

UC Berkeley

UC Berkeley Previously Published Works

Title

A Multi-dimensional Framework for Documenting Students' Heterogeneous Experiences with Programming Bugs

Permalink

<https://escholarship.org/uc/item/7gd6n6zd>

Journal

Cognition and Instruction, 41(2)

ISSN

0737-0008

Authors

DeLiema, David
Kwon, Yejin Angela
Chisholm, Andrea
[et al.](#)

Publication Date

2023-04-03

DOI

10.1080/07370008.2022.2118279

Copyright Information

This work is made available under the terms of a Creative Commons Attribution License, available at

<https://creativecommons.org/licenses/by/4.0/>

Peer reviewed



A Multi-dimensional Framework for Documenting Students' Heterogeneous Experiences with Programming Bugs



David DeLiema^a , Yejin Angela Kwon^b , Andrea Chisholm^c , Immanuel Williams^d,
Maggie Dahn^e , Virginia J. Flood^f , Dor Abrahamson^g , and Francis F. Steen^h 

^aDepartment of Educational Psychology, University of Minnesota; Minneapolis, Minnesota, USA; ^bDepartment of Mathematics, University of California, Berkeley, California, USA; ^cDepartment of Cognitive Science, University of California, Berkeley, California, USA; ^dStatistics Department, Cal Poly San Luis Obispo, California, USA; ^eDepartment of Informatics, University of California, Irvine, California, USA; ^fDepartment of Learning and Instruction, University at Buffalo the State University of New York, Buffalo, New York, USA; ^gSchool of Education, University of California, Berkeley, California, USA; ^hDepartment of Communication, University of California, Los Angeles, California, USA

ABSTRACT

When teachers, researchers, and students describe productively responding to moments of failure in the learning process, what might this mean? Blending prior theoretical and empirical research on the relationship between failure and learning, and empirical results from four data sets that are part of a larger design-based research project, we investigate the heterogeneous processes teachers and students value and pursue following moments in which computer bugs thwart their immediate progress on an activity. These include: (1) resolving moments of failure; (2) avoiding recurring failures; (3) preparing for novel failures; (4) engaging with authority; and (5) calibrating confidence/efficacy. We investigate these processes taking into account the personal, social, and material context in which students and teachers collaborate when encountering broken computer programs, in addition to teachers' planning efforts and the community's reflections on past debugging experiences. We argue that moments of failure are not simply occasions for seeking resolutions. They are points of departure for decisions about how and what to foreground and interleave among a range of valued processes. Overall, this study aims to support research on the heterogeneous processes that shape how students new to a discipline such as computer programming respond to getting stuck.

Learning trajectories are commonly punctuated by stretches of difficulty. In the research literature, they are variably described as impasses (VanLehn, 1988), breakdowns (Koschmann et al., 1998), snags (Lave et al., 1984), and/or failures (Kapur, 2008), and they can drive long-term learning (Kapur, 2016; Sinha & Kapur, 2021). Acknowledging this potential, in this paper, we investigate the heterogeneous processes teachers and students value and pursue following moments in which their immediate progress on an activity is thwarted by friction, problems, obstacles, etc. To lead with a student's perspective on these processes, we highlight sixth grade Zoa's¹ reflections on debugging during a two-week Summer computer science (CS) workshop. Zoa's reflections during and following coding sessions, including in the art piece featured below (see Figure 1), cover substantial ground. Across the two weeks, instead of attending to a single process following moments of failure, Zoa comments on self-efficacy ("I'm slowly dying. I got this," "doubted myself"), strategies for debugging ("compared my work," "experiment with my code"), collaboration ("my

CONTACT David DeLiema  ddeliema@umn.edu  Department of Educational Psychology, College of Education and Human Development, University of Minnesota, Room 152 EdSciB, 56 East River Road, Minneapolis, MN 55455, USA.

¹All names are pseudonyms

© 2022 Taylor & Francis Group, LLC

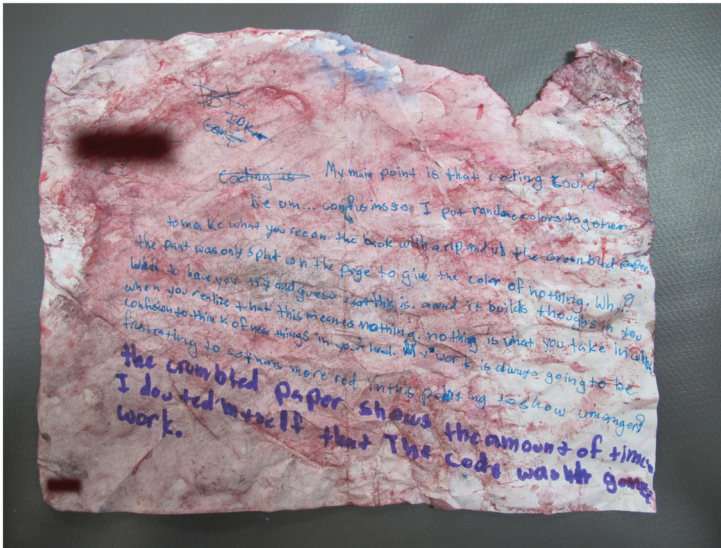


Figure 1. Zoa's abstract artwork about the emotional experience of debugging.

friend helped”), failure recurrence (“cuz then I’ll forget”), and naturally, whether a resolution emerges (“kill da bugs,” “going to work”).

In the vein of recognizing these varied experiences with failure, and drawing inspiration from learning sciences research that embraces heterogeneity (e.g., Rosebery et al., 2010), we ask: When teachers, researchers, and students describe productively responding to moments of failure in the learning process, what might this mean? Zoa’s reflections allude to the complexity of this issue. Toward the end of our first of three years of data collection on a design-based research project (DeLiema et al., 2020), as we investigated the multi-dimensionality of failure, we reflected on its characteristics and noted five processes following moments of failure that teachers and students persistently valued and pursued. In reviewing prior literature addressing the relationship between failure and learning, we found each of these represented by a rich empirical and theoretical literature:

1. *Resolving moments of failure:* Arriving at a point at which an impasse is considered resolved (Roschelle, 1992; Russ & Berland, 2019).
2. *Avoiding recurring failures:* Taking action to limit the chance the “same” impasse returns and/or being better prepared to find a resolution should it recur (Barnett & Ceci, 2002; Kapur, 2016; Reason, 1990).
3. *Preparing for novel failures:* Learning general processes for fixing new (not-yet-recurring) impasses (Katz & Anderson, 1987; Pea et al., 1987).
4. *Engaging with authority:* Contributing with voice and action to the process of handling the impasse (Engle & Conant, 2002; Foster, 2014; Langer-Osuna, 2016).
5. *Calibrating confidence/efficacy:* Gauging individual and collective expectations about capacity to resolve the impasse (Bandura, 1982; Kallia & Sentance, 2018).

In preparing for our second year of data collection, our design-based research team formulated the above framework to give ourselves a comprehensive target to consider when designing, documenting, and evaluating our approach to supporting students following their encounters with programming bugs. In this paper, we extend our inquiry into four empirical analyses of multi-week 5th–10th grade coding workshops. The framework surfaces and labels “underlying

regularities” (Engle & Conant, 2002, p. 401) within students’ and instructors’ day-to-day efforts to debug code. By abstracting these underlying regularities into a framework, we hope it is general enough to inform educational researchers’ collective efforts to understand how and when students and instructors pursue and balance these distinct facets of debugging following learning impasses, with implications for how to support newcomers to a practice. Central to this framing of failure is an awareness that computer programming teaching and learning occur in large part in spaces outside of the code artifacts themselves. These spaces have often been overlooked in the CS education research literature. In their charge to the field to move past this technocentrism, Sengupta et al. (2021) note that “the complexity of teaching in classroom contexts that include coding has barely been investigated” (p. 4), largely because studies of coding have inferred what is valuable to students “primarily—in many cases, exclusively—from the code created by the students rather than from observations of their experiences” (p. 6). In the present study, we join a growing community of scholars who are centering the conversational process of talk, gesture, artifact use, and movement between teachers and students during debugging (Flood et al., 2018; Hennessy Elliott, 2020; Heikkilä & Mannila, 2018; Jayathirtha et al., 2018; Sengupta et al., 2021; Silvis et al., 2022; Steinberg & Gresalfi, 2021; Wang et al., 2021).

The paper begins with a literature review of each of these five processes to argue for their centrality to failure and to capture diverse considerations around each. We then apply the multi-dimensional framework in multiple empirical contexts to investigate how instructors design with these facets of failure in mind; how a student, and their classroom instructors and peers, foreground each during coding; what patterns/regularities occur in one student’s coding practice; and how instructors and students reflect post-hoc on each process. Altogether, the framework gives us some purchase in understanding students’ heterogeneous experiences with failure in the early stages of learning a new practice. Finally, we comment on directions for future research building on this framework.

Literature review and theoretical framework

We use the term failure to refer to an event in the learning process that participants consider to be a breakdown or impasse en route to a goal (Koschmann et al., 1998), however fleeting or lasting the impasse becomes. More specifically, in our research context, we consider both brief and prolonged situations in which students notice an adverse outcome when running their code and then either attempt to change their code to alter that outcome or abandon/shift their goal. We interchangeably use failure, impasse, and bug to refer to these moments of breakdown.

A core area of exploration around failure is educational researchers’ appeal to the productive potential of failure to spur learning, what Gomoll et al. (2018) described as “positioning failure as a constructive activity” (p. 91). For example, Manalo and Kapur (2018), in their introduction to a special issue on failure and creativity, noted that “experiences of failure can and should be utilized in more positive and productive ways” (p. 1) and that students should “persist in the face of failure (and hence make it possible to benefit from such experiences)” (p. 2). In a stark echo of this perspective, an educator quoted in Maltese et al. (2018) described student success and failure in the following way: “Failure is when they give up. Success is when they persist” (p. 2018). Indeed, researchers have substantively examined student persistence following failure (e.g., Duckworth et al., 2007), though not without powerful critiques that this research neglects context, values persistence only in some areas, and eschews unequal access to resources (McGee & Stovall, 2015; Rose, 2015). An even more recent line of research adopted a meta perspective on this topic by considering *beliefs* about whether failure is productive to learning, classified as either a “failure-is-enhancing” or “failure-is-debilitating” mindset (Haimovitz & Dweck, 2016).

In response to this research, we reflected on what might make failure “constructive,” “productive,” or “enhancing,” specifically in terms of the learning processes that take place

following computer bugs. In the research literature, we found heterogeneous perspectives on this topic. Though not exhaustive, the framework we propose includes five processes that are routinely studied as central parts of post-impasse learning. In our review of each below, we address prior research on failure and learning across domains, paying particular attention to computer science, our focal domain, and we explicitly document a range of perspectives on each process.

Resolving moments of failure

Teachers and students often work to arrive at resolutions to impasses. For example, early computer programming educational research set a precedent for valuing revisions to programming learning environments that allow students to get more bugs fixed, or “reduce the volume of these errors” (Freeman, 1964, p. 15). A common outcome measure in debugging research is whether students become more efficient at fixing bugs (e.g., Ko & Myers, 2004). Students themselves might strongly orient to this goal. For example, in an after-school program focused on science, engineering, and the arts, students preferred to have researchers video record their “finalized projects (or moments they identified as successful)” (Vossoughi & Escudé, 2016, p. 54). Similarly, in the context of video game play, some players viewed impasses in games as failure only when the impasse persistently prevented them from arriving at a resolution: “These players, after numerous attempts at the same stage of a level, felt as if they hit a wall that they couldn’t get past” (C. G. Anderson, 2020, p. 4). In many ways, arriving at a resolution following a moment of failure is a deeply held value. In a recent study drawing on feminist epistemologies, ethics of care, and awareness of environmental sustainability, Silvis et al. (2022) generatively asked: “What responsibilities do children assume for ensuring that coding robots are in working order?” (p. 1). Studying the conversations between children and their teachers following failures and risks for a robot toy, Silvis et al. saw how children viewed technologies as “worthy of care” (p. 7) and thus deserving of repair and maintenance. Moreover, this resolution process is valued outside of child-focused learning settings. Actions by corporations that downplay or deny foundational problems are understandably problematic (Reason, 1990). Even more fundamentally, the value of fixing problems extends into the social fabric of conversation, where speakers and listeners work to swiftly “repair” problems with speech production and comprehension (see Kitzinger, 2012 for a review). In learning settings, we know a great deal about processes that tend to lead to resolutions. Two recent studies are worth highlighting. In a science gaming context, early and numerous failures led to ultimate success because moments of friction spurred “collaborative discourse and resulted in a focus on mechanics” relevant to the target science (C. G. Anderson et al., 2018). In this way, latent success does not derive simply from amassing failures but from incrementally improving from one impasse to the next (Yin et al., 2019).

In contrast, learning communities can de-emphasize normatively correct resolutions. In some kindergarten classrooms, for example, teachers explicitly respond to students in a way that signals that “all answers are accepted” (Donaldson, 2019, p. 7). And yet even when swift accuracy is not expected, researchers and educators may still value “latent productivity” or “quality” (Kapur, 2008). For example, in a study of how two high school students work toward conceptual change, Roschelle (1992) attended to accuracy as an eventual achievement of collaborative work: “the students’ conception changed in ways that made it more compatible with a scientific interpretation of velocity and acceleration” (Roschelle 1992, p. 264). Relatedly, positive responses to failure may hinge “on whether ‘success’ was *eventually* attained by meeting an internal or external goal” (Maltese et al., 2018, p. 120, our emphasis).

These observations draw attention to a fundamental tension between the (longer-term) value of overcoming an impasse and the (shorter term) value of giving students space to formulate their approach to an impasse. Russ and Berland (2019) squarely examined this tension by noting the difficulties of framing inaccurate resolutions to problems as nonetheless valuable, presenting “a

framework for describing the activity of science learning that reduces this slippage by giving knowledge construction true priority over the canon” (p. 279). With more slack in the collective expectation that students precipitously arrive at a fix to a problem, educational researchers are increasingly recognizing how students “positioning themselves as not-understanding” contributes productively to classroom discourse (Watkins et al., 2018, p. 576). This line of research on uncertainty raises questions about what counts as a resolution to an impasse. In the history of mathematics, “success and failure were not predetermined, but up for debate,” a form of “negotiable failure” (Trninic et al., 2018, p. 77). Points of view within modern feminist theory (Halberstam, 2011) and within the Japanese ethos of *wabi-sabi* (Maheux, 2016) embrace this fuzzy space, and even relish the presence of flaws, an orientation to failure familiar to some maker space communities (Martin, 2015). A related orientation to failure occurs during play, a stance toward activity defined by the paradox that we typically avoid failure and yet within play, often prefer it even as we work to overcome it (Juul, 2013).

In summary, an extensive literature documents a range of considerations around how students respond to impasses with efforts to find a resolution. How often students fix problems and how much time elapses during the fixing process are central considerations. In addition, there is perhaps growing tolerance for honoring the moment of breakdown itself prior to finding a fix, and recognizing the subjectivity (and productivity) of determining whether a resolution has been found.

Avoiding recurring failure

The second process we consider is avoiding recurring impasses. Students and instructors often invest time not only in resolving a here-and-now impasse, but also responding in a way that staves off the likelihood of recurrence. Jordan and Henderson (1995) succinctly capture this value in their reflection on conversational repair: “People learn by experience which kinds of troubles *tend to recur* and what range of resources can be assembled and held available for their solution” (p. 71, our emphasis). Efforts to guard against recurring failure arise in many contexts. When responding to trauma, such as a fight with cancer, people invest time pushing back “not only against present threats but also against possible future setbacks” (Taylor, 1983, p. 1161), specifically aiming to “forestall a recurrence” (p. 1168). These dynamics can also be studied as a collective accomplishment, in which learners “are exposed to others’ errors and are thus prevented from making errors of their own” (Wong & Lim, 2019, p. 2). This issue is relevant to educational research because of persistent, “relatively permanent bugs” (VanLehn, 1988, p. 10). In computer science, Spohrer et al. (1985) describe these as bug dependencies: “the situation in which a single underlying misconception gives rise to several bugs in a single program” (p. 184). Impasses may signal underlying conceptions that will re-generate impasses unless addressed, and indeed, CS education researchers explicitly study bug recurrence and persistence (Smith & Rixner, 2019).

Guarding against recurrence requires learning about the impasse. Jeffries (1982) described how responding to programming impasses can catalyze a process of understanding “the nature of the misconception” (p. 2). More directly, Endres (1975) raised questions about the connection between understanding causes of impasses and preventing impasses: “Why was the particular error made? What caused the error? Closely related to this (as we will show later) is the question: What could have been done to prevent this particular error?” (p. 329). Pea et al. (1987) similarly considered pedagogical implications around the “prevalence and systematicity of student bugs” (p. 24). What makes guarding against failure recurrence challenging for educators and students is the reality that a vast number of causes accumulate to cause breakdowns (Hesslow, 1988; Ko & Myers, 2005). Multiple activities and systems, whether internal or external to the student, and each containing their own affordances and constraints, layer together to make possible a

“trajectory of accident opportunity” (Reason, 1990, p. 208) that would need to be understood to prevent recurring problems.

Perhaps because of these factors, there is a persistent tension between taking quick routes to fixing a problem and slower routes to developing enough understanding to avoid failure recurrence. One of the enduring statements on this balancing act comes from Cazden (1981). Cazden’s account of “performance before competence” captures how expert support that helps a newcomer solve a problem in the short term does not necessarily prepare the learner for the next, similar problem that arises. As Manalo and Kapur (2018) note: “While some of our failed efforts may clearly lead to useful lessons and more successful subsequent attempts, there are also many other failure experiences from which we apparently learn nothing” (p. 1). In a reversal of this tradeoff, the preparation for future learning framework (Bransford & Schwartz, 1999) and productive failure approach to design (Kapur, 2008) concede to short-term failure in service of generating deep learning relevant to related, upcoming problems. Either way, awareness of “recurring” problems raises questions about learning transfer. How similar to a previous impasse does a current impasse need to be to count as a recurring issue? Barnett and Ceci’s (2002) transfer taxonomy provides a number of dimensions to guide this inquiry, such as recognizing the same physical and functional context across several days, different social contexts, and different task constraints.

In summary, following impasses, members of learning communities often work to prevent these impasses from recurring, or at least to be in a better position to handle them should they re-arise. A middle school student from our computer science workshops described this dynamic in her debugging practice: “Writing down notes helps me and also creating like little summaries of what I did to fix codes *so that in the future if I run into the same problem* I can just look back into my notes and say oh that’s what I did” (our emphasis). Toward this end, researchers and educators have considered the permanence of bugs, how the diagnosis of causes of impasses informs actions to prevent impasses, and how short-term failure can nonetheless prepare students for similar problems.

Preparing for novel failures

A third process frequently enters the picture following an impasse: learning general skills to resolve upcoming, novel impasses. These skills are often described as debugging, troubleshooting, or problem-solving processes. An early example comes from Polya’s (1945) description of problem-solving questions and practices that students can use to “solve *future* problems” (our emphasis, p. 4) for which the students do not have ready-at-hand strategies. These practices have been studied across a number of academic domains. In the context of reading, Palinscar and Brown (1984) described and taught specific practices for when readers “are attempting to overcome a comprehension failure (debugging state)” (p. 119). In the clarification process of their reading framework, readers monitor their comprehension failures and follow up by re-reading a previous section or clarifying the difficulty. Similarly, the science modeling literature recognizes the value of teaching the practice of iterating on models that initially fail to predict the target phenomenon. This practice involves developing overarching skills of evaluating models, testing models against other ideas, and revising models (Schwarz et al., 2009). Similarly, in the cognitive apprenticeship framework, Collins et al. (1988) introduced control strategies for monitoring whether a problem exists and then learning “how to select among the various possible problem-solving strategies, how to decide when to change strategies, and so on,” especially “if one is stuck” (Collins et al., 1988, p. 14–15). In the context of outdoor play, Baker et al. (2022) examined how a parent—supporting a child to reach a high rope—asked the child to assess the efficacy of a proposed fix, thus interleaving a broadly valuable debugging skill into the conversation. In each case above, learners develop skills capable of helping them get unstuck when novel impasses arise.

In computer science, debugging skills are important because impasses are common, and at times challenging, parts of the practice (Murphy-Hill et al., 2013; Perscheid et al., 2017). More directly, empirical work demonstrates that these skills need to be nurtured (Klahr & Carver, 1988, p. 377). While the term “debugging” has served as a catch-all for any facet of fixing code, there is a literature around the more narrow facet of debugging that we are highlighting. For example, experienced programmers have written volumes that articulate effective processes and tools for debugging unknown, especially taxing problems in code (Metzger, 2004; Zeller, 2009). There is also considerable knowledge of how modern professional programmers experience debugging, including how often bugs arise, what tools are used, and how debugging teaching and learning in professional communities take place (Beller et al., 2018; Murphy-Hill et al., 2013; Perscheid et al., 2017).

In addition, we know a good deal about how novice programmers approach debugging (Katz & Anderson, 1987; Liu et al., 2017). For example, there are specific routes students take when writing and repairing programs, such as tinkering (Berland et al., 2013), and these routes can predict long-term learning (Blikstein et al., 2014). A significant research thread in this vein is directed at building tools to help new programmers find and fix bugs (Hristova et al., 2003; Ko & Myers, 2009; Lazar et al., 2018; Lee et al., 2018; Miljanovic & Bradbury, 2017). There is even a comprehensive guide to help newcomers learn the nuts and bolts of debugging (Lysecky & Vahid, 2018). Related research threads have focused on what distinguishes novice debugging from expert debugging (Vessey, 1985), how teachers new to programming instruction debug code (Kim et al., 2018), and whether students can transfer computer science programming and debugging skills (Klahr & Carver, 1988; Pea & Kurland, 1984).

In summary, there are a number of specific considerations around the teaching, learning, and practice of debugging. In contrast to fixing a here-and-now impasse and preparing for recurring impasses, debugging involves learning skills to handle unforeseen, novel impasses. Learning to debug means learning broadly relevant “strategies for dealing with errors” (Reason, 1990, p. 245). Central factors include strategies for debugging, tools for debugging, processes of teaching and learning debugging, knowledge of bug types to guide where to develop a debugging practice, and the extent to which debugging skills transfer to new domains.

Engaging with authority

A fourth consideration around how students respond to impasses is the extent to which students are involved actively in the repair process. Regardless of whether a learning activity foregrounds arriving at a fix, preparing for deep learning to avoid a specific recurring impasse, or teaching debugging strategies useful for a number of unknown future bugs, learning communities often value that students’ voices and actions consequentially shape the repair process. This value is powerfully captured in work on students’ rightful presences in the classroom (Calabrese Barton & Tan, 2019). In our understanding of authority, we also build explicitly on the *productive disciplinary engagement* framework, in which Engle and Conant (2002) referred to authority within the context of impasses: “Part of such authority is a matter of students having an active role, or agency, in defining, addressing, and resolving such problems” (p. 404). We further recognize Langer-Osuna’s (2016) distinction between intellectual authority (positioning that addresses the credibility of an information source) and social authority (capacity to shape the division of labor and actions during collaborative work). Relatedly, Adair (2014) defined agency as the capacity “to influence and make decisions about what and how something is learned” (p. 219), and rightly drew attention to how “standardized childhood” classrooms (Fuller, 2007) disproportionately stifle agentic opportunities for students from marginalized communities. Interleaving this prior work, we view authority as students having an active role, both intellectually and socially, in determining how bugs are resolved in their code.

In the computer science educational research literature, a facet of authority is addressed in research on debugging independence. Pea et al. (1987) distinguished the collaborative process of repair during human conversation from situations in which the programmer works alone at a computer: “Whereas the ‘debugging’ of natural language discourse is socially accomplished, and often inexplicit, the novice programmer must *go it alone* in total explicitness” (p. 8, our emphasis). In newer threads of work, learning scientists studying electronic textiles have quantified situations in which students “largely worked by themselves (55.6%) to troubleshoot challenges of crafting with conductive thread” in contrast to receiving teacher support (Searle et al., 2018, p. 130). Debugging research on whether students can “correct their faulty code themselves” (Lee et al., 2018, p. 1) aligns with currents outside of computer science research. For example, the child development literature covers autonomy support (Meuwissen & Carlson, 2019; Whipple et al., 2011) through attention to parenting practices in which children are “given the opportunity to solve problems on their own” (Bindman et al., 2015, p. 757; see also Baker et al., 2022). A related pedagogical framework in mathematics instruction, focused on minimal interventions, recruits empathetic listening in service of “the student generally find[ing] some kind of resolution of their own” (Foster, 2014, p. 152). The other end of this division of labor has been studied in part through the construct of helicopter parenting, a dimension of which addresses how the parent “solves any crisis or problem [the child] might have” (Schiffrin et al., 2019).

However, we would caution against reductions of authority exclusively to independence. Complex divisions of labor exist in both large organizations (Cole & Engeström, 1993) and in dyadic settings such as math tutoring interactions (DeLiema, 2017), and there are calls to reject the false binary of adult- versus child-centered pedagogy (Vossoughi et al., 2021). In addition, peripheral participation serves as a legitimate form of action and mode of learning (Lave & Wenger, 1991). Indeed, the construct of rightful presence (Calabrese Barton & Tan, 2019) foregrounds authority within the context of learning communities. A related thread of research on this topic proposes the construct of fluid synchrony (Mejía-Arauz et al., 2018). These collaborations involve “fluently attuned participants who provide relevant contributions in a shared endeavor oriented toward a shared goal” and can shape the flow of work during pair programming (Mejía-Arauz et al., 2018, p. 117). By not equating authority with independence in these ways, we can continue to recognize how students within collaborative participation frameworks have power with voice and action to shape the approach to resolving impasses. In this way, Flood et al. (2018) documented how instructors can make space for learners to engage in debugging practices in ways that preserve their agency in problem solving.

In summary, there is a central place for research on failure that attends to how students develop the skill to navigate the debugging process, not only in the sense of working alone, but also in having and building capacity to select, enact, defend, prioritize, and expand their debugging process while working with others.

Calibrating efficacy

A final consideration around how students respond to impasses is the process of deciding whether arriving at a resolution is feasible. This consideration is well captured within the construct of self-efficacy: “Perceived self efficacy is concerned with judgments of how well one can execute courses of action required to deal with prospective situations” (Bandura, 1982, p. 122). We note that perceived efficacy (whether by the individual or collective) is a conjecture or judgment about capacity to handle a prospective set of conditions. Because of the uncertainty that comes with some impasses, judgments about capacity to navigate the conditions following an impasse matter a great deal. Indeed, Bandura (1982) mapped perceptions of efficacy specifically to failure, noting that efficacy judgments shape how much people continue to engage “in the face of obstacles or aversive experience” (p. 123). In addition, decades of research suggest that efficacy

judgments following impasses are key factors in shaping the motivation to continue ahead (see Eccles & Wigfield, 2002 for an overview).

In the computer science (CS) education literature, researchers have attended to the relationship between perceived efficacy/confidence and debugging. For example, high perceived self-efficacy among undergraduate students in an introductory programming course correlated with successful debugging (Wiedenbeck, 2005). More broadly, a grounded theory analysis of undergraduate students' reflections on learning to program revealed that efficacy was a frequent topic, especially when students described "encountering difficulties," "dealing with difficulties," and "succeeding" (Kinnunen & Simon, 2011). A few studies have found that the experience of pushing through an impasse and ultimately succeeding generates impressions of high efficacy, such as "boosts in self-confidence" following successful debugging (Maltese et al., 2018, p. 120) and higher efficacy ("I didn't feel as though I could do as much before") following learning from failure (Estabrooks & Couch, 2018, p. 112). However, Kinnunen and Simon (2012) also recognized "a somewhat surprising combination of positive programming episode and negative self-efficacy judgment" (p. 23), in which students critically evaluated the pace and effort of their debugging process.

Bracketing off for a moment the factors that predict and result from perceived self-efficacy, we note that an underlying assumption of this research is often that high self-efficacy is valued. For example, Michaeli and Romeike (2019) explicitly noted that "fostering students' debugging self-efficacy and eventually self-reliance is essential" (p. 6). While this goal might be valuable in the long run, an alternative perspective on this topic recognizes that efficacy judgments, even perceived low efficacy, could be helpful to the process of goal and strategy revision. As Bandura (1982) noted, *enacted* efficacy is a complex notion: "operative competence requires orchestration and continuous improvisation of multiple subskills to manage ever-changing circumstances" (p. 122). The process of forming accurate calibrations of efficacy has to account for that complexity, and in many cases, may pick up on important information. For example, low perceived self-efficacy has been found to correlate with programming "misconceptions" (Kallia & Sentance, 2018), which suggests that efficacy judgments can productively signal conceptual areas worthy of learning investment. More fundamentally, learning success is contingent on external circumstances, such as having access to resources, time, expert pedagogy, and buffers against failure, among many other supports, which have been unequally distributed on a persistent basis (McGee & Stovall, 2015; Rose, 2015). Thus, instead of adopting the position that all impasses during learning are necessarily resolvable, one might attend to the informative process of how students and instructors calibrate confidence as part of the process of demanding accountability from multiple stakeholders to support student learning. Drawing on a similar set of considerations as noted in the previous section around divisions of labor in collaborative activity, we would also caution against the reduction of perceived efficacy to *self*-efficacy, and instead, propose to attend additionally to perceptions of *collective*-efficacy, or how people judge their capacity to "resolve their problems ... during concerted effort" (Bandura, 1982, p. 143).

In summary, students formulate judgments about their capacity to navigate prospective learning experiences, especially following impasses. These perceptions of self- and collective-efficacy have been found to impact students' motivation to persist following impasses, and importantly, may provide key signals that internal and external resources are misaligned with task expectations. Because enacted efficacy itself is a complex process, we view the process of judging efficacy as requiring calibration, perhaps through moments of holding judgments in "provisional status" (Bandura, 1982, p. 124). These dynamics can be studied as the evolving understanding of connections between resources/skill and the conditions following an impasse.

Heterogeneous processes surrounding failure

In the literature review above, we make the case that each of five processes is central to what instructors, students, and researchers have valued and pursued following impasses during

learning. Despite substantial research around each process, we argue that our field would benefit from a deeper understanding of whether and how these processes are foregrounded, backgrounded, and blended when impasses arise during teaching and learning. This call resembles a construct introduced to describe students' failure as mosaic mindsets (Simpson et al., 2018), which entail "viewing failure as either positive or negative based on context" (A. Anderson et al., 2019, section "Failure descriptions"). We extend the mosaic metaphor here to frame failure moments as pluripotent, capable of moving in multiple directions. That is, participants invested in a goal disrupted by an impasse may choose to actively pursue a resolution to the impasse, prepare to avoid the impasse in the future, use the occasion to develop general debugging skills, involve the contributions of newcomers, or weigh their efficacy to resolve the problem, including combinations of these processes. This framing of failure draws inspiration from Sengupta et al.'s (2021) mapping between computer programming and Bakhtin's scholarship on language (Bakhtin, 1986). In bridging computational utterances and Bakhtin's account of voice as filled with tension, representative of heterogeneous speakers, and lacking neutrality, Sengupta et al. argued that when "we begin to see coding as conversations between students (and teachers), the complexities of negotiating multiple perspectives inherent in such conversations offer both opportunities and challenges for engaging with the code and the simulation" (Sengupta et al., 2021, p. 30). When studying debugging in the context of classroom conversation, we conjecture that debugging is similarly a tension-filled activity in which teachers and students negotiate what to foreground and value among distinct processes following a moment of failure.

Taking this possibility as a point of departure, we ask four research questions: (1) To what extent do computer science instructors plan instruction around bugs with these five processes in mind? (2) How does one student, in collaboration with peers and instructors, engage with each process throughout and following debugging? (3) What patterns/regularities around each process occur in one student's coding practice over a sustained period? (4) When prompted explicitly, how do instructors and students reflect post-hoc on these five processes? By acknowledging heterogeneous processes surrounding failure, we aim to start to understand the extent to which these processes are intermingled during programming teaching and learning.

Method

This paper reports on the results of a series of analyses as part of a design-based research project (Dahn & DeLiema, 2020; DeLiema, 2017; DeLiema et al., 2020). The site for this research is a nonprofit community learning center on the West Coast of the United States in a large city, and the design-based research participants included researchers (graduate students, a postdoctoral researcher, and multiple faculty with expertise in CS, art making, and educational research) and practitioners (CS teachers, workshop organizers, software developers). Research iteratively moved through quick cycles of data collection/analysis and redesign taking place during multiple 8-session weekend programs throughout the year (Fall, Winter, and Spring), followed by stable Summer implementations. Comprehensive data collection took place during these stable implementations at a two-week workshop in 2017 (Summer 1) and a three-week workshop in 2018 (Summer 2), each of which ran five days a week, 9am–4pm. The research reported in this paper comes from Summer 1 and Summer 2.

Framework timeline

The framework we propose solidified between Summer 1 and Summer 2. Several factors shaped the framework: (1) a literature review of research on failure, debugging, and learning; (2) considerations from our grant-funded advisory board about how to operationalize "productive" debugging; (3) awareness of recurring themes that grounded our research-practice partnership

conversations about the design of the learning space ahead of Summer 1; and (4) bottom-up analyses of teacher-student interactions during debugging. With a preliminary framework in mind, we addressed RQ1 and RQ2 by analyzing existing data from Summer 1, and we addressed RQ3 and RQ4 by explicitly designing Summer 2's workshop and measures around the framework.

Participants

The 5th–10th grade students participating in the program came from schools across the city. Student demographics varied across workshops, though most students identified as Latino/Latina, and small proportions of students identified as Asian, Black, and White, including students who identified with multiple categories. Students demonstrated either financial need or attendance at a Title 1 school to enroll in the free program. There was slightly more demand than room available at each workshop; students were admitted if they had attended past coding workshops in the space or if their workshop application demonstrated high interest. The student data for our analysis come from a classroom of 18 students (RQ 2, Summer 1) and two classrooms totaling 40 students (RQs 3 and 4, Summer 2). Each classroom contained about an equal blend of students new to programming and students with some prior programming experience. Instructors in these classrooms were mostly undergraduate students majoring in computer science; a few were recent graduates working in tech careers. All were at the beginning stages of developing their teaching practice. Eight instructors participated in the professional development sessions around conjecture mapping, described below (RQ 1). The two instructors followed in Summer 1 for RQ 2—Mia and Jad—were both undergraduates pursuing CS majors at universities in the area: Mia was in her second year of her program, and Jad was in his third. Both instructors had also been teaching in the weekend program of this workshop during the Fall, Winter, and Spring program. An author of this paper (DeLiema) served as the researcher in this classroom and spent most of his time taking field notes during each class but occasionally circulated the room during coding time to help students with debugging. In Summer 2, six instructors—including Mia, two authors of this manuscript (DeLiema and Dahn), and three additional instructors, all of whom had experience teaching at the site—were part of the sample for RQs 3 and 4.

Workshop design

The classroom parts of this data set involved working in PixelBots, a programming environment in which students used JavaScript to control the movement and painting actions of an animal avatar around a two-dimensional grid. PixelBots takes advantage of a central design principle of Logo (Papert, 1980) by having students program the movement and painting actions of an animal avatar on a grid. The PixelBots software was designed to offer a number of features to support students' debugging (see [Figure 2](#)). The Summer 1 curriculum invited students to learn how to write functions that could paint different shapes, building toward a Picasso-like cubist painting that creatively decomposed an object into abstract, recurring shapes. Students explored core coding concepts along the way, including how to call functions, how to sequence code, how to add arguments to function calls, how to use loops, how to create functions, and how to create multi-parameter functions. The first six days of the workshop were a mix of whole-class discussion about these concepts and practice implementing the concepts on self-paced playlists. Students then prototyped their final artwork and spent three days coding before sharing their pieces at a final peer showcase. In Summer 2, the PixelBots curriculum covered the same concepts but emphasized challenges instead of a cumulative project. Motivated by our research team and instructors' reflection on the first year's design conjectures, additional changes to the Summer 2 curriculum included students self-selecting the difficulty of their PixelBots challenges, instructors

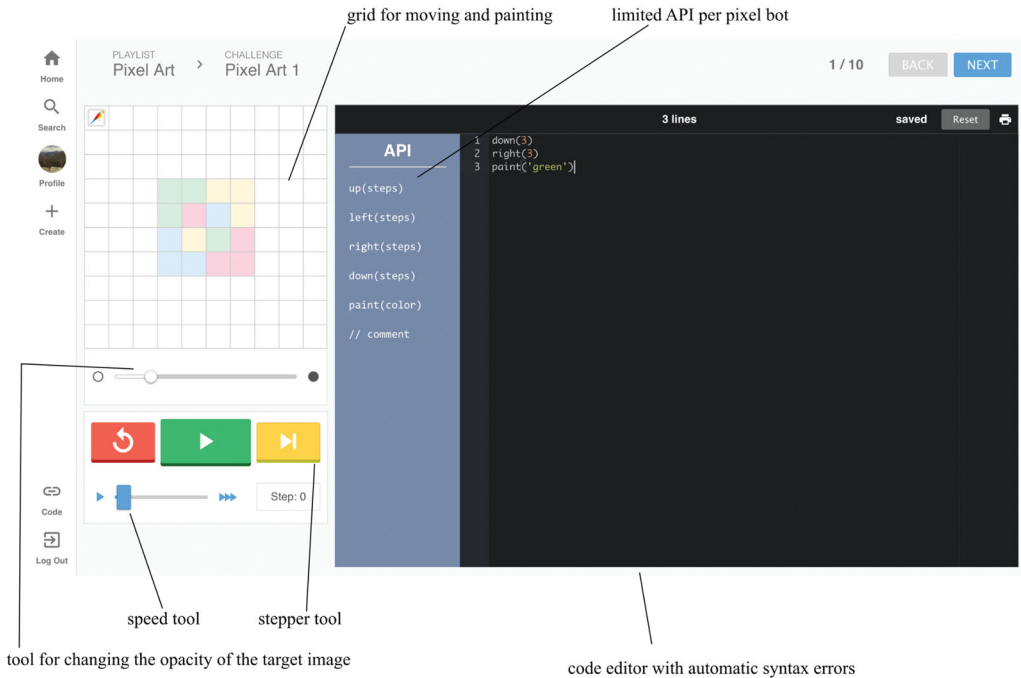


Figure 2. PixelBots programming interface with relevant debugging features highlighted.

emphasizing and promoting peer-to-peer teaching and learning, and instructors spending less time explaining concepts in whole-class settings and more time with small groups.

The arts-based component of both workshops differed from prior exemplars of arts-integrated computer science curriculum emphasizing tangible artifacts (e.g., Buechley et al., 2013) in that students in our workshops were not only using programming and art to create novel artifacts (Pepler & Wohlwend, 2018) but were also creating art as a medium to reflect on and express their experiences learning to program (see Dahn & DeLiema, 2020; Dahn et al., 2020). Across both Summers, students worked in the art space for about an hour each day. In Summer 1, an author of this paper who had several years of experience teaching visual arts, designed and taught the curriculum. Summer 1 arts classes took place in an open, indoor space at the learning center in a building across the courtyard from where students coded. The art curriculum was designed to support students in telling stories about success and failure in the context of learning coding and debugging. Over the course of the workshop, students worked on different art projects in relation to the experience of learning to code. These projects included abstract watercolor pieces about emotional aspects of coding experience, comic-strip story panels about events that happened during coding (or that students imagined happening), poetry reflections on feelings related to particular lines of written code, graph-like depictions of emotional trajectories related to coding over time, and data visualizations of their debugging experiences. Throughout the workshop, the art teacher-researcher reinforced the idea that art making had flexible goals and that art was a way of reflecting on and communicating ideas and feelings about their experiences. Students also had the opportunity to share their work with peers and families at the workshop's conclusion as part of the final showcase in a gallery-like context. In Summer 2, art making was more fully integrated into students' coding experiences as they explored and successively revisited three projects—abstract emotion watercolor drawings, code poems, and data visualization of their personal experiences—within the coding classroom and immediately preceding or following coding activities. See Dahn and DeLiema (2020) and Dahn et al. (2020) for additional details on the arts-based component of the workshop.

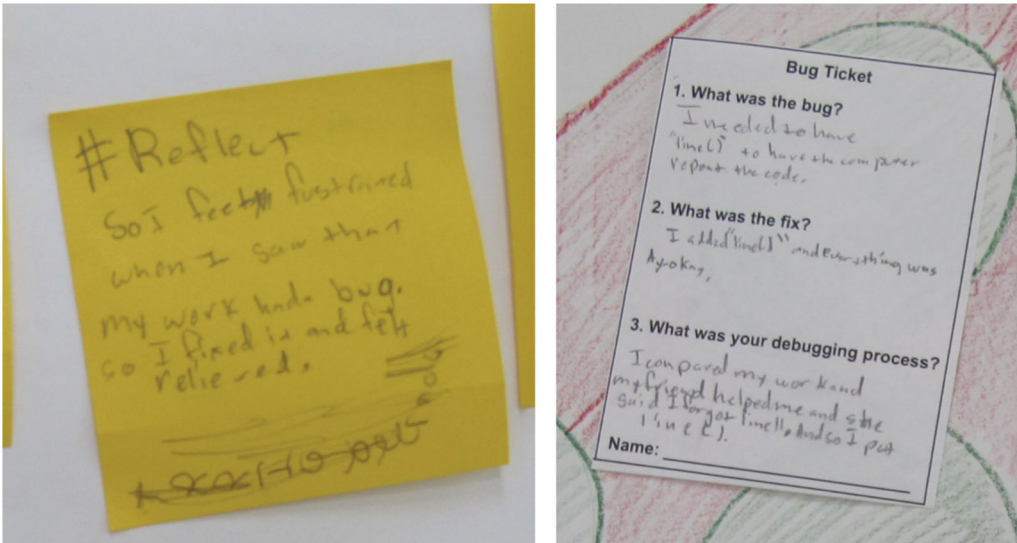


Figure 3. An example of one of Zoa's post-it notes (left) and bug tickets (right).

Debugging pedagogy designs

We implemented a number of designs to spark student reflection and communication about students' debugging processes (DeLiema et al., 2020). In lined-paper journals, students set goals at the beginning of class about how they wanted to push their debugging practice forward that day, and students reflected on whether their past debugging strategies had been working. Students engaged with similar reflective prompts at the end of class. These reflections also often took place on post-it notes (see Figure 3) that students then stuck to a specific category on a poster board; the board offered five categories that aligned with the five steps of debugging that instructors set out to implement (see details in the Professional Development section below). Students were invited to use hashtags on their post-it notes and journals as a way to incorporate emotion into the reflections. In addition, students were invited to fill out bug tickets at the end of class (see Figure 3), small slips of paper that had the following questions: "1. What was the bug? 2. What was the fix? 3. What was your debugging process?" Finally, instructors occasionally told stories about how experts in contexts outside of programming approached failure in their practice (Lin-Siegler et al., 2016), such as how a rock climber persisted throughout a difficult section of a climb, with the goal of validating strategic thinking during failure.

Professional development

During two weeks of professional development leading up to Summer 1, a researcher and all eight instructors collaboratively developed conjecture maps (Sandoval, 2014) describing three of the workshop's debugging-focused designs. Conjecture maps are detailed hypotheses about expected connections between learning theories, how these theories are implemented in classroom instruction, short-term outcomes of these designs evidenced by students' observable actions, and long-term goals of the learning space. They are a critical tool for design-based researchers (e.g., Bakker, 2018). We invited instructors into the conjecture mapping process, both its design and evaluation, to foreground their perspective and help democratize our teaching and research process (Ryan et al., 2019). These conjectures served as guideposts for our data analysis and deeply informed revisions to our pedagogical approaches in Summer 2 (described earlier). These conjecture maps focused on three designs: (1) student goal setting about debugging and reflection on

debugging in journals; (2) instructor modeling, prompting, and reflecting on debugging strategies (designed with the reciprocal teaching framework from Palinscar & Brown, 1984 in mind); and (3) students using five debugging steps, including debugging goal setting, describing what went wrong, proposing causal explanations, testing fixes, and reflecting on the debugging process (see DeLiema et al., 2020 for more detail). As part of professional development, instructors practiced their debugging instruction in mock classrooms with other instructors who pretended to be students, and instructors spent time watching video of prior classroom debugging instruction focused on *noticing* (Sherin & van Es, 2005) key moments of the debugging process.

Data sources

Our data sources span participation in coding class, the artifacts produced along the way, and reflection on interviews. To capture participation in the coding classroom, we used a combination of overview cameras, participant (table) cameras, screen and audio recordings, and views of student gestures toward their computer screens. To get these angles, we used two GoPro cameras at each corner of the tables where students coded and one overview GoPro camera. Screen recordings, in addition to audio from the laptops, were captured using Quicktime software on Apple laptops and Screencastify software on Chromebooks. End-of-workshop interviews focused on debugging and asked students to talk about how they define debugging, what their debugging process is, how they tried to improve with debugging, how they felt when they encountered bugs, whether the art influenced their coding, what caused bugs in their code, how they experienced the post-its and journals, what it was like working with their instructors, and where they have room for improvement with debugging. A separate interview in Summer 1, conducted by the art instructor/researcher, more specifically probed students' art making experiences. In the art-focused interview, students selected two of their works of art and described what they were trying to show and if/how they thought art making might have shaped their coding practice during the course of the workshop. For Summer 2, we also created a Likert-scale survey with five questions (corresponding with our framework), iteratively piloted the survey with students in Winter 2018 by having students answer the questions and voice their understanding of the questions and their answers, and then asked students and instructors to take the survey following PixelBots class twice a week for the three weeks of camp (see [Appendix](#) for survey questions). The students filled out the survey describing their own debugging experiences, and the instructors shared the task of filling out one survey for each student in their class with the goal of trying to describe that individual student's debugging experience on that day. Finally, we video recorded our Summer 1 professional development sessions focused on conjecture mapping and saved copies of the iteratively designed conjecture maps themselves.

Analysis plan

Analysis plan for research question 1

To gauge the extent to which instructors planned their debugging teaching taking into account the five processes we describe in our framework, we examined our three collaboratively constructed conjecture maps (see images of these maps in the Findings section). The analysis tied conjecture map statements and connections to one or more of the five processes in the framework. In particular, we attended to the short- and long-term outcomes that instructors predicted would result from proposed debugging designs. By iteratively working through the data, documenting in memos our rationale for mapping statements to categories, and attending to the facets of the constructs covered in the literature review and described in our definitions below—an enactment of the constant comparative method (Glaser, 1965)—we evaluated the extent to which instructors naturally expected connections between their planned instruction and the processes

described in our framework. Because some short- and long-term outcomes in the conjecture maps covered substantial ground, we were open to each addressing multiple processes in the framework.

Analysis plan for research question 2

To examine debugging during coding class, we conducted a case study from Summer 1 data. The boundary of the case study included one student over two weeks of PixelBots programming, her interactions with two instructors during whole-class, small-group, and one-on-one debugging, and her interactions with peers. Data sources included video-audio recordings of the student programming, two interviews, and multiple artifacts (e.g., journals, bug tickets) produced along the way. The focal student, Zoa, was a sixth grader who had coded at this organization's fall robotics workshop before attending the Summer session. We focused on this student for three reasons: (1) We hoped to focus on a student with some experience in coding who could move past basic syntax errors and into more complex logic bugs. We wanted to avoid focusing on a student in the classroom with the highest content knowledge/most programming experience because we wanted to leave open the possibility that the focal student would learn from their peers. (2) We wanted to focus on a student who had committed to participating in the debugging designs at the workshop, including writing post-it notes, bug tickets, journaling, and art making. Almost all of the students fit this criterion. (3) Lastly, we wanted a student for whom we had good data, including good audio and video of coding sessions, good photographs of journals/artwork, and good audio during interviews. We then randomly picked Zoa out of the remaining pool (five students) who fit all of the above criteria.

Our analysis approach drew on principles of the constant comparative method (Glaser, 1965) and multimodal interaction analysis (Goodwin, 2018; Jordan & Henderson, 1995). The analysis was driven top-down by the five processes in the framework. In line with recent multimodal, interactional debugging and programming research (e.g., Heikkilä & Mannila, 2018; Murphy et al., 2010; Sengupta et al., 2021), after forming a synchronized composite of video angles and screen recordings in Adobe Premiere for each day of Zoa coding, we marked each moment of discourse related to debugging, and created multimodal transcripts (representations of language, code, gesture, and body movement) for moments we judged central to our analysis, following conversation analysis conventions (Jefferson, 2004) interleaved with multimodal representations in black italics between double parentheses. All other data sources were transcribed or photographed. Through successive examinations of data, written memos documenting our reasoning, and data sessions with researchers outside of our team, we iteratively combed through each data source, exploring possible mappings between aspects of the data (e.g., a sentence in an artist statement, a strip of dialogue) and the five framework processes. In considering data from practice, artifacts, and reflection (Sandoval, 2012), we did not aim in this case study to comprehensively describe Zoa's growth or overall debugging trajectory, nor did we consider whether the data triangulated to amount to a common/consistent account of Zoa's debugging practice (Tracy, 2010). Instead, we approached the analysis looking for whether there were clear instances in which participants foregrounded each of the five processes in the framework. Though we do not report on this analysis in the present paper, we also started to investigate possible connections between the framework characteristics by examining a stretch of debugging for Zoa that spanned two workshop days.

Analysis plan for research question 3

To more specifically document patterns/regularities around each of the five processes in our framework in one student's coding practice over a sustained period, we conducted a case study using Summer 2 data. We selected Maribel, the focus student, according to the same criteria we

noted for research question 2. Maribel was a rising sixth grader who was learning to code for the first time. This case study focused entirely on the video and audio recordings of Maribel's coding process over three weeks of PixelBots class. Two coauthors of this study worked together in Adobe Premiere to mark every bug the student encountered and categorized each according to the definitions provided in the core constructs section (below). Two coauthors developed and refined these definitions and the accompanying coding scheme by coding the first week of Maribel's data, meeting to discuss discrepancies, refining the coding scheme, and re-coding the data (see coding scheme below). The open-ended approach to refining the coding scheme continued once each coder then separately coded the remainder of the data. The two coauthors in addition to a third coauthor met each week for a year to discuss any misalignment between the coding scheme and the video data, refined the categories, and re-coded the data. The process was iterative and collaborative to the final day of coding. All of the coded data was stored in a spreadsheet to make possible the construction of data visualizations showing the student's approach to bugs over time.

Approach to Video Coding for RQ 3. For our RQ3 analysis, we considered the start of the debugging session to be the moment the student noticed and started to work on fixing the bug, and we considered the resolution to be the moment the student moved on to a new goal following a period of debugging (however brief). In addition, we distinguished the observable failure from the cause of the failure. The observable failure is the functional part of the code, or the observable product of the code when run, which the student dispreferred and aimed to fix. The cause of the failure, typically described as a bug in the CS ed literature, is the flaw in the code that generated the functional problem. With these two terms in mind, we attended to debugging outcomes that either resolved or left unresolved both the observable failure and the underlying cause.

Our observations of debugging strategies focused on observable facets of students' problem-solving practices when faced with a bug (see full list in [Figure 9](#)): slowing down the code execution process with the PixelBots slider, stepping through code, looking up example syntax in the API, working with peers, working with instructors, simulating code in gesture or drawing, consulting help pages, creating new functions, reverting to a previous code version, and referencing past working code. We also attended to experimentation by considering the amount of code the student deleted (code within one or two lines versus anything involving three or more lines) and whether the proposed new code worked with the existing coding concept (e.g., trying to work with a loop that was broken) or used a new coding technique (e.g., reverting to a known technique, such as hard coding a sequence of commands, after a newer technique, such as the loop, did not work). Finally, we attended to both actual statements of self-efficacy (e.g., "I got this," "I doubted that the code would work") and proxy markers of efficacy during and following debugging, including markers of low efficacy (e.g., in the context of code not working, publicly cursing, slumping shoulders, groaning, grimacing, jamming the play button) and markers of high efficacy (e.g., announcing epiphany moments, expressing self-praise, showing excitement/enthusiasm about a debugging strategy). Lastly, we categorized bug types tailored to Maribel's coding practice by attending to all of her bugs and noticing commonalities both in their functional features and their underlying causes. Our discussion section covers some of the discoveries, uncertainties, and tensions that emerged around these constructs that we think would help researchers working with these constructs moving forward.

Analysis plan for research question 4

To gauge how instructors and students reflected post-hoc on these five processes, we analyzed our survey data from Summer 2. We analyzed the survey data by implementing two categorical analysis techniques: the Chi-Square Goodness of Fit Test (GOF test) and the Cochran-Mantel-Haenszel test (CMH test) to understand the rate of student and teacher responses as well as to

measure the agreement between students and teachers on our five proposed facets of debugging. More specifically, the GOF test is a hypothesis test used to determine how well a sample data distribution fits a specified distribution of data. The null hypothesis is that the sample data distribution matches the specified distribution, whereas the alternative hypothesis is that the sample data distribution does not match the specified distribution. The specified distribution in this paper is that the levels of the responses are all equally likely (e.g., students are equally likely to report rarely, sometimes, and always fixing bugs). The CMH test, however, is used to determine if there is agreement between two teachers and students regarding a categorical binary response over a discrete number of time points. The null hypothesis is that there is no agreement between teacher and student, whereas the alternative hypothesis is that there is agreement between teacher and student. Each test was implemented on each construct for students and teachers.

Results

Study 1: conjectures linking instructional designs with student learning processes

Our collaboratively constructed conjecture maps from Summer 1's professional development session (see Figure 4) document that instructors positioned each of the five framework processes as possible and valued outcomes of the workshop's debugging designs (see Table 1). The instructors formulated and refined these short- and long-term student learning processes in their own words, viewed them as key processes to consider around failure, and used them to anchor the team's design conversations. Altogether, instructors highlighted students' debugging processes (e.g., accurate hypotheses, effective explanations of the cause of bugs, likelihood of discovering bug); planning around specific bugs (e.g., specific gaps, specific questions); general debugging strategies (e.g., increased use of debugging resources; learning new debugging strategies); more personalization of debugging strategies and more student authority (e.g., full capacity to navigate debugging, student voice); and high self-efficacy (e.g., "ready to fix that bug" mindset).

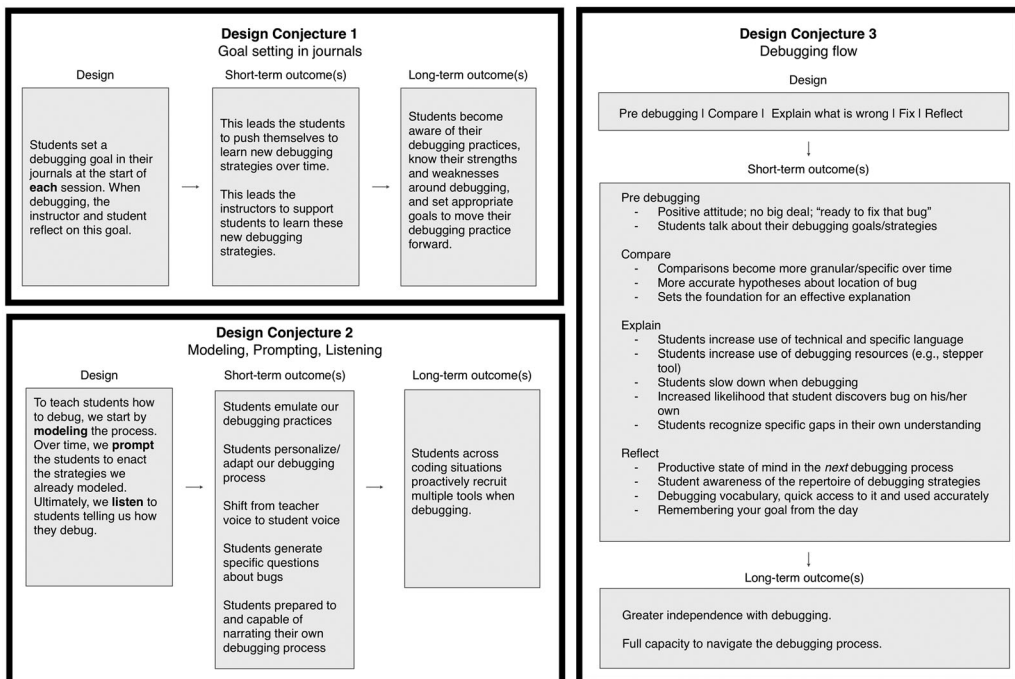


Figure 4. Collaboratively designed conjecture maps from Summer 1's PD workshop.

Table 1. Summary of connections between valued debugging outcomes and our conjecture maps.

Five debugging processes	Outcomes embedded in three conjecture maps from Summer 1
Fixing bugs	More accurate hypotheses about location of bug (conjecture 3) Sets the foundation for an effective explanation (conjecture 3) Increased likelihood that student discovers bug on his/her own (conjecture 3)
Avoiding recurring bugs	Students recognize specific gaps in their own understanding (conjecture 3) Students generate specific questions about bugs (conjecture 2)
Growing the toolkit of generalized debugging strategies	Students push themselves to learn new debugging strategies over time (conjecture 1) Instructors support students to learn these new debugging strategies (conjecture 1) Students set appropriate goals to move their debugging practice forward (conjecture 1) Students prepared to and capable of narrating their own debugging process (conjecture 2) Students across coding situations proactively recruit multiple tools when debugging (conjecture 2) Comparisons become more granular/specific over time (conjecture 3) Students increase use of debugging resources (e.g., stepper tool) (conjecture 3) Full capacity to navigate the debugging process (conjecture 3) Student awareness of the repertoire of debugging strategies (conjecture 3) Debugging vocabulary , quick access to it and used accurately (conjecture 3)
Debugging authority	Students personalize/ adapt our debugging process (conjecture 2) Shift from teacher voice to student voice (conjecture 2) Students prepared to and capable of narrating their own debugging process (conjecture 2) Increased likelihood that student discovers bug on his/her own (conjecture 3) Full capacity to navigate the debugging process (conjecture 3) Remembering your goal from the day (conjecture 3) Greater independence with debugging (conjecture 3)
Self-efficacy with debugging	Positive attitude; no big deal; 'ready to fix that bug' (conjecture 3) Productive state of mind in the next debugging process (conjecture 3) Students know their strengths and weaknesses around debugging (conjecture 1)

The bolded words in the outcomes column most represent the framework process described in the first column.

This analysis provides evidence that instructors keyed into all five framework processes when considering the purpose of their debugging-related pedagogical practices, and described them in enough detail and with enough precision for our research team to map them onto the multi-dimensional framework. This signals that instructors in this setting gauged students' progress around failure across a heterogeneous set of considerations. Four of the facets of the framework that instructors noted were spread across more than one conjecture map; instructors thus predicted that different debugging pedagogies would contribute to the same valued facet of responding to failure. In contrast, bug fixing was judged to be relevant only to conjecture 3's focus on the steps of debugging, a design that focused on finding solutions to bugs. Another emergent property of this analysis is that we can recognize that instructors emphasized general-purpose debugging and student authority over and above the other three processes, mentioning them more often and expecting to see them materialize across designs. Lastly, as we noted in our literature review, each of these processes is not uniform in its own right. For instance, authority captures working independently ("on his/her own"), personalizing the debugging process ("adapt our debugging process"), and commanding the debugging process ("full capacity to navigate").

Study 2: case study of enacted debugging

Beyond instructors' planning efforts, our first case study documents the extent to which participants foregrounded these five processes surrounding debugging in their teaching and learning within one classroom. Across three types of data—participation during coding class, artifacts produced along the way, and reflection through interviews—we found moments in which each facet of responding to failure was foregrounded by Zoa, her classmates, and her instructors. Below, we

provide at least one transcript of coding activity, one artifact, and one statement made in post-hoc reflection for each framework process to document how Zoa, her classmates, and her instructors foregrounded each during coding and reflection.

Fixing bugs

Over the course of the two weeks, instructors and peers routinely lent a hand when Zoa was debugging, signaling at least implicitly that bug fixes were important enough to warrant extra social resources. In a complementary sense, the software consistently signaled this value: PixelBots challenges from Day 1 contained automated methods for checking code accuracy, and PixelBots playlists contained numbered challenges that invited students to compare progress on how many problems they had accurately solved. Together, these software features emphasized the value of students precipitously arriving at correct code. This value was described on the final day of coding before an event showcasing student work: The instructors asked Zoa and her classmates to be especially conservative with their code that day to avoid introducing bugs that might disrupt their projects.

There were numerous examples from Zoa's coding practice of her pursuing bug fixes. We focus on one transcript from the first day of coding, in which Zoa is working with her neighbor, Kay:

Transcript 1 (Day 01)

```
01 Zoa: ((Zoa types turnleft into code editor))
02     Okay turn le:ft (1.2) (one) (.)
03 Flo: [Wait (0.4) you gotta (0.6) () capitalize the e1
04     [((points with pencil eraser at code editor and then retracts
05         pencil))]
06     [Like it like it yeah]
07     [((points with left hand at code editor and then retracts hand))]
08     [((points with pencil eraser at code editor and then retracts
09         pencil))]
10 Zoa: [((deletes lowercase l and types capital L))]
11 Flo: Ya:↑:::w
12     (0.5)
13 Zoa: [>Yaw<]
14     [((hands and head enact an abbreviated "dab" dance move))]
15     [((smiles))]
16 Flo: [Ya:↑:w]
```

In Transcript 1, Zoa and Flo celebrated the moment when code flipped from a broken state to a fixed state (lines 11–16). Flo isolated the bug in talk by stating “the e1” (Transcript 1, line 3, abbreviated subsequently as T1.L03) and gesture (pointing three separate times at the code; T1.L04, T1.L07, T1.L08), and described an exact fix to the bug (“capitalize”; T1.L03). Both students then laminated vocal cheers (T1.L11, T1.L13) onto the moment of fixing the error, and Zoa added a celebratory smile and a “dab” dance move (where the head drops quickly into the crook of the arm as both arms angle upward; T1.L14). In this way, Zoa and Flo established a precedent on the first day of coding that fixes to bugs were deserving of celebration.

In Zoa's artwork, she alluded to this value by connecting a smiley emoji to bugs that were “fixed” and an angry/frustrated emoji to the experience of being “confused” (see Figure 5). Attention to this value is evident in Zoa's writing about debugging: “Then I #fixed it” (Journal Day 2), “I added ‘line()’ and everything was Ay-okay” (BugTicket Day 3), and “#Fix #Got a bug #Finished the bug” (Post-it Day 6). In her end-of-camp interview, Zoa similarly emphasized bug

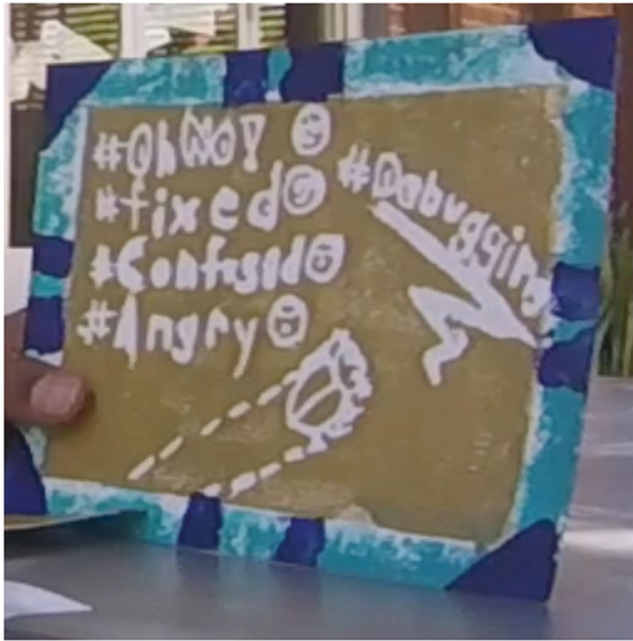


Figure 5. Zoa's artwork with a smiley emoji around "fixed" and angry emoji around "confused."

fixes as an overall goal. When things were fixed, they were "done with." More directly, Zoa explained: "There's a bug here I need to try to fix it so I can continue... so we can move our bug away and continue our work and enjoy our coding." In this last strip of talk, Zoa drew a sharp distinction between broken code and correct code; it was correct code that led to continuation and joy. In summary, in these coding practices, artifacts, and reflections, Zoa, her peers, and instructors surfaced the process and value of fixing bugs.

Avoiding recurring bugs

Beyond getting bugs fixed, the coding workshop provided occasions for Zoa to focus her attention on avoiding recurring bugs. Materially speaking, this value was embodied in the bug ticket, which on Day 3 an instructor explained in a way that highlighted its purpose as a resource for eclipsing common bugs: "If you guys have a bug that you think you really need to remember for next time, and you think it'll be super helpful for someone else if they run into that specific kind of bug, we have these bug tickets. You can take one, fill one out." This articulation of the purpose of the bug ticket brought to the surface its value as a tool for "remembering" a "specific kind of bug" for "next time."

The value of avoiding recurring bugs also arose in peer-to-peer interactions during debugging. Toward the end of the first day of the workshop, Zoa learned to write example syntax in her journal after she watched Flo do the same:

Transcript 2 (Day 01)

01 Zoa: **((leans to the side to look at Flo's laptop screen))**
 02 Do you have the code?
 03 (1.6)
 04 Flo: [Eh::: I (kind of)
 05 [**((picks up code journal lying open next to laptop))**
 06 I [have it=

07 [**((index finger and thumb touch journal page forming a bracket**
 08 **over written text))**
 09 Zoa: =>Oh yeah you:: um< (0.4) wrote it down (0.8)
 10 [I'm gonna write down my (
 11 **((picks up journal))**
 12 **((moves journal to other side of laptop, glances at laptop**
 13 **screen, and starts writing in journal))**

In Transcript 2, following Zoa's request for example code to help her debug a broken program (T2.L02), we see Flo highlight in gesture and talk where she had written down working code (T2.L04-08), and we see Zoa starting to engage with the same practice on her own (T2.L09-13). Whether intentionally or not, this practice provided Zoa with an effective way to curb recurring syntax bugs. For example, from this point on, Zoa consistently wrote working syntax in her journal. Transcript 3 provides one example from five days later, in which Zoa both spontaneously kept the practice alive and framed it around recurring bugs:

Transcript 3 (Day 05)

01 Zoa: **((corrects a mistake in two-parameter function))**
 02 I need to take a note on this cuz then I'll forget
 03 Sam: Multiple parameters maybe?

In the brief exchange above, Zoa wrote working code in her journal to make sure that she did not “forget” a specific bug fix, which could be understood as either a proactive way to prevent problems down the road should Zoa need to use the same syntax, or help her debug related errors should they recur. The student sitting beside her, Sam, participated by offering a title (“multiple parameters”) for her journal entry. By the end of the workshop, Zoa's documentation of working code amounted to an array of syntactic guidelines spread throughout her journal and other artifacts:

- I forgot line() (Bug Ticket Day 3)
- Next time ... make sure I have line on my code (Post-it Day 3)
- Tips: circle (500,500,100,100) (Journal)
- Every function Begins with keyword Function (Journal)
- Always starts with word if (Journal)

These syntactic guidelines were narrow; they prepared Zoa to correct bugs relevant to specific syntax, which she signaled with phrases like, “next time,” “every,” and “always.” Finally, during her end-of-workshop interview, Zoa spelled out how her debugging process involved looking for her common syntax mistakes, such as, “checking if I'm missing a curly brace.” In each of these examples from Zoa's coding, journaling, and interviewing, we see Zoa, her peers, and instructors foregrounding the value of preventing specific, recurring bugs in code.

Preparing for novel bugs

Beyond getting bugs fixed and avoiding specific bugs, the instructors and Zoa at this coding workshop invested time in teaching, developing, and reflecting on general-purpose debugging strategies. This investment was geared toward students expanding their set of techniques for responding to bugs, especially novel bugs for which students did not yet have ready-at-hand fixes. Zoa's instructors emphasized this value on Day 2 when Jad presented a slide with a long list of debugging strategies and mindsets, and then invited his students to set a goal: “Before we start coding, let's do a debugging goal real quick... let's pick one strategy we want to work on today.” Noticing that students had been gravitating toward mindset goals, a researcher-teacher

don't find the bug, I comment it out... I ask myself what did I do wrong, how was it before, and how can I change it?" In each example drawn from Zoa's moment-to-moment coding experience, journal reflections, and interviews, we saw Zoa growing and reflecting on her tool-kit of debugging strategies.

Engaging with authority in the debugging process

Beyond the processes surrounding failure that we discussed above, the coding workshop included features that nudged students to engage with authority in the debugging process. While authority could refer to a number of dimensions of learning environments (see our literature review), we followed Zoa's lead of foregrounding independence. For example, in her end-of-workshop interview, Zoa stated: "[The instructors] never really do the work for me. They give me hints... I would figure out the problem." Looking granularly at the division of labor when Zoa debugged, we found a complex picture. Zoa learned new debugging strategies in the context of debugging her code with a peer or with an instructor. For many of the strategies she learned (e.g., check API, write down correct syntax, read error messages, step through code, experiment with code), there were then numerous instances of Zoa applying that strategy on her own. In her coding journal, Zoa captured some of this complexity with respect to the division of labor during her debugging sessions. Zoa referred to fixes coming from herself (statements with "I") and from other people in the classroom (a "friend" and "awesome helpers"): "Then I #fixed it" (Journal Day 2), "I compared my work and my friend helped me" (BugTicket Day 3), "I added 'line()' and everything was okay" (BugTicket Day 3), "I encountered the problem by asking for assistance from awesome helpers" (Journal). To provide one example from her coding activities on day 7, we saw an instructor deliberately giving Zoa space to discover a fix:

Transcript 6 (Day 07)

```

01      ((Mia walks over to Zoa and leans down next to her))
02  Mia:  Are you havin:: uh some iss::ues
03      (1.0)
04  Zoa:  ↑I don't know:: [who-uh::: (.) why it has the exes:::
05          [ ((drags cursor back and forth over red xs in
06          editor))
07      I'm looking::
08      (0.4)
09  Mia:  So:: (1.0) right now you just ha:v::e (6.5) um (.) look at
10      the::: (0.4)
11      [compare your repeat syntax with (.) the API's repeat syntax
12  Zoa:  [ ((drags cursor around the repeat loop once, and then places the
13      cursor near the bottom of the loop))
14      [(28.0)
15      [ ((moves cursor to API repeat syntax and then back to the repeat
16      loop in her own code. Toward the end of this period, deletes
17      color and comma following color, and adds comma after num))
18      (1.5)
19  Mia:  [Yup      ] (1.5) >There ya go<
20      [ ((nodding))
21      (0.4)
22  Zoa:  Okay.
```

Looking at the details of this transcript, a few noteworthy moments with respect to authority stood out. The student narrated active investment in finding the bug

(I'm looking::), the instructor appeared to be on the cusp of presenting a solution ("right now you just ha:v::e), and then the instructor paused for 6.5 seconds before offering a debugging strategy (T6.L10-11), the same strategy we highlighted in the section above. For 28 seconds of silence (T6.L14), the student actively compared the syntax in the API with her own syntax and enacted a fix to her code (T6.L15-17), before the instructor affirmed the correction (T6.L19). Authority here manifested in a number of ways. The student conveyed her active strategy at the outset, the instructor held off from providing a solution and instead offered a tool, and both participants then allowed enough space for the student to uncover both the bug and its resolution.

Outside of coding time, in response to an interview question about what it was like to work with her instructors during debugging, Zoa described her own motivation for moving away from instructor support toward the end of the workshop: "Since I've been doing so many codes, I decided to like, instead of asking, I tried to compare with my partner ... because if you keep on asking for help, you're never really going to improve, you're just kind of getting the work done by someone." This suggests that Zoa wanted to practice driving the debugging process without instructor support, and that Zoa indeed pursued that independence. Zoa's interest in driving her own debugging process, and the instructors' efforts to signal that value, resulted in a number of instances in which Zoa independently attempted to debug problems she had not yet encountered in her code.

Self-efficacy during debugging

The last characteristic that we considered in this case study is self-efficacy. On Days 3 and 6, Jad told his students a few stories at the start of class—one about the author J. K. Rowling and one about the basketball player Isaiah Thomas—that focused on believing in oneself even when success did not arrive immediately. Jad explained the moral of these stories as trying to "have no fear, have faith, experiment" even when we do not immediately reach our goals or when "we just have really bad days."

In practice, Zoa's stated expectations throughout the two-week workshop about whether her debugging approach would work arrived as a mixed bag. The following contradictory declaration Zoa made on Day 3 best documents this viewpoint: "I'm slowly dying. I got this." Simply put, Zoa wavered in her expectation that she could debug her code. In some cases, Zoa made visible an assessment that her approach to debugging may not work. For example, on Day 3, Zoa worked consistently over 10 min to find a syntax bug (an errant curly brace) in her code (with some support from Flo), but fell short of finding and fixing it. Throughout that period, Zoa made a series of statements suggesting that she would not find the bug:

Transcript 7 (Day 03)

01 Zoa: >I give up<

02 Zoa: Screw it (.) if you won't let me (.) if you say this is WRONG (.)

03 WHAT IS WRO::NG WITH THIS

04 Zoa: Why isn't it wor::king (1.4) I had a year of expe:rience

05 Zoa: Oh:: M::ia:: we're dy::ing

06 Flo: There's nothing wrong

07 (0.2)

08 Zoa: There's nothing wrong

09 Flo: NOTHING wrong
 10 Zoa: It is: (.) pure perfection.

 11 Zoa: So:: I was ty:ping and then like (.) for some reason it said there
 12 was an error? but then my spelling was perfect so it's just
 13 like the [computer that's having the bug
 14 Flo: [It was perfe::ction

The above excerpts show Zoa stating that she was giving up (T7.L01), questioning how she could have a year of coding experience and still not be able to find the bug (T7.L04), and ultimately blaming the computer, not her code (pure perfection), for producing the problem (T7.L06-10). In her abstract artwork, Zoa described her experience with these feelings of doubt: “The crumbled paper shows the amount of times I doubted myself that the code wasn’t gonna work.” And in her interview at the end of camp, Zoa described her emotional response in moments when she could not debug her code: “And then angry is that I wouldn’t be able to find it and I would get mad.” In fact, there were a few moments on Days 6 and 7 when Zoa outwardly expressed frustration (e.g., saying “... darn it!” and hanging her head down on the table before exclaiming: “my life!”) during debugging episodes that did not lead to immediate fixes. In one manifestation of this experience, Zoa, Flo, and Sam on Days 3 and 4 repeatedly sang a variety of failure-themed lyrics and melodies from popular songs: Coldplay’s lyric, “When you try your best but you don’t succeed,” A Great Big World’s lyric, “Say something I’m giving up on you,” and Gloria Gayner’s lyric, “I was petrified.”

In contrast, in a journal entry, Zoa described wanting to have the expectation that her bugs would be fixed: “#Determinationmode I want to believe.” Indeed, Zoa capably fixed numerous other bugs, both tackled together with other people and approached by herself. After one string of successful fixes on Day 3, Zoa looked up from her code and said, “This is actually fun. I plan on doing this at home,” a remark that suggested her confidence with debugging was high at that moment. In her end-of-workshop interview, Zoa also remarked, “Okay, so there’s a bug, no deal. I just gotta figure out how to fix it. So, almost positive at first.” On Day 8, an instructor who had not visited her classroom asked Zoa: “Do you have any bugs here? Or is it bug free?” Zoa replied: “Everything is bug free if you know what you’re doing. We’re all experienced now so not many bugs with PixelBots,” broadcasting confidence with her coding and debugging practice. Overall, these excerpts from Zoa’s talk about whether her approach to debugging would work document her fluctuations in confidence, ranging from wanting to call an end to her search for the bug to describing a sense of positivity at the start of a debugging session.

Brief Discussion of Heterogeneity in Zoa’s Case Study. We documented in this case study that Zoa, and the classmates and instructors with whom she worked, encountered failure along each of the multiple processes called out in our framework. Resonant with failure as mosaic framings (A. Anderson et al., 2019; Simpson et al., 2018), notions of computational utterances as heterogeneous (Sengupta et al., 2021), and learning as inherently heterogeneous (Rosebery et al., 2010), bugs in code served as points of departure for reflecting on self-efficacy, growing the toolkit of general debugging strategies, preparing to avoid/handle similar bugs, positioning Zoa with authority to handle the bug, and simply doing away with the problem. For example, on Day 1, a bug served as a point of departure for getting to a quick fix (Transcript 1), whereas on Day 7, a bug served as a point of departure for an instructor to both promote student authority and reinforce a general debugging strategy (Transcript 7). Moments of coding, artifacts produced along the way, and strips of reflection each served as activities through which Zoa and her collaborators could foreground these multiple facets of failure. The qualitative work in this case study shows the details through which these processes come to the foreground (e.g., how a bug ticket serves as a prompt to reify a general debugging strategy), while capturing some heterogeneity

within each (e.g., wavering self-efficacy; crediting debugging to oneself or others), but the case study falls short of documenting patterns within each process over time. Our next case study pursues this angle.

Study 3: a quantitative, longitudinal examination of the framework through a case study

Our next case study, focused on Maribel, documents the longitudinal trajectory of four framework facets over the course of three weeks of coding for one student.² The analysis of Maribel's coding in PixelBots focused on 181 distinct bugs that she encountered during the Summer 2 workshop. The data visualizations below document a range of features of the process through which Maribel responded to these bugs. The purpose of this longitudinal analysis was to observe trajectories of stability and change within the framework processes, and then raise questions about their interconnections.

The first visualization captures when Maribel encountered bugs and how long they persisted (Figure 6). The second shows whether the observable problem (OP) was resolved, whether the fix addressed the bug that caused the observable problem (BC), and whether Maribel tackled/attempted debugging the cause (Figure 7).

From these two visualizations, we see that Maribel encountered bugs throughout the three weeks of coding activities. Her debugging sessions lasted anywhere from a few seconds, to a few minutes, and even tens of minutes, and gaps between bugs had a range of shorter (a few seconds) and longer (15 min) durations. Days 8 and 9 in particular were filled with prolonged debugging sessions, while Day 4 had shorter but more frequent debugging sessions. We note these features to provide a baseline description of the prevalence of debugging for Maribel, and to signal that days differed with respect to the debugging load Maribel experienced. Attending to the first facet of our framework, we can see in Figure 7 that Maribel resolved the vast majority of her bugs (the first three rows); there were only ten instances of ceasing to pursue a bug fix.

However, in considering bug recurrence, the second feature of our framework, we can attend to Figure 7 again to see the extent to which Maribel addressed the cause of the observable problem, or whether she solved the problem in a way that eschewed the underlying cause. For an example of the latter, a student working on a function definition that includes a loop to paint a line might encounter an observable problem (e.g., the PixelBot paints too many squares) and return to an earlier developed skill to resolve it, such as “hard coding” the same line with individual commands instead of with a loop. This would avoid what was originally broken about the loop that caused the observable problem. For Maribel, we can see in the second and fourth rows that Maribel occasionally tackled a bug causing an observable problem, but did not succeed, and then either found a different way to solve the observable problem (unrelated to the bug) or abandoned the goal altogether. In rows three and five, we see examples of Maribel never attempting to address the bug that caused the observable problem, and instead resolving the observable problem a different way or abandoning the goal altogether. Because addressing the causes of bugs is critical to bug recurrence (Endres, 1975; Pea et al., 1987), these pathways through debugging that avoid the underlying cause *might* indicate areas of investment for Maribel's emerging coding practice. Even more directly, in Figure 8, we can consider the specific bugs that Maribel encountered and whether they recurred.

This data visualization more precisely documents whether the bug types that Maribel encountered, based on a categorization scheme we developed bottom up by attending to Maribel's

²This analysis does not consider authority; Maribel almost exclusively worked alone, and though independence is not synonymous with authority, with this particular approach of looking only at bugs that Maribel encountered, we found it difficult to pursue other facets of debugging authority.

LONGITUDINAL BUG FREQUENCY

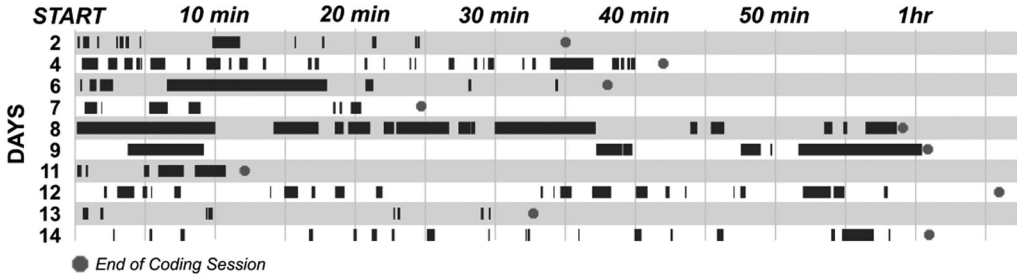


Figure 6. Longitudinal visualization of bugs encountered and bug duration.

LONGITUDINAL BUG FIX

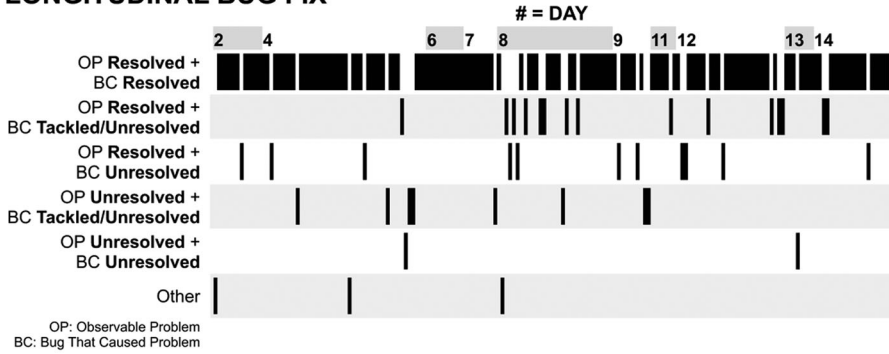


Figure 7. Visualization of resolutions with respect to bugs and bug causes.

coding activity, recurred throughout her practice (the individual black squares in each row) and how often they recurred (greener colors on the spectrum indicate delayed recurrence while redder colors indicate more immediate recurrence). To highlight a few details about this visualization, we can note that some bugs occurred frequently for a stretch of time, and then faded away, such as when Maribel used the wrong argument in loops (row 6) and function definitions (row 13), and came across computer literacy bugs (row 13). Other bugs showed the reverse pattern. For example, with syntactic spelling errors, Maribel encountered a few and then several all in succession toward the end of the workshop as she struggled to correctly use or spell the names of the various parameters she had used in her function definitions. In contrast, other bugs occurred only infrequently, whether over a short span of time or throughout the three-week workshop. Similar to Figure 7, Figure 8 shows where an instructor and/or Maribel might adjust their approach to failure provided that a problem is persistently recurring or arising only occasionally.

Considering the third facet of the framework, we created a data visualization documenting Maribel’s growth in debugging strategies (Figure 9) and one visualization documenting Maribel’s use of debugging strategies related to the length of the bug (Figure 10).

In Figure 9, we see that one of Maribel’s debugging strategies, which involved specifically replacing one or two features of code within the context of her existing strategy, persisted throughout the workshop and occurred frequently. Others persisted throughout the workshop but with less frequency. A range of new debugging strategies entered the picture roughly between bug 60 and bug 155 (corresponding mostly with the workshop’s second week). With the stepper tool, first used on bug 78, Maribel used it repeatedly for a short duration of bugs, and then continued to use it sporadically, while other strategies, such as looking up reference code and consulting an

LONGITUDINAL BUG CATEGORY

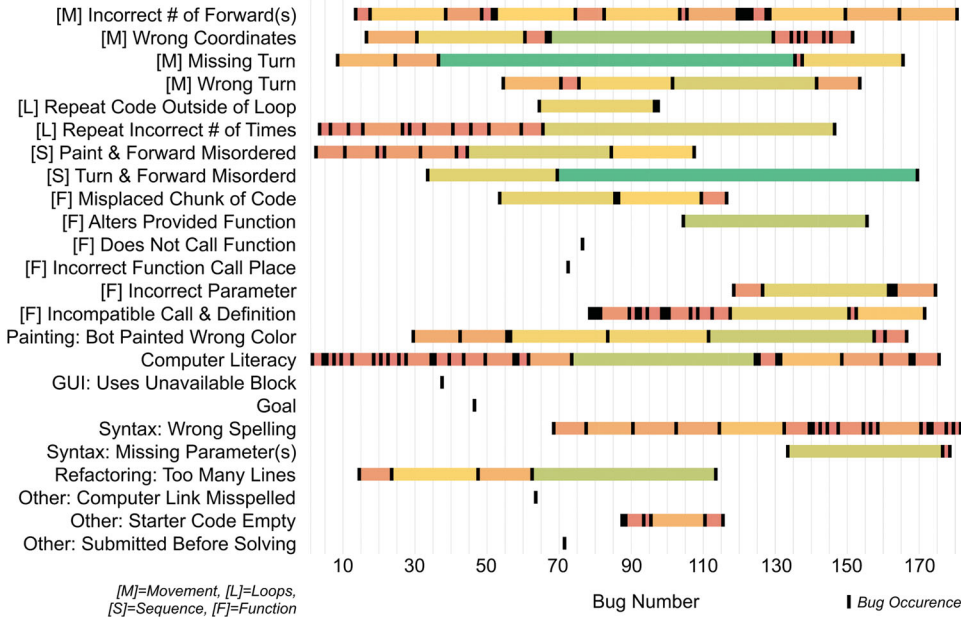


Figure 8. Recurring bugs over time (in black) and their frequency (in color), with redder sections documenting quick recurrence and greener sections documenting delayed recurrence.

LONGITUDINAL BUG STRATEGIES

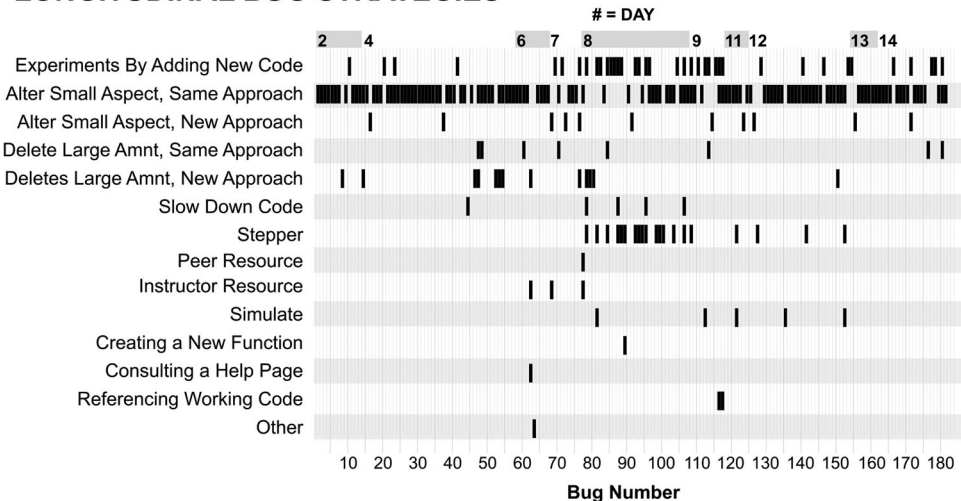


Figure 9. Use of debugging strategies over time.

online help page, were used once or twice before never re-appearing in her practice. In comparing Maribel’s use of strategies with the duration of her debugging sessions (Figure 10), it is evident that Maribel’s use of two, three, and four strategies almost always occurred during her moderately long and very long debugging sessions. This suggests that the long debugging sessions that occurred during week 2 of the coding workshop may have driven Maribel’s exploration of new debugging strategies and efforts to deploy multiple debugging strategies at once.

NUM STRATS VS. BUG DURATION

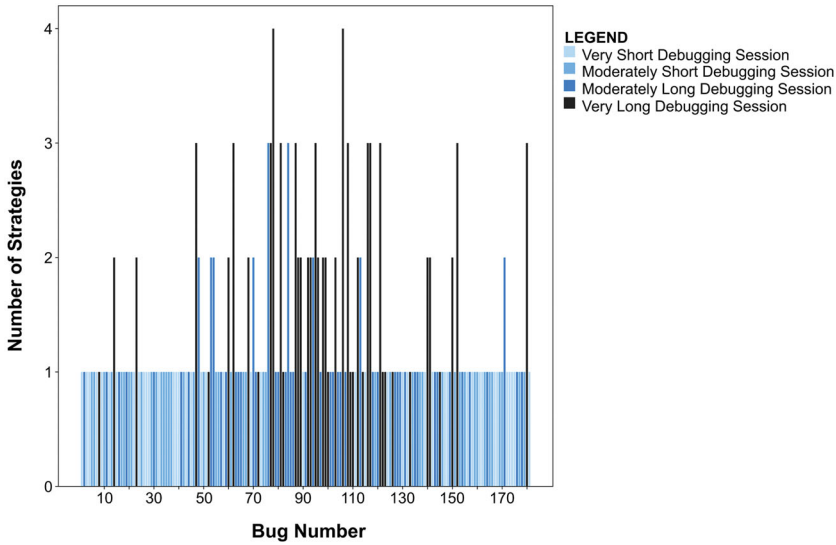


Figure 10. Number of debugging strategies used relative to bug duration.

LONGITUDINAL SELF EFFICACY

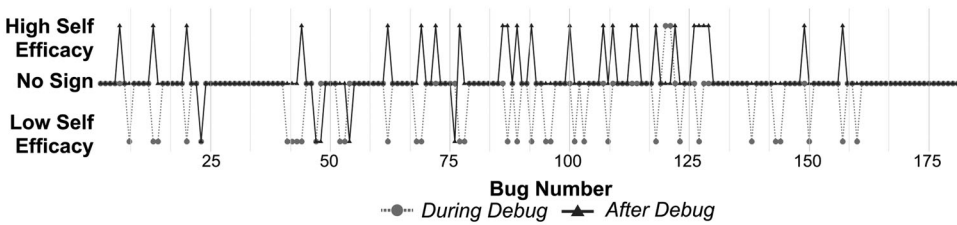


Figure 11. Outward proxy indicators of high and low efficacy before and after debugging.

Finally, we tracked when Maribel expressed outward signs of confidence/pride versus signs of frustration and anger as proxy indicators of her self-efficacy both during and after debugging (Figure 11). This visualization shows a number of noteworthy patterns. First, proxy indicators of high efficacy and low efficacy tended to occur together: Maribel tended to pair her outward signals of frustration and low confidence with signals of confidence and pride. Moreover, high efficacy signals during coding tended to occur following debugging and low efficacy signals tended to occur during debugging. Lastly, Maribel’s high and low efficacy signals appeared all throughout the three weeks of the coding workshop.

Brief discussion of Maribel’s case study

Taking into account all six data visualizations, they present a striking depiction of the range of debugging experiences that Maribel encountered over just three weeks. We see that debugging sessions were short or long; resolved or unresolved; used to pursue or avoid the cause of the problem; part of a long or infrequent stretch of bugs; related to previous bugs or surfacing novel bugs; tackled with specific debugging strategies and specific numbers of strategies; and accompanied with high, low, and no signs of outward self-efficacy. By capturing the recent and distant past of each of Maribel’s bugs, including how she experienced each of four framework processes along the way, these visualizations show the heterogeneity of debugging experiences for Maribel.

More specifically, the combination of visualizations show that each bug is part of a longer and multi-dimensional trajectory. A specific bug might be recurring, tackled with often-used debugging strategies, accompanied with outward indicators of low self-efficacy, and lasting a long time, in contrast to a bug that has appeared for the first time, is tackled with a new debugging strategy, accompanied with no signals of efficacy, and lasting only a brief time. We see stretches of growth in debugging strategies (before many of these strategies fall away), stretches of rapidly recurring bugs, spikes and dips in proxy markers of efficacy, and an overwhelming trend toward Maribel fixing her bugs. One of our intentions in documenting these patterns, the diversity within each failure process, and the heterogeneity in how multiple failure processes line up around a particular bug is to raise questions about what kind of pedagogy would be most supportive for Maribel at specific points in her trajectory with failure (out of 181 total moments over three weeks). We explore this question in the discussion section.

Study 4: post-hoc reflection on facets of the framework

Lastly, we present data on instructors' and students' post-hoc reflections on each facet of failure to provide a window into their summary judgments of each and whether any substantive differences exist between their points of view (see Table 2 and Figure 12). This analysis points to whether the learning community tends to experience the full range of failure experiences or a partial range, including whether teachers and students align in their impressions of each failure process.

The GOF test was used for students' and teachers' responses with respect to each question to determine if their responses were equally likely (e.g., across the three weeks, students reported that bugs were equally likely to be fixed *rarely*, *occasionally*, or *always* on a given day). This gives us a rough sense of whether the learning community experienced the full spectrum of each failure process or just one end of the spectrum. The scale for each question was between 1 to 10 where 1 represented an end of the spectrum for the question (*never* fixing bugs; *lots of help* from instructors) and 10 represented the other end of the spectrum for a given question (e.g., *always* fixing bugs; *never* getting help). In order to implement the GOF test based on the amount of data that was collected, ranges were created and specified based on the scale. The responses 1–3 were labeled “Rare,” 4–6 were labeled “occasional,” and 7–10 were labeled as “frequent.” These levels provide meaning to what the students and teachers imply by their numerical selection.

Table 2 indicates that students' responses were only equally likely for the question about debugging strategies. This implies that students' responses to the other four questions indicate that either one or two level(s) were more likely to occur than the other level(s). More specifically, by examining Figure 12, we can see that students were more likely to report fixing bugs, working alone, encountering old bugs, and having higher confidence. If these are read as valid indicators of the failure culture in the classroom, they show that most students are experiencing a narrow

Table 2. Students' and instructors' post-hoc reflections on five facets of failure.

Questions	Chi-Square Goodness of Fit Test	Chi-Square Goodness of Fit Test	Cochran–Mantel–Haenszel Test
	Students	Teachers	
Bugs fixed today	86.6 ($p < .003$)	106.2 ($p < .003$)	5.199 ($p < .05$)
New bugs encountered	30.23 ($p < .003$)	3.17	17.570 ($p < .003$)
Debugging strategies used	1.15	18.93 ($p < .003$)	1.114
Worked alone	43.9 ($p < .003$)	1.94	14.067 ($p < .003$)
during debugging			
High confidence	86.25 ($p < .003$)	52.09 ($p < .003$)	6.491
during debugging			

Note: The sample size was 274 and the degrees of freedom was 2. Since there were 15 tests run on this data set, $\alpha = 0.05$ was divided by 15 to correct for inflation of Type I Error.

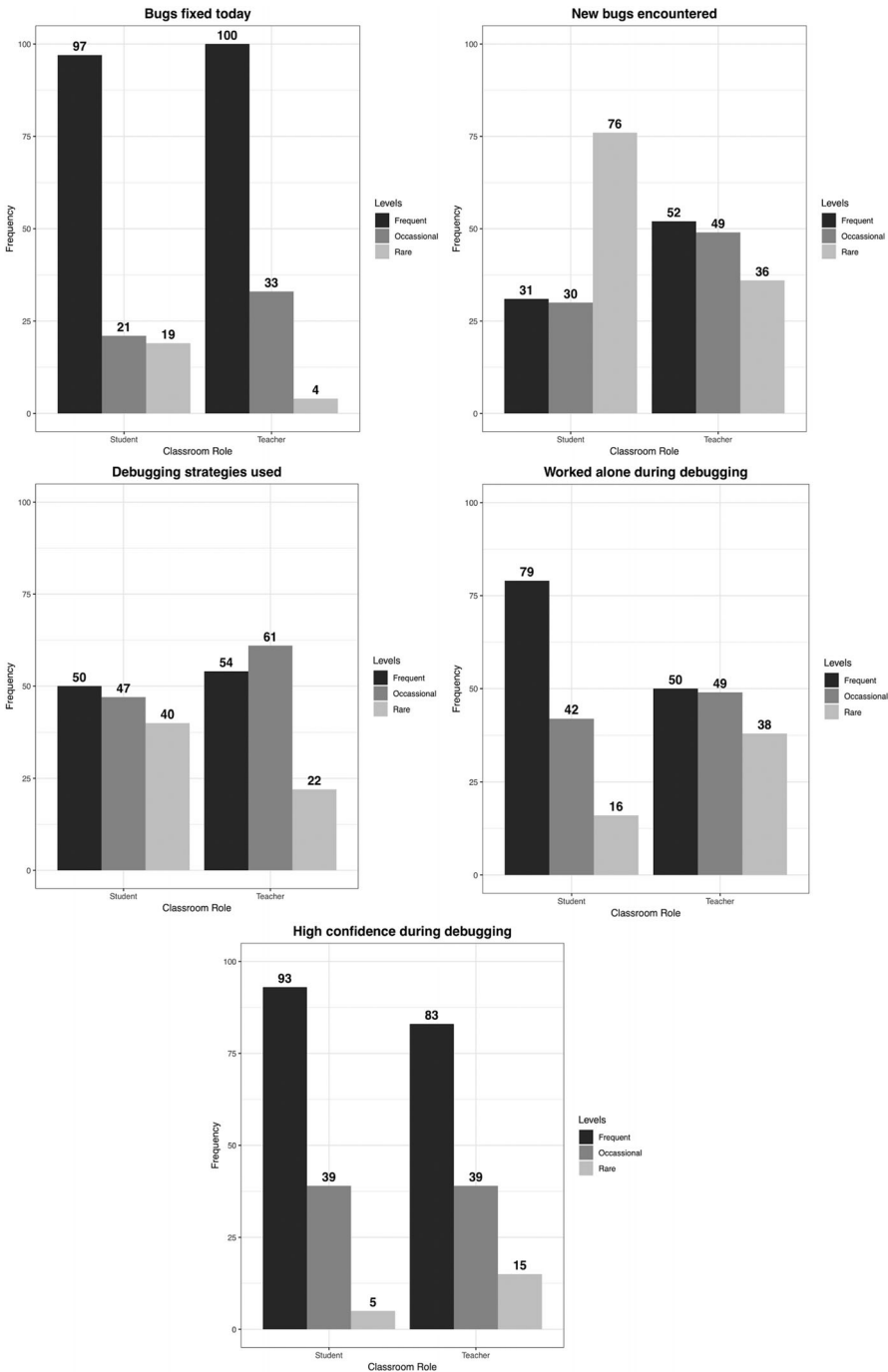


Figure 12. Survey results of students and instructors' views of five framework facets.

slice of these failure processes: overwhelmingly succeeding when faced with bugs, working independently, returning to bugs they've already encountered, and feeling confident along the way.

On the other hand, Table 2 indicates that teachers' responses were equally likely for questions about new bugs encountered and working alone during debugging. This implies that teachers' responses to the other three questions indicate that either one or two level(s) were more likely to

occur than the other level(s). [Figure 12](#) shows that for questions about bugs fixed and high confidence during debugging, the “frequent” option was most likely, and for questions about debugging strategies used, instructors more often reported that students were frequently and occasionally using debugging strategies. These data suggest that instructors and students viewed the debugging culture in their classrooms not as an even distribution of the processes described in our framework, but rather as tilting toward some processes happening more frequently, such as bug fixes and high confidence.

In addition, we compared instructors and students’ views with one another to determine if they perceived these processes surrounding failure in different ways. In order to implement the CMH test based on the amount of data that was collected, we needed to make the responses binary for each scale. The responses 1–4 were labeled “rare,” 5–10 were labeled “frequent.” The CMH works by analyzing the role (student vs teacher) and the constructs across multiple time points to determine whether there is an association between teacher and student responses. Based on the data there were six time points at which the students and teachers were measured. Instructors and students tended to align in their views along the dimensions of bug fixes, recurring bugs, and collaboration. In contrast, students and instructors tended not to view the use of debugging strategies and confidence in similar ways. Instructors were more likely to report that students were using more debugging strategies (across weeks) and experiencing lower confidence (in particular during week 1). This analysis raises questions about why instructors and students might diverge in their impressions of some processes, how divergent and convergent views will shape debugging teaching and learning, and how instructors and students can generate open communication channels and/or gather more precise data to more deeply understand students’ experiences with failure. In all, this variation across students and instructors, including the variation within students and within instructors, suggest that participants in the same space do not necessarily align in their views about the processes surrounding failure and do not equally experience all facets of failure.

Discussion

This study engaged with multiple processes surrounding failure as they unfolded in one learning community during our first year of data collection on a design-based research project. As we clustered this heterogeneity into five processes linked to how students respond to failure—fixing bugs, avoiding recurring bugs, growing the toolkit of debugging strategies, engaging with authority in the debugging process, and calibrating efficacy to debug—we found a rich literature examining the importance of each one of these processes (e.g., [Engle & Conant, 2002](#); [Kallia & Sentance, 2018](#); [Kapur, 2016](#); [Katz & Anderson, 1987](#); [Russ & Berland, 2019](#)). In the next stage of data collection and analysis (studies 1–4 reported above), we explicitly traced these five processes in multiple ways within this learning community, finding that they were deeply embedded in the teachers and students’ activity. Each of these five processes shaped how instructors planned in advance of CS workshops to support debugging. They were selectively foregrounded and/or highlighted in social interaction, the artifacts produced along the way, and reflective interviews with one student, her peers, and her instructors over two weeks. In addition, each process unfolded with facets of both stability and variation at different points for one student during the process of writing and debugging code over three weeks. Lastly, when explicitly reflecting back on how students experienced each failure process, instructors and students were not entirely in agreement, and both viewed the classroom’s failure experiences as tending toward an uneven expression of some processes (e.g., fixing bugs and having higher confidence more often than not).

These research findings demonstrate that students and instructors engaged with a heterogeneous set of values and processes following moments of failure. By extending the failure as mosaic point of view ([A. Anderson et al., 2019](#); [Simpson et al., 2018](#)) and notions of

computational utterances as inherently heterogeneous (Sengupta et al., 2021), we framed moments of failure as pluripotent. Teachers and students can foreground, background, and blend each of these five processes during classroom planning, actions taken to advance learning, and moments of reflection related to impasses. We can consider heterogeneity around instructors' rationales for specific learning design choices (e.g., asking students to set debugging goals to encourage them both to expand their debugging strategies and gauge their own strengths). Similarly, we can consider heterogeneity around how students and instructors respond to specific bugs, for example, by working solely to resolve them or preparing to avoid their recurrence. In addition, when examined over a longitudinal horizon, we see patterns of stability and variation in how a student experiences each failure process (e.g., experiencing short-term growth in debugging strategies or a string of outward expressions of low efficacy). Lastly, judgments about each failure process may be viewed by participants from conflicting points of view and reveal that participants are encountering a partial slice of failure experiences.

We suggest that research on failure can usefully acknowledge this heterogeneity and pursue a number of questions around how to balance these processes surrounding failure given the history, goals, and resources of specific students and learning communities. We unpack some of these implications below.

What constitutes progress in responses to failure?

We return to the question that framed this paper: When teachers, researchers, and parents talk about youth handling failure in the learning process in productive ways, what might they mean? If a well-resourced, expert approach to failure would constitute realizing each of the proposed five processes surrounding failure on a consistent basis, how might we evaluate a learning community's efforts to support students' progress toward that end? A number of considerations come to mind. At the least, we would caution against answers to this question that focus narrowly on one process from this framework, unless learning communities have committed to strongly valuing one over and above the other processes related to failure. For example, the teachers in our learning community formulated more design conjectures around students growing their repertoire of debugging strategies and students expressing authority, suggesting perhaps that those facets of responding to failure were most important to them. Conversely, both students and instructors reported post-hoc that bugs were nearly always fixed and that student confidence was often high, and both case studies suggested that bug fixing was valuable. These considerations raise the prospect that progress with respect to failure can be judged according to the participants' stated goals and/or commonly experienced failure practices. In light of the diverse ways that each characteristic has been studied in the literature (e.g., whether and when it matters to arrive at normative resolutions; when to value self-efficacy; what to consider a recurring problem), there is a case that learning communities' values with respect to failure could drive judgments of progress.

More fundamentally, we would advocate for more research that deepens our understanding of how a student's approach to one process impacts the other processes. From the existing research literature, we know that some of these processes are intertwined, such as efficacy judgments relative to recurring misconceptions (Kallia & Sentance, 2018), or short-term failures preparing students for similar, upcoming problems (Kapur, 2016). Growing our knowledge about interconnections between all five processes, especially as they unfold over time in a learning community, could serve as key resources for educators. This research approach could lead to testable and actionable conjectures, such as whether investing in *general-purpose debugging skills* sets up a student empowered with *high authority to fix a bug*. Conversely, approaches that decenter the value of arriving precipitously at *normatively correct fixes* (Donaldson, 2019; Maheux, 2016; Russ & Berland, 2019) may set students up to focus more squarely on *general-purpose debugging* and

background concerns about *efficacy*. Focusing exclusively on one process might eschew an understanding of how other failure processes are contributing.

A possible starting point in teaching and learning settings would be to value situations in which educators support students along all five processes. Perhaps over time, stronger theoretical predictions and empirical results could guide more nuanced claims about what kinds of early signals in newcomers' practices suggest that the learning community is fully delivering in its failure support. On the other hand, in consideration of definitions of equity that highlight tailored support for students (e.g., Gutiérrez, 2011), an alternative gauge of progress would focus on whether learning communities are supporting students to address the failure processes that students value most and/or judge to be most challenging for themselves specifically. Because it is unreasonable to expect that a learning community could fast-track a student toward growth along all five processes at once, how participants weigh the costs and benefits of addressing certain failure processes is a key future research question to consider.

Pedagogical implications

In considering the pedagogical implications of our research, we would emphasize that educational researchers have previously developed actionable and evidence-based strategies to support each of the five processes we cover in this paper (we describe some of this research in our literature review). We hope this paper raises new questions for educational researchers and teachers that focus on how to balance these processes. These questions include when and in what order each facet should be foregrounded in the learning process, how to pursue them simultaneously, what kinds of supports and resources students could benefit from along the way, and how to formatively take stock of all five processes. Rosebery et al. (2010) argue that “when heterogeneity is deeply engaged, the opportunities for learning multiply” (p. 353). In line with Rosebery et al.'s (2010) pedagogical recommendations, intentionally broadening and attuning educators' attention to these five processes may yield robust support for students across their frequent bouts with bugs.

This balancing act could take place during a single debugging session: How can instructors promote authority, bug fixes, deep learning, high confidence, and debugging strategy growth during a single debugging session? If such support is not possible following a single encounter with a bug, then how can instructors make judgments about which to foreground during a given debugging session? Consider, for example, an instructor working with Maribel (study 3) at different points along her debugging trajectory over three weeks. Beyond considerations about the present bug (e.g., how long it has lasted, what debugging strategies Maribel has used so far), how would knowledge about Maribel's recent self-efficacy judgments, encounters with related bugs, and her overall range of debugging strategies suggest that different failure processes should take priority? We would note that to consider these questions, instructors would need approaches to formatively understanding how students are engaging with each process of debugging. In this way, the failure processes we highlight in this paper could guide instructors' reflective processes over the course of an instructional unit or during day-to-day and moment-to-moment interactions with students.

In addition to using these five processes to guide thinking about instructional supports, instructors could also consider how their pre-planned activity design choices, material designs, and the organization of classroom discourse selectively foreground ways to make progress within each. Our instructors who pursued this through conjecture mapping recognized that different learning designs contributed to the same failure process, and noted in detail the slight variations in how they viewed each valued failure process. It might be useful to think of the proposed five processes as a point of departure to gauge the efficacy of specific classroom designs with respect to a community's goals around failure. As such, instructors could design formative markers of

each process as a way to gather feedback on how students are experiencing failure in their learning environments. Making these characteristics explicit in learning communities could help to further an equitable culture of failure that foregrounds the processes instructors and students aim most to cultivate.

Failure in other domains

Looking beyond computer science, we suggest that these five failure processes might provide a guiding framework for documenting how newcomers to a discipline experience failure, and can serve as a point of departure for thinking about lesson design and assessment choices. Translating these five processes from the specific computer science context described in this paper, we can describe them in general terms in the following way: determine which impasses need to be fixed and when; curtail impasses that continue to arise; learn strategies for debugging novel impasses; engage with authority in the repair process; and calibrate confidence relative to skill, resources, task demands, etc. These processes could be prioritized and specified according to the particulars of a discipline, including informal activities such as playing outdoors as a family (e.g., see Baker et al., 2022). For example, on a given day of instruction, a science educator might devalue a student's science model making accurate predictions, and instead prioritize the student avoiding errors already found in classmates' models while also practicing independence with the process of iteratively tinkering with and assessing the model. These features provide an overall framework for valuing, and perhaps guiding, the teaching of troubleshooting.

Shortcomings and future research

Finally, we discuss weaknesses of the current study and additional openings for future research. One weakness of the current study is its focus on case studies. Future work could aim to understand how students and instructors in other settings, especially instructors with more expertise than those in our learning community, navigate failure over time at the intersection of these five processes. Another weakness of the current study is that we do not granularly examine interconnections between the proposed failure processes. Future research could aim to systematically track where specific configurations of these processes create stronger or weaker opportunities for progress. A third weakness of the current study is that we have only scratched the surface with how to intentionally design learning spaces taking into consideration all five processes. Future design research could involve researchers collaborating with educators to design specifically around these processes of debugging. Lastly, we note that despite our expansive definition of authority from the literature review, we often fell short of evaluating it as a collaborative accomplishment, and instead highlighted independence. Future work should continue to problematize that narrow interpretation.

We also wish to note a few measurement considerations for researchers who utilize and extend this framework. In particular, through the process of developing a coding scheme for study 3, we developed a few key insights/questions about failure. The first covers considerations about what counts as a bug and when the debugging session starts and ends. A range of considerations come into play, such as whether a code flaw was introduced by the student or located in starter code; noticed by the student or as-yet-unseen; changing as new bugs enter the program; repaired with the student noticing the fix or not yet noticing. Each of these considerations reshapes the view of the debugging situation. The second (related) consideration is distinguishing between the observable flaw in the output (the robot going out of bounds or painting the wrong grid cell) and the flaw in the code that caused that observable discrepancy. The tension is that multiple lines of code can in principle be positioned as the flaw, and this open-endedness makes it challenging to

discern whether the programmer addressed the root cause and what would count as bug recurrence (DeLiema et al., 2021).

Conclusion

Through our engagement with the heterogeneity of failure in one learning community, we documented distinct processes that teachers and students pursue and value during debugging. We argue that these processes resist simplification; they do not reduce to each other, but represent genuinely distinct responses to impasses. Moments of failure can serve as catalysts for pursuing fixes, warding off similar failures, inviting practice with general-purpose debugging strategies, centering the contributions of newcomers, and calibrating efficacy, including combinations of these features. Each one of these five processes not only materialized in our study context but consistently played a role in students' and instructors' approaches to failure, both in terms of what they hoped to accomplish and how they worked through debugging. Moreover, when reflecting on their experiences with bugs, students and teachers in the learning community diverged on their interpretation of some processes (e.g., students' confidence and their use of debugging strategies), and described an uneven distribution of experiences with several processes (e.g., fixing bugs more often than leaving bugs unresolved). Failure, in this way, is not simply an occasion for seeking a resolution. Failure is a point of departure for a decision about how and what to foreground and interleave among a range of valued processes. Failure framed as a heterogeneous potential can invite educators, students, and researchers to grapple with central questions about what combination of processes to foreground following failure, how to pursue them, who makes these determinations, and how learning communities calibrate their approach over time.

Acknowledgments

We express our immense gratitude for the students and educators who collaborated on this research, and we thank the thoughtful peer reviewers who guided this work to publication.

Funding

This work was supported by the National Science Foundation under Grant Nos. 1612770, 1607742, and 1612660.

ORCID

David DeLiema  <http://orcid.org/0000-0002-2014-0313>
 Yejin Angela Kwon  <http://orcid.org/0000-0001-8923-0491>
 Andrea Chisholm  <http://orcid.org/0000-0002-2484-799X>
 Maggie Dahn  <http://orcid.org/0000-0001-5982-8169>
 Virginia J. Flood  <http://orcid.org/0000-0003-1808-9923>
 Dor Abrahamson  <http://orcid.org/0000-0003-2883-4229>
 Francis F. Steen  <http://orcid.org/0000-0003-2963-4077>

References

- Adair, J. K. (2014). Agency and expanding capabilities in early grade classrooms: What it could mean for young children. *Harvard Educational Review*, 84(2), 217–241. <https://doi.org/10.17763/haer.84.2.y46vh546h4112144>
- Anderson, A., Goetze, M., Simpson, A. M., & Maltese, A. V. (2019). Where should learners struggle? Developing a failure mindset through maker activities. *Connected Science Learning*, 1(12). <https://www.nsta.org/connected-science-learning/connected-science-learning-october-december-2019/where-should-learners>
- Anderson, C. G. (2020, September). Hits, quits, and retries-player response to failure in a challenging video game. In *International conference on the foundations of digital games* (pp. 1–7). ACM.

- Anderson, C. G., Dalsen, J., Kumar, V., Berland, M., & Steinkuehler, C. (2018). Failing up: How failure in a game environment promotes learning through discourse. *Thinking Skills and Creativity*, 30, 135–144. <https://doi.org/10.1016/j.tsc.2018.03.002>
- Baker, J., DeLiema, D., Hufnagle, A. S., Carlson, S. M., Sharrat, A., & Williams Ridge, S. (2022). Impasses in the wild: Autonomy support in naturalistic, parent-child outdoor play. *Frontiers in Education*, 7, 1–28. <https://doi.org/10.3389/educ.2022.885231>
- Bakhtin, M. M. (1986). *Speech genres and other late essays* (V. W. McGee, Trans.). University of Texas Press.
- Bakker, A. (2018). *Design research in education: A practical guide for early career researchers*. Routledge.
- Bandura, A. (1982). Self-efficacy mechanism in human agency. *American Psychologist*, 37(2), 122–147. <https://doi.org/10.1037/0003-066X.37.2.122>
- Barnett, S. M., & Ceci, S. J. (2002). When and where do we apply what we learn? A taxonomy for far transfer. *Psychological Bulletin*, 128(4), 612–637. <https://doi.org/10.1037/0033-2909.128.4.612>
- Beller, M., Spruit, N., Spinellis, D., & Zaidman, A. (2018). *On the dichotomy of debugging behavior among programmers* [Paper presentation]. Proceedings of ICSE '18: 40th International Conference on Software Engineering, Gothenburg, Sweden. <https://doi.org/10.1145/3180155.3180175>
- Berland, M., Martin, T., Benton, T., Petrick Smith, C., & Davis, D. (2013). Using learning analytics to understand the learning pathways of novice programmers. *Journal of the Learning Sciences*, 22(4), 564–599. <https://doi.org/10.1080/10508406.2013.836655>
- Bindman, S. W., Pomerantz, E. M., & Roisman, G. I. (2015). Do children's executive functions account for associations between early autonomy-supportive parenting and achievement through high school? *Journal of Educational Psychology*, 107(3), 756–770.
- Blikstein, P., Worsley, M., Piech, C., Sahami, M., Cooper, S., & Koller, D. (2014). Programming pluralism: Using learning analytics to detect patterns in the learning of computer programming. *Journal of the Learning Sciences*, 23(4), 561–599. <https://doi.org/10.1080/10508406.2014.954750>
- Bransford, J. D., & Schwartz, D. L. (1999). Rethinking transfer: A simple proposal with multiple implications. *Review of Research in Education*, 24, 61–100.
- Buechley, L., Peppler, K., Eisenberg, M., & Kafai, Y. (Eds.) (2013). *Textile messages: Dispatches from the world of e-textiles and education*. Peter Lang.
- Calabrese Barton, A., & Tan, E. (2019). Designing for rightful presence in STEM: The role of making present practices. *Journal of the Learning Sciences*, 28(4-5), 616–658. <https://doi.org/10.1080/10508406.2019.1591411>
- Cazden, C. (1981). Performance before competence: Assistance to child discourse in the zone of proximal development. *Quarterly Newsletter of the Laboratory of Comparative Human Cognition*, 3, 5–8.
- Cole, M., & Engeström, Y. (1993). A cultural-historical approach to distributed cognition. In G. Salomon (Ed.), *Distributed cognitions: Psychological and educational considerations* (pp. 1–46). Cambridge University Press.
- Collins, A., Brown, J. S., & Newman, S. E. (1988). Cognitive apprenticeship: Teaching the craft of reading, writing and mathematics. *Thinking: The Journal of Philosophy for Children*, 8(1), 2–10.
- Dahn, M., & DeLiema, D. (2020). Dynamics of emotion, problem solving, and identity: Portraits of three girl coders. *Computer Science Education*, 30(3), 362–389. <https://doi.org/10.1080/08993408.2020.1805286>
- Dahn, M., DeLiema, D., & Enyedy, N. (2020). Art as a point of departure for storytelling about the experience of learning to code. *Teachers College Record: The Voice of Scholarship in Education*, 122(8), 1–42. <https://doi.org/10.1177/016146812012200802>
- DeLiema, D. (2017). Co-constructed failure narratives in mathematics tutoring. *Instructional Science*, 45(6), 709–735. <https://doi.org/10.1007/s11251-017-9424-2>
- DeLiema, D., Bye, J., & Marupudi, V. (2021). *Programming instructors and students' active (and partial) debugging: Deviation noticing, causal modeling, and intervening* [Paper presentation]. Annual Conference of the American Educational Research Association, Virtual Annual Meeting.
- DeLiema, D., Dahn, M., Flood, V. J., Asuncion, A., Abrahamson, D., Enyedy, N., & Steen, F. F. (2020). Debugging as a context for collaborative reflection on critical thinking and emotion. In E. Manolo (Ed.), *Deeper Learning, Communicative Competence, and Critical Thinking: Innovative, Research-Based Strategies for Development in 21st Century Classrooms* (pp. 209–228). New York: Routledge.
- Donaldson, M. (2019). Harnessing the power of fantastic attempts: Kindergarten teacher perspectives on student mistakes. *The Journal of Educational Research*, 112(4), 535–549. <https://doi.org/10.1080/00220671.2019.1598329>
- Duckworth, A. L., Peterson, C., Matthews, M. D., & Kelly, D. R. (2007). Grit: Perseverance and passion for long-term goals. *Journal of Personality and Social Psychology*, 92(6), 1087–1101.
- Eccles, J. S., & Wigfield, A. (2002). Motivational beliefs, values, and goals. *Annual Review of Psychology*, 53(1), 109–132.
- Endres, A. (1975). An analysis of errors and their causes in system programs. In M. L. Shooman & R. T. Yeh (Eds.), *ACM SIGPLAN notices - International conference on reliable software* (Vol. 10, pp. 327–336). ACM. <https://doi.org/10.1145/390016.808455>

- 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. https://doi.org/10.1207/S1532690XCI2004_1
- Estabrooks, L. B., & Couch, S. R. (2018). Failure as an active agent in the development of creative and inventive mindsets. *Thinking Skills and Creativity*, 30, 103–115. <https://doi.org/10.1016/j.tsc.2018.02.015>
- Flood, V. J., DeLiema, D., Harrer, B. W., & Abrahamson, D. (2018). Enskilment in the digital age: The inter-ational work of learning to debug. In J. Kay & R. Luckin (Eds.), *Rethinking learning in the digital age: Making the learning sciences count, Proceedings of the 13th International Conference of the Learning Sciences* (Vol. 3, pp. 1405–1406). International Society of the Learning Sciences.
- Foster, C. (2014). Minimal interventions in the teaching of mathematics. *European Journal of Science and Mathematics Education*, 2(3), 147–154. <https://doi.org/10.30935/scimath/9407>
- Freeman, D. N. (1964). Error correction in CORC, the Cornell computing language. In *Proceedings of the Fall Joint Computer Conference - Part I* (pp. 15–34). ACM. <https://doi.org/10.1145/1464052.1464055>
- Fuller, B. (2007). *Standardized childhood: The political and cultural struggle over early education*. Stanford University Press.
- Glaser, B. G. (1965). The constant comparative method of qualitative analysis. *Social Problems*, 12(4), 436–445. <https://doi.org/10.2307/798843>
- Gomoll, A., Tolar, E., Hmelo-Silver, C. E., & Šabanović, S. (2018). Designing human-centered robots: The role of constructive failure. *Thinking Skills and Creativity*, 30, 90–102. <https://doi.org/10.1016/j.tsc.2018.03.001>
- Goodwin, C. (2018). *Co-operative action*. Cambridge University Press.
- Gutiérrez, R. (2011). Context matters: How should we conceptualize equity in mathematics education?. In B. Herbel-Eisenmann, J. Choppin, D. Wagner, & D. Pimm (Eds.), *Equity in discourse for mathematics education: Theories, practices, and policies* (pp. 28–50). Springer.
- Haimovitz, K., & Dweck, C. S. (2016). What predicts children’s fixed and growth intelligence mind-sets? Not their parents’ views of intelligence but their parents’ views of failure. *Psychological Science*, 27(6), 859–869. <https://doi.org/10.1177/09567976166639727>
- Halberstam, J. (2011). *The queer art of failure*. Duke University Press.
- Heikkilä, M., & Mannila, L. (2018). Debugging in programming as a multimodal practice in early childhood education settings. *Multimodal Technologies and Interaction*, 2(3), 42. <https://doi.org/10.3390/mti2030042>
- Hennessy Elliott, C. (2020). “Run it through me.” Positioning, power, and learning on a high school robotics team. *Journal of the Learning Sciences*, 29(4-5), 598–641. <https://doi.org/10.1080/10508406.2020.1770763>
- 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). Harvester Press.
- Hristova, M., Misra, A., Rutter, M., & Mercuri, R. (2003). Identifying and correcting Java programming errors for introductory computer science students. In S. Grissom, D. Knox, D. Joyce, & W. Dann (Eds.), *ACM SIGCSE bulletin* (Vol. 35, pp. 153–156). ACM. <https://doi.org/10.1145/792548.611956>
- Jayathirtha, G., Fields, D., & Kafai, Y. (2018). Computational concepts, practices, and collaboration in high school students’ debugging electronic textile projects. In S.-c. Kong, D. Andone, G. Biswas, T. Crick, H. Ulrich Hoppe, T.-c. Hsu, R. Huang, R. Kwok-yiu, C.-k. Looi, M. Milrad, J. Sheldon, J.-l. Shih, K.-f. Sin, M. Tissenbaum, J. Vahrenhold (Eds.), *International conference on computational thinking education* (pp. 27–32). The Education University of Hong Kong.
- Jefferson, G. (2004). Glossary of transcript symbols with an introduction. In G. H. Lerner (Ed.), *Conversation analysis: Studies from the first generation* (pp. 13–31). Benjamins.
- Jeffries, R. (1982, March). *A comparison of the debugging behavior of expert and novice programmers* [Paper presentation]. Annual Meeting of the American Educational Research Association, New York City, NY, USA. <http://digitalcollections.library.cmu.edu/awweb/awarchive?type=file&item=360539>
- Jordan, B., & Henderson, A. (1995). Interaction analysis: Foundations and practice. *Journal of the Learning Sciences*, 4(1), 39–103. [Database] https://doi.org/10.1207/s15327809jls0401_2
- Juul, J. (2013). *The art of failure: An essay on the pain of playing video games*. MIT Press.
- Kallia, M., & Sentance, S. (2018). Are boys more confident than girls? The role of calibration and students’ self-efficacy in programming tasks and computer science. In A. Mühling & Q. Cutts (Eds.), *WiPSCE ’18: Proceedings of the 13th Workshop in Primary and Secondary Computing Education*, Article 16 (pp. 1–4). ACM.
- Kapur, M. (2008). Productive failure. *Cognition and Instruction*, 26(3), 379–424. <https://doi.org/10.1080/07370000802212669>
- Kapur, M. (2016). Examining productive failure, productive success, unproductive failure, and unproductive success in learning. *Educational Psychologist*, 51(2), 289–299. <https://doi.org/10.1080/00461520.2016.1155457>
- Katz, I. R., & Anderson, J. R. (1987). Debugging: An analysis of bug-location strategies. *Human-Computer Interaction*, 3(4), 351–399. https://doi.org/10.1207/s15327051hci0304_2
- Kim, C., Yuan, J., Vasconcelos, L., Shin, M., & Hill, R. B. (2018). Debugging during block-based programming. *Instructional Science*, 46(5), 767–787. <https://doi.org/10.1007/s11251-018-9453-5>

- Kinnunen, P., & Simon, B. (2011). CS Majors self-efficacy perceptions in CS1: Results in light of social cognitive theory. In M. E. Caspersen, A. Clear, & K. Sanders (Eds.), *Proceedings of the Seventh International Workshop on Computing Education Research (ICER'11)* (pp. 19–26). ACM.
- Kinnunen, P., & Simon, B. (2012). My program is ok—Am I? Computing freshmen's experiences of doing programming assignments. *Computer Science Education*, 22(1), 1–28. <https://doi.org/10.1080/08993408.2012.655091>
- Kitzinger, C. (2012). Repair. In J. Sidnell & T. Stivers (Eds.), *Handbook of conversation analysis* (pp. 229–256). Wiley-Blackwell.
- Klahr, D., & Carver, S. M. (1988). Cognitive objectives in a LOGO debugging curriculum: Instruction, learning, and transfer. *Cognitive Psychology*, 20(3), 362–404. [https://doi.org/10.1016/0010-0285\(88\)90004-7](https://doi.org/10.1016/0010-0285(88)90004-7)
- Ko, A. J., & Myers, B. A. (2004). Designing the Whyline: A debugging interface for asking questions about program failures. In E. Dykstra-Erickson & M. Tscheligi (Eds.), *Proceedings of the ACM Conference on Human Factors in Computing Systems* (pp. 151–158).
- Ko, A. J., & Myers, B. A. (2005). A framework and methodology for studying the causes of software errors in programming systems. *Journal of Visual Languages & Computing*, 16(1-2), 41–84. <https://doi.org/10.1016/j.jvlc.2004.08.003>
- Ko, A. J., & 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). ACM. <https://doi.org/10.1145/1518701.1518942>
- 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. https://doi.org/10.1207/s15327884mca0501_3
- Langer-Osuna, J. M. (2016). The social construction of authority among peers and its implications for collaborative mathematics problem solving. *Mathematical Thinking and Learning*, 18(2), 107–124. <https://doi.org/10.1080/10986065.2016.1148529>
- Lave, J., Murtaugh, M., & de la Rocha, O. (1984). The dialectic of arithmetic in grocery shopping. In B. E. Rogoff & J. E. Lave (Eds.), *Everyday cognition: Its development in social context* (pp. 67–94). Harvard University Press.
- Lave, J., & Wenger, E. (1991). *Situated learning: Legitimate peripheral participation*. Cambridge University Press.
- Lazar, T., Sadikov, A., & Bratko, I. (2018). Rewrite rules for debugging student programs in programming tutors. *IEEE Transactions on Learning Technologies*, 11(4), 429–440. <https://doi.org/10.1109/TLT.2017.2743701>
- Lee, V. C., Yu, Y. T., Tang, C. M., Wong, T. L., & Poon, C. K. (2018). ViDA: A virtual debugging advisor for supporting learning in computer programming courses. *Journal of Computer Assisted Learning*, 34(3), 243–258. <https://doi.org/10.1111/jcal.12238>
- 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), 314–328. <https://doi.org/10.1037/edu0000092>
- 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. <https://doi.org/10.1080/08993408.2017.1308651>
- Lysek, R., & Vahid, F. (2018). Teaching students a systematic approach to debugging. In T. Barnes & D. Garcia (Eds.), *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (pp. 1104–1104). ACM. <https://doi.org/10.1145/3159450.3162222>
- Maheux, J. F. (2016). Wabi-Sabi mathematics. *Journal of Humanistic Mathematics*, 6(1), 174–195. <https://doi.org/10.5642/jhummath.201601.11>
- Maltese, A. V., Simpson, A., & Anderson, A. (2018). Failing to learn: The impact of failures during making activities. *Thinking Skills and Creativity*, 30, 116–124. <https://doi.org/10.1016/j.tsc.2018.01.003>
- Manalo, E., & Kapur, M. (2018). The role of failure in promoting thinking skills and creativity: New findings and insights about how failure can be beneficial for learning. *Thinking Skills and Creativity*, 30, 1–6. <https://doi.org/10.1016/j.tsc.2018.06.001>
- Martin, L. (2015). The promise of the maker movement for education. *Journal of Pre-College Engineering Education Research (J-PEER)*, 5(1), 4. <https://doi.org/10.7771/2157-9288.1099>
- McGee, E. O., & Stovall, D. (2015). Reimagining critical race theory in education: Mental health, healing, and the pathway to liberatory praxis. *Educational Theory*, 65(5), 491–511. <https://doi.org/10.1111/edth.12129>
- Mejía-Arauz, R., Rogoff, B., Dayton, A., & Henne-Ochoa, R. (2018). Collaboration or negotiation: Two ways of interacting suggest how shared thinking develops. *Current Opinion in Psychology*, 23, 117–123. <https://doi.org/10.1016/j.copsyc.2018.02.017>
- Metzger, R. (2004). *Debugging by thinking: A multidisciplinary approach*. Elsevier Digital Press.
- Meuwissen, A. S., & Carlson, S. M. (2019). An experimental study of the effects of autonomy support on pre-schoolers' self-regulation. *Journal of Applied Developmental Psychology*, 60, 11–23. <https://doi.org/10.1016/j.appdev.2018.10.001>

- Michaeli, T., & Romeike, R. (2019). Improving debugging skills in the classroom: The effects of teaching a systematic debugging process. In Q. Cutts & T. Brinda (Eds.), *Proceedings of the 14th workshop in primary and secondary computing education* (pp. 15:1–15:7). ACM.
- Miljanovic, M. A., & Bradbury, J. S. (2017). RoboBUG: A serious game for learning debugging techniques. In J. Tenenbergh, D. Chinn, L. Malmi, A. Korhonen, & J. Sheard (Eds.), *Proceedings of the 2017 ACM Conference on International Computing Education Research* (pp. 93–100). ACM.
- Murphy, L., Fitzgerald, S., Hanks, B., & McCauley, R. (2010, August). Pair debugging: A transactive discourse analysis. In M. E. Caspersen (Eds.), *Proceedings of the Sixth International Workshop on Computing Education Research* (pp. 51–58). ACM.
- 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 2013 International Conference on Software Engineering* (pp. 332–341). IEEE Press.
- Palinscar, A. S., & Brown, A. L. (1984). Reciprocal teaching of comprehension-fostering and comprehension-monitoring activities. *Cognition and Instruction*, 1(2), 117–175. https://doi.org/10.1207/s1532690xci0102_1
- Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books.
- Pea, R. D., & Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New Ideas in Psychology*, 2(2), 137–168. [https://doi.org/10.1016/0732-118X\(84\)90018-7](https://doi.org/10.1016/0732-118X(84)90018-7)
- Pea, R. D., Soloway, E., & Spohrer, J. C. (1987). The buggy path to the development of programming expertise. *Focus on Learning Problems in Mathematics*, 9, 5–30.
- Peppler, K., & Wohlwend, K. (2018). Theorizing the nexus of STEAM practice. *Arts Education Policy Review*, 119(2), 88–99. <https://doi.org/10.1080/10632913.2017.1316331>
- Perscheid, M., Siegmund, B., Taelumel, M., & Hirschfeld, R. (2017). Studying the advancement in debugging practice of professional software developers. *Software Quality Journal*, 25(1), 83–110. <https://doi.org/10.1007/s11219-015-9294-2>
- Polya, G. (1945). *How to solve it*. Princeton University Press.
- Reason, J. (1990). *Human error*. Cambridge University Press.
- Roschelle, J. (1992). Learning by collaborating: Convergent conceptual change. *Journal of the Learning Sciences*, 2(3), 235–276. https://doi.org/10.1207/s15327809jls0203_1
- Rose, M. (2015, May 14). Why teaching kids to have ‘grit’ isn’t always a good thing. *Washington Post; Answer Sheet*. <http://www.washingtonpost.com/blogs/answer-sheet/wp/2015/05/14/why-teaching-kids-to-have-grit-isnt-always-a-good-thing/>
- Rosebery, A. S., Ogonowski, M., DiSchino, M., & Warren, B. (2010). “The coat traps all your body heat”: Heterogeneity as fundamental to learning. *Journal of the Learning Sciences*, 19(3), 322–357. <https://doi.org/10.1080/10508406.2010.491752>
- Russ, R. S., & Berland, L. K. (2019). Invented science: A framework for discussing a persistent problem of practice. *Journal of the Learning Sciences*, 28(3), 279–301. <https://doi.org/10.1080/10508406.2018.1517354>
- Ryan, Z., DeLiema, D., & Abrahamson, D. (2019, April). Understanding instructors’ reflections on conjecture maps and their impact on design-based research. In F. S. Azevedo (Session Organizer), *STEM teacher education and cognition. Roundtable Session Conducted at the Annual Meeting of the American Educational Research Association*, Toronto, Canada.
- Sandoval, W. (2012). Situating epistemological development. In J. van Aalst, K. Thompson, M. J. Jacobson, & P. Reimann (Eds.), *The future of learning: Proceedings of the 10th international conference of the learning sciences* (Vol. 1, pp. 347–354). International Society of the Learning Sciences.
- Sandoval, W. (2014). Conjecture mapping: An approach to systematic educational design research. *Journal of the Learning Sciences*, 23(1), 18–36. <https://doi.org/10.1080/10508406.2013.778204>
- Schiffrin, H. H., Erchull, M. J., Sendrick, E., Yost, J. C., Power, V., & Saldanha, E. R. (2019). The effects of maternal and paternal helicopter parenting on the self-determination and well-being of emerging adults. *Journal of Child and Family Studies*, 28(12), 3346–3359. <https://doi.org/10.1007/s10826-019-01513-6>
- Schwarz, C. V., Reiser, B. J., Davis, E. A., Kenyon, L., Achér, A., Fortus, D., Shwartz, Y., Hug, B., & Krajcik, J. (2009). Developing a learning progression for scientific modeling: Making scientific modeling accessible and meaningful for learners. *Journal of Research in Science Teaching*, 46(6), 632–654. <https://doi.org/10.1002/tea.20311>
- Searle, K. A., Litts, B. K., & Kafai, Y. B. (2018). Debugging open-ended designs: High school students’ perceptions of failure and success in an electronic textiles design activity. *Thinking Skills and Creativity*, 30, 125–134. <https://doi.org/10.1016/j.tsc.2018.03.004>
- Sengupta, P., Dicks, A., & Farris, A. V. (2021). *Voicing code in STEM: A dialogical imagination*. MIT Press.
- Sherin, M., & van Es, E. (2005). Using video to support teachers’ ability to notice classroom interactions. *Journal of Technology and Teacher Education*, 13(3), 475–491.

- Silvis, D., Clarke-Midura, J., Shumway, J. F., Lee, V. R., & Mullen, S. (2022). Children caring for robots: Expanding computational thinking frameworks to include a technological ethic of care. *International Journal of Child-Computer Interaction*, 33, 100491. <https://doi.org/10.1016/j.ijcci.2022.100491>
- Simpson, A., Anderson, A., Maltese, A., & Goeke, M. (2018). "I'm going to fail": How youth interpret failure across contextual boundaries. In J. Kay and R. Luckin (Eds.), *Rethinking learning in the digital age: Making the learning sciences count, 13th International Conference of the Learning Sciences (ICLS)* (Vol. 2, pp. 981–984). International Society of the Learning Sciences.
- Sinha, T., & Kapur, M. (2021). When problem solving followed by instruction works: Evidence for productive failure. *Review of Educational Research*, 91(5), 761–798. <https://doi.org/10.3102/00346543211019105>
- Smith, R., & Rixner, S. (2019, February). The error landscape: Characterizing the mistakes of novice programmers. In E. K. Hawthorne, M. A. Pérez-Quñones, S. Heckman, & J. Zhang (Eds.), *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (pp. 538–544).
- Spohrer, J. C., Soloway, E., & Pope, E. (1985). A goal/plan analysis of buggy Pascal programs. *Human-Computer Interaction*, 1(2), 163–207. https://doi.org/10.1207/s15327051hci0102_4
- Steinberg, S., & Gresalfi, M. (2021). Agency and expressivity in programming play. In E. de Vries, Y. Hod, & J. Ahn (Eds.), *Proceedings of the 15th International Conference of the Learning Sciences - ICLS 2021* (pp. 581–584). International Society of the Learning Sciences.
- Taylor, S. E. (1983). Adjustment to threatening events: A theory of cognitive adaptation. *American Psychologist*, 38(11), 1161–1173. <https://doi.org/10.1037/0003-066X.38.11.1161>
- Tracy, S. J. (2010). Qualitative quality: Eight "big-tent" criteria for excellent qualitative research. *Qualitative Inquiry*, 16(10), 837–851. <https://doi.org/10.1177/1077800410383121>
- Trninic, D., Wagner, R., & Kapur, M. (2018). Rethinking failure in mathematics education: A historical appeal. *Thinking Skills and Creativity*, 30, 76–89. <https://doi.org/10.1016/j.tsc.2018.03.008>
- VanLehn, K. (1988). Towards a theory of impasse-driven learning. In H. Mandl & A. Lesgold (Eds.), *Learning issues for intelligent tutoring systems* (pp. 19–41). Springer-Verlag.
- Vessey, I. (1985). Expertise in debugging computer programs: A process analysis. *International Journal of Man-Machine Studies*, 23(5), 459–494. [https://doi.org/10.1016/S0020-7373\(85\)80054-7](https://doi.org/10.1016/S0020-7373(85)80054-7)
- Vossoughi, S., Davis, N. R., Jackson, A., Echevarria, R., Muñoz, A., & Escudé, M. (2021). Beyond the binary of adult versus child centered learning: Pedagogies of joint activity in the context of making. *Cognition and Instruction*, 39(3), 211–241. <https://doi.org/10.1080/07370008.2020.1860052>
- Vossoughi, S., & Escudé, M. (2016). What does the camera communicate? An inquiry into the politics and possibilities of video research on learning. *Anthropology & Education Quarterly*, 47(1), 42–58. <https://doi.org/10.1111/aeq.12134>
- Wang, X. C., Flood, V. J., & Cady, A. (2021). Computational thinking through body and ego syntonicity: Young children's embodied sense-making using a programming toy. In E. de Vries, Y. Hod, & J. Ahn (Eds.), *Proceedings of the 15th International Conference of the Learning Sciences - ICLS 2021* (pp. 394–401). International Society of the Learning Sciences.
- Watkins, J., Hammer, D., Radoff, J., Jaber, L. Z., & Phillips, A. M. (2018). Positioning as not-understanding: The value of showing uncertainty for engaging in science. *Journal of Research in Science Teaching*, 55(4), 573–599. <https://doi.org/10.1002/tea.21431>
- Whipple, N., Bernier, A., & Mageau, G. A. (2011). Broadening the study of infant security of attachment: Maternal autonomy-support in the context of infant exploration. *Social Development*, 20(1), 17–32. <https://doi.org/10.1111/j.1467-9507.2010.00574.x>
- Wiedenbeck, S. (2005). Factors affecting the success of non-majors in learning to program. In R. Anderson, S. A. Fincher, & M. Guzdial (Eds.), *Proceedings of the First international Workshop on Computing Education Research (ICER '05)* (pp. 13–24).
- Wong, S. S. H., & Lim, S. W. H. (2019). Prevention–permission–promotion: A review of approaches to errors in learning. *Educational Psychologist*, 54(1), 1–19. <https://doi.org/10.1080/00461520.2018.1501693>
- Yin, Y., Wang, Y., Evans, J. A., & Wang, D. (2019). Quantifying the dynamics of failure across science, startups and security. *Nature*, 575(7781), 190–194.
- Zeller, A. (2009). *Why programs fail: a guide to systematic debugging*. Elsevier.

