. Cheng, & K. Pohl (Eds.), *Proceedings of the 35th International Conference on Software Engineering* (pp. 332-341). Institute of Electrical and Electronics Engineers Press.
- Oyserman, D., Terry, K., & Bybee, D. (2002). A possible selves intervention to enhance school involvement. *Journal of Adolescence, 25*(3), 313–326.

- Palinscar, A. S., & Brown, A. L. (1984). Reciprocal teaching of comprehension-fostering and comprehension-monitoring activities. *Cognition and Instruction, 1*(2), 117-175.
- Perscheid, M., Siegmund, B., Taeumel, M., & Hirschfeld, R. (2017). Studying the advancement in debugging practice of professional software developers. *Software Quality Journal, 25*(1), 83-110.
- Ripley, G. D., & Druseikis, F. C. (1978). A statistical analysis of syntax errors. *Computer Languages, 3*(4), 227-240.
- Schön, D. A. (1983). *The reflective practitioner: How professionals think in action*. New York: Basic Books.
- Schwartz, D. L., & Martin, T. (2004). Inventing to prepare for future learning: The hidden efficiency of encouraging original student production in statistics instruction. *Cognition and Instruction, 22*(2), 129–184.
- Sengupta, P., Dickes, A., & Farris, A. (2018). Toward a phenomenology of computational thinking in STEM education. In M. S. Khine (Ed.), *Computational Thinking in the STEM Disciplines: Foundations and Research Highlights* (pp. 49-72). Springer International Publishing.
- Suchman, L. A. (1987). *Plans and situated actions: The problem of human-machine communication*. New York: Cambridge University Press.
- Weiner, B. (1985). An attributional theory of achievement motivation and emotion. *Psychological Review, 92*(4), 548–573.
- Zeller, A. (2009). *Why programs fail: A guide to systematic debugging*. Burlington, MA: Elsevier.