Human-Centered Process-Oriented Intro to Programming*

Adalbert Gerald Soosai Raj
asoosairaj@ucsd.edu
University of California, San Diego
La Jolla, CA, USA

Ayaan M. Kazerouni
ayaank@calpoly.edu
California Polytechnic State University
San Luis Obispo, CA, USA

Leah Perlmutter
leahperl@cornell.edu
Cornell University
Ithaca, NY, USA

Alex Chao
alexchao@ucsd.edu
University of California, San Diego
La Jolla, CA, USA

Haolin Li
hal180@ucsd.edu
University of California, San Diego
La Jolla, CA, USA

Kyla Ma
kym003@ucsd.edu
University of California, San Diego
La Jolla, CA, USA

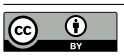
Adrian Nazzal
anazzal@ucsd.edu
University of California, San Diego
La Jolla, CA, USA

Maria Kallia
maria.kallia@glasgow.ac.uk
University of Glasgow
Glasgow, United Kingdom

Abstract

The rise of Generative Artificial Intelligence (GenAI) has fundamentally changed the pedagogy of introductory programming. When students use GenAI to generate code for automated grading pipelines, learning often becomes transactional (i.e., solution-oriented exchange) where the final product is prioritized over the human effort required to learn. The context of this work is an introductory programming course that one author taught, and other authors contributed to the course design. Before changes to our course, assignments were primarily auto-graded, and students began treating them as hurdles to be cleared. Students disengaged, learning communities eroded, and failure rates on exams increased despite students having near-perfect homework scores. In response, we redesigned our CS1 course to be *Human-Centered* and *Process-Oriented* refocusing on the human journey of learning and problem-solving. To restore social interactions in our course, we integrated social learning through TA-facilitated study groups. To restore authentic engagement with the course material, we replaced autograded programming assignments with collaborative, manually graded homework assignments with significant handwritten components. For code-writing tasks, students maintained *process journals* to document their problem-solving journey, including errors encountered and help sought. Preliminary results suggest this approach rebuilt social interactions and restored authentic student engagement. We observed a moderate-to-strong positive correlation between homework scores and exam scores. By replacing machine-led solution-oriented interactions with our human-centered process-oriented approach, we provide a rigorous and meaningful path for introductory programming in the age of GenAI.

*Course website: https://cse8a-6r-fall2025.github.io/



This work is licensed under a Creative Commons Attribution 4.0 International License.
SIGCSE TS 2027, Sacramento, CA, USA
© 2027 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-3075-7/2027/02
https://doi.org/10.1145/3857222.3860223

CCS Concepts

• **Social and professional topics → Computing education**.

Keywords

human-centered pedagogy; process-oriented learning; intro to programming; authentic engagement; social learning

ACM Reference Format:
Adalbert Gerald Soosai Raj, Ayaan M. Kazerouni, Leah Perlmutter, Alex Chao, Haolin Li, Kyla Ma, Adrian Nazzal, and Maria Kallia. 2027. Human-Centered Process-Oriented Intro to Programming. In *Proceedings of the 58th ACM Technical Symposium on Computer Science Education V.1 (SIGCSE TS 2027), February 17–20, 2027, Sacramento, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3857222.3860223

1 Introduction

The tension between performing well on assessments and genuinely mastering course material is not new. It has been documented in education research for decades, from the classic distinction between surface and deep approaches to learning [15] to work on how chasing grades can come at the expense of genuine understanding [6]. The release of Generative Artificial Intelligence (GenAI) in late 2022 sharpened this tension in a way that fundamentally changed how we teach introductory computer science. For the faculty authors of this report, the 2024–2025 academic year raised special challenges in our teaching careers. We observed a new and troubling form of isolation in the classroom. While students had previously shown high levels of engagement and curiosity, many were now utilizing GenAI to complete programming assignments in a way that reduced learning to a superficial exchange, i.e., prioritizing the accumulation of points over the actual mastery of concepts.

By the fall of 2024, this shift reached a breaking point. At UC San Diego, we saw a sudden change in how students approached their work. Instead of seeing programming assignments as a way to learn, many students treated them as chores to be finished as quickly as possible using AI. This created a deceptive sense of success. Students were getting near-perfect scores on take-home

programming assignments, but struggled significantly during proctored exams and skill demonstrations. The results were concerning. In Fall 2024, when the same instructor last taught this course before our redesign, the median assignment score was 94.2%, yet 41.7% of students were on track to receive a D or F under the course's stated grading thresholds. Similar challenges have been reported in other introductory CS courses [5]. Students were turning in code they did not understand, created by tools they did not control.

Our own course structure was partly to blame. In classes with 200–600 students, we relied heavily on automated grading of programming assignments to keep up with the workload. This created a loop where no humans were involved: a student would ask an AI for a solution and then submit it to an autograder for points. No one looked at the student's code or understood their struggle. This lack of real connection made us feel like our assignments had lost their purpose, leading to a deep sense of burnout and making us question the value of take-home assignments. We further noted a significant drop in student-tutor interactions around the same time. At UC San Diego, attendance at optional tutor-led help sessions in intro programming courses plummeted from 30–40 students per week to 3–4 students per week by the fall of 2024. This could be because many students in our intro programming courses may have turned to GenAI for immediate assistance.

We began to find a way forward by looking at the work of our peers and talking to the broader computing education community. At the 2025 ICER conference, the first author met the second and third authors of this report, finding that we all shared the same observations and concerns about our classrooms. A turning point came through an interaction with Maria Kallia (a co-author of this paper), the author of a provocative paper on the need for authentic human engagement in the age of GenAI [12]. Her work argues that we must search for "authentic engagement" to prevent students from becoming "silicon souls" who lack a deep connection to their own learning. These discussions and papers helped us realize that we could not solve this crisis by simply banning or policing the use of GenAI tools. Instead, our team of the first three authors came together to design an intro programming course that fundamentally changed what we valued.

A machine can now write code in seconds, so the value of our class can't be the code anymore. It has to be the human thinking behind it. We decided to shift focus from the final code to the person doing the work. This report describes our new *Human-Centered, Process-Oriented* approach. Our goal was to bring back social interactions and real engagement. In the following sections, we explain how we moved away from a solution-oriented transactional learning toward a model that celebrates the human side of learning to program. This paper focuses on the design and implementation of our course, including handwritten homework, process journals, and study groups, while a deeper analysis of each of these components is beyond the scope of this paper. Each of these components has been used before. Our contribution is combining these known interventions in a meaningful way in a single course. We did this in response to the challenges and opportunities that GenAI has brought to computing education. Instructors can also pick and choose the interventions that may work for their course. Future work will study the impact of individual components and how they interact.

2 Background and Design Goals

Kallia [12] argues that current integrations of GenAI in computing cannot be understood through an instrumentalist lens alone. This risks reducing education to outcome-centered and efficiency-driven models of learning at the expense of fundamental human capacities such as reflective reasoning, learning through struggle and failure, sociality and shared intentionality, and eventually the development of professional identity. We echo Kallia's call to preserve human-centered education. We focus on two aspects of human-centered learning: *social learning* and *process-oriented learning*.

Social learning. Vygotsky's Sociocultural Theory [25] suggests that cognitive development is inherently social and occurs through interaction within the Zone of Proximal Development (ZPD). We have observed that GenAI tools can "collapse" the ZPD by providing final solutions immediately, bypassing the cognitive struggle necessary for learning. We sought to counter this by introducing human-facilitated study groups where TAs and peers could provide the scaffolding for students to internalize core competencies, moving the focus from the machine-generated product back to the socially constructed process of learning [3].

The learning environment and community can positively influence how students learn. Meta-analytic studies (e.g., [11, 22]) suggest cooperative learning and peer connections can increase a student's social integration in the classroom or institution, leading to improvements in psychological health, academic achievement, and persistence. Similar effects have been reported in computing [1, 4, 10, 20]. For example, Cutts et al. [4] reported that "social learning" competencies that are implicit in introductory CS courses, like explaining one's thinking in discipline-appropriate language, appeared to develop felicitously during interactions with other students in "thinkathons" (out-of-class self-paced group revision sessions). Alvarado et al. [1] implemented "micro-classes" in which students worked in staff-led groups of 20–30, and their students reported increased satisfaction and sense of community in the class.

Process-oriented learning. Learning is a process of human development. The learner is—ideally—changed through completing tasks like programming assignments. However, when assignments and grading focus only on product (the completed program), students come to understand that product is the sole objective, and to prioritize answers over the self-changing, often frictional, process of working out a solution themselves.

In the era of GenAI, this risk is particularly acute for courses that rely on autograded programming assignments; homework can become a point collection exercise rather than a meaningful learning activity.

The need to emphasize process is well known in computing education, with numerous efforts to give feedback on intermediate artifacts like examples [26], test cases [7], development plans and processes [8, 14, 27], or submission metrics [24]. After completing written reflections on their developmental processes and interactive lecture sessions, students in CS courses have demonstrated improved metacognitive abilities and programming skills [13, 17, 21, 23].

Taking our cue from Kallia [12], we aimed to frame coursework as an invitation to engage, not merely to comply. We sought to

give students space to explore alternative approaches, make mistakes, and reflect on their learning trajectories. Building upon the work discussed above, we asked students to document the process through which they arrived at a final answer in real time. By clearly signaling to students that we value process alongside product, and mastery over performance, we hope to nudge students toward mastery goal orientations [2, 18] and self-regulatory strategies.

These ideas gave us two design goals. First, we wanted to restore social interactions in our course. Second, we wanted to restore authentic engagement with the course material by valuing process over product. The next section describes how we put these goals into practice.

3 Course Description

We offered our course, *Human-Centered Process-Oriented Intro to Programming*, over 10 weeks at a research-intensive public university in the United States. We taught the in-person course in conjunction with a remote course, sharing some resources such as course staff and videos of in-person lectures. This experience report covers only the in-person course, which had 268 students and 18 staff members, including 1 instructor, 3 graduate teaching assistants (TAs), and 14 undergraduate tutors. The instructor had taught this course 4 times in the past.

We designed the course for students with no prior programming experience, to build proficiency across five core competencies inspired from Porter and Zingaro [19]: 1) Code Tracing, 2) Code Comprehension, 3) Code Writing, 4) Debugging, and 5) Testing.

Our course had three major course components: 1) Collaborative and Process-Oriented Homework Assignments, 2) Competency-Based Process-Oriented Exams, and 3) Social Learning.

3.1 Process-Oriented Homework and Exams

3.1.1 Collaborative and Process-Oriented Homework Assignments. To address the assignment-exam gap described in the introduction and move away from the transactional nature of automated grading, we redesigned our homework assignments to prioritize collaboration, human effort, process orientation and reflection over machine-verified correctness. The course included six homework assignments, distributed as printed or digital worksheets and submitted via Gradescope. Each assignment included 4–5 questions: one code writing question completed in an IDE, and remaining questions covering core competencies such as code comprehension, debugging, and testing, handwritten in the blanks of the worksheet on paper or a tablet. Figure 1 shows a sample homework problem.[1] To enable authentic human collaboration, we encouraged students to work with peers and/or course staff but we still required them to submit their own handwritten work for the non-coding questions.

For code-writing tasks, we required students to keep individual **process journals** to document their problem-solving journey, including errors encountered and help sought from peers or staff. We asked them to document seven steps in their journals: 1) Understanding the Problem (required), 2) Writing the First Version of the Code (required), 3) Running and Testing (required), 4) Debugging and Adjustments, 5) Further Testing and Refinement, 6) Resources Consulted, 7) Final Version of the Code (required). We provided

[1]Sample Homework: https://cse8a-6r-fall2025.github.io/homework-3-solution.pdf

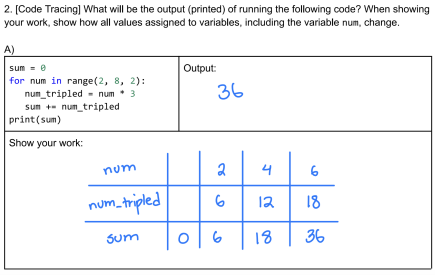


Figure 1: A sample handwritten homework problem.

a sample journal at the start of the term to give students ideas on how to record their process, including artifacts such as photos from handwritten notebooks, IDE screenshots of code and output, typed reflections on approach, and emotions felt (e.g., frustrated, accomplished, happy) at different stages. Figure 2 shows the step 1 (i.e., understanding the problem) from our sample process journal.[2] Students coded, tested, and debugged in an IDE and composed process journals from scratch based on the sample journal.

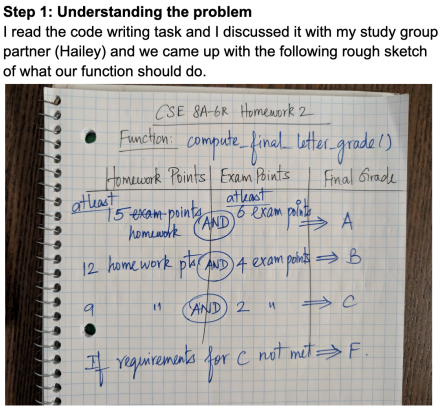


Figure 2: Understanding the problem step in our sample process journal with typed commentary and handwritten work.

3.1.2 Competency-Based Process-Oriented Exams. To ensure authentic assessment, we conducted two high-stakes, in-person exams (midterm and final) with questions parallel to homework questions in structure and format, testing proficiency across all five core competencies. Exam questions required students to show their work (i.e., their process of arriving at their answer).[3]

3.2 Social Learning

We included *social learning* as a core pillar of the course, intending to rebuild student learning communities that GenAI tools like ChatGPT had eroded [9]. Social learning included five weekly events across four activity types: 1) Readings, 2) Active Learning Lectures (twice a week), 3) Pair Programming Labs, and 4) Study Groups.

Our readings and associated quizzes were integrated into a free online interactive textbook.[4] The readings were designed to prepare students for the week's collaborative learning activities during

[2]Process Journal: https://cse8a-6r-fall2025.github.io/sample-process-journal.pdf
[3]Sample Exam: https://cse8a-6r-fall2025.github.io/midterm-exam.pdf
[4]Cogniterra interactive textbook: https://cogniterra.org/course/705/

lectures, labs, and study groups. Every page in the reading had a question to test students' knowledge at the end of the page. Students had an infinite number of attempts to answer these reading questions before the deadline.

Each week, the two 80-minute lectures revolved around a lecture handout[5] that students completed collaboratively during class. Lectures included homework-style problems, which required students to show their process. Students worked on each problem individually, then discussed in pairs or small groups, and finally had a class-wide discussion. Around 10 TAs and tutors were present during every lecture (roughly one staff member per 27 students) to help with student questions during individual and small-group work time. We required students to submit their completed lecture handout on Gradescope at the end of lecture.

Each week included a 50-minute lab where we assigned a programming problem focusing on a core competency. We introduced pair programming at the beginning of the course and required all students to pair program in labs throughout the term.

3.2.1 Study Groups. We replaced traditional, TA-led discussion sections with Study Groups that were designed to move students from passive listening to active, peer-to-peer problem-solving in small groups with staff support.

We divided the class into small, tightly-knit cohorts of 10–15 students, each led by a graduate TA or an undergraduate tutor. The small group size was critical to provide individual attention and ensure no student could remain invisible in the crowd, a common issue in large-scale intro courses. We had 21 study groups that met on various days and times of the week. All students selected a study group at the start of the quarter and attended the same one throughout the quarter. The TA or tutor facilitating each study group asked students what they each wanted to do that week, paired them up or formed small groups accordingly. Although study groups were part of our social learning component, students were allowed to work on their own if preferred.

Per the collaborative nature of the course, these groups served as the primary space for working on homework. While students ultimately submitted assignments individually to maintain personal accountability, we highly encouraged collaboration and discussion. TAs and tutors facilitated these discussions, helping students verbalize their logic and navigate stuck moments together.

By meeting weekly with the same peers and staff member, students built a consistent support network. This structure directly addressed the goal of increasing community engagement, transforming the often-isolated task of learning to code into a shared journey of discovery and struggle.

3.3 Specifications Grading

To align our assessment with the course philosophy of valuing the human process, we replaced traditional percentage-based grading system with *Specifications Grading*. The core philosophy of this system is high standards with multiple tries [16].

Within each homework, individual problems were graded on a binary scale: **1) Pass:** The process (work shown or process journal) is satisfactory and the end product is correct, **2) Not yet:** The process is incomplete/missing or the end product is incorrect. The overall assignment was then given a point value (0 - 3) based on the percentage of "Pass" marks (e.g., 3 points (exemplary) for all problems, 2 points (satisfactory) for at least 80% of problems). To emphasize that learning is a process, we implemented a one-week resubmission window for all homework assignments (except the last). This allowed students to act on manual feedback, correct their mistakes, and earn back points, shifting the focus from "avoiding mistakes" to "learning from them."

Exams assessed the five core competencies using a three-tiered rubric for each question: *Fully Correct (2 points)*, *Mostly Correct (1 point)*, or *Incorrect (0 points)*. Points were totaled and converted into Exam Points (0 - 3) based on percentage thresholds. An *Exam Replacement Policy* let a higher final exam score replace the midterm score, so final grades reflected ultimate mastery rather than initial struggles.

Students also earned up to three social points per week for participating in five weekly events (one reading, two lectures, one lab, one study group); because social learning's value lies in real-time interaction, these points could not be made up asynchronously.

We calculated final letter grade with a two-step "Base + Modifier" system. Minimum thresholds for homework and exam points determined a base grade: A, B, or C. As stated in the syllabus, an "A" required 15/18 homework points AND 6/6 exam points, a "B" required 12/18 homework points AND 4/6 exam points, and a "C" required 9/18 homework points AND 2/6 exam points. At the end of the quarter, the instructor reduced each homework point threshold by one point when computing final letter grades to account for the overall difficulty of the course, while the exam point thresholds remained unchanged. Social Learning points then adjusted the base grade: exemplary engagement earned a plus modifier, satisfactory engagement kept the base grade, and needs-improvement engagement earned a minus modifier — or, if social points fell below a minimum, a full letter-grade reduction.

4 Course Evaluation

4.1 Correlation between Homework and Exams

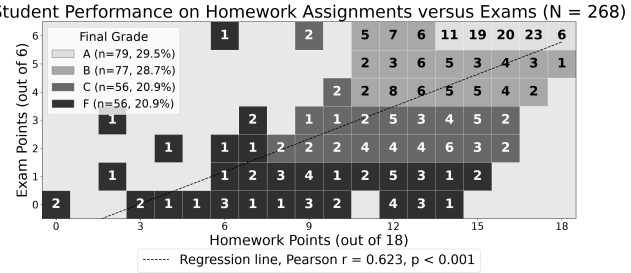


Figure 3: Homework-Exam Alignment and Base Grades. The number in each cell is the count of students at that homework/exam point combination. Each cell is shaded by base grade under the course's adjusted joint homework and exam thresholds (see Section 3.3). The dashed line shows the least-squares regression line (Pearson $r = 0.623$, $p < 0.001$).

With GenAI tools, students can generate correct code for auto-graded assignments with minimal effort or understanding, leading

[5]Lecture Handout: https://cse8a-6r-fall2025.github.io/sample-lecture-handout.pdf

to an anti-pattern of earning near-perfect homework scores yet failing exams. This disconnect motivated our shift to process-oriented, manually graded homework. By requiring students to demonstrate their thinking rather than simply produce correct output, we aimed to better align homework with exams.

The same instructor taught this course in Fall 2024 pre-redesign ($n = 480$ in-person students), grading two components: assignments and assessments. Final grade was min(assignment%, assessment%), a formula similar to the joint threshold of our redesigned course, where students must meet minimum thresholds in both homework and exams for a given base grade. Assessments in this previous course had a replacement policy analogous to our redesigned course's exam replacement policy.

Data from the previous offering illustrates the anti-pattern. Students readily accumulated assignment points for autograded work, with a median assignment score of 94.2%, and 67.5% of students scoring 90% or above. Yet the median assessment score was only 78.3%. Under initial grading thresholds mentioned in the syllabus (i.e., at least 70% needed on both assessments *and* assignments for a C grade), 41.7% of students would have received a D or F, including, strikingly, 31% of students who passed the assignment component but earned insufficient assessment scores. The instructor lowered thresholds to compute final grades, and ultimately 24.8% of students received a D or F. Our redesign aimed to address this anti-pattern of inflated assignment scores masking poor assessment performance.

To assess homework-exam alignment in our redesigned course, we analyzed the relationship between homework points (out of 18) and exam points (out of 6) for all students. The median homework score was 14/18 points and the median exam score was 4/6 points.[6] Figure 3 presents a heatmap of these two measures.

We observed a moderate-to-strong positive correlation between homework and exam performance (Pearson $r = 0.623$, $p < 0.001$, $n = 268$).[7] The heatmap shows that 29.5% of students ($n = 79$) earned a base grade of A, 28.7% ($n = 77$) earned a B, 20.9% ($n = 56$) earned a C, and 20.9% ($n = 56$) earned an F. Under the initial syllabus thresholds, before relaxing homework thresholds by one point, 21.6% of students ($n = 58$) would have received an F. This compares favorably to the previous course, where 41.7% of students would have received a D[8] or F under initial thresholds. Among students who met the adjusted homework threshold (at least 8 of 18 points, $n = 247$), 14% ($n = 35$) nonetheless received an F due to insufficient exam performance (0–1 of 6 points), underscoring that homework success alone did not guarantee a passing grade. The regression line and spread of scores around it reveal that while the two measures are positively associated, the relationship is not deterministic: some students with high homework points earned low exam scores, while a few with limited homework points performed well on exams.

We interpret this positive correlation as encouraging evidence for our human-centered, process-oriented course design. Our homework required students to document their thinking through handwritten work and process journals, practicing the same articulation of computational reasoning that exams assessed. We believe this sustained, process-focused engagement, rather than simply submitting correct code, built the underlying competency that translated to exam performance. The moderate correlation ($r = 0.623$) reflects that exam success depends on more than homework alone; factors such as performance under timed conditions, use of office hours, and study group participation likely also play a role. Nonetheless, the strong statistical significance and direction of the correlation suggest that regular engagement with process-oriented homework meaningfully predicts exam success in our course.

[6] Our course uses specifications grading, so scores are categorical judgments rather than percentages; we do not convert these medians to percentages, as that would imply a granularity the scores lack.

[7] The p-value is the probability of a correlation this large occurring by chance if no true relationship existed.

[8] Specifications grading emphasizes high standards with multiple tries, so we did not define a D grade in our redesigned course.

4.2 Student Sentiments

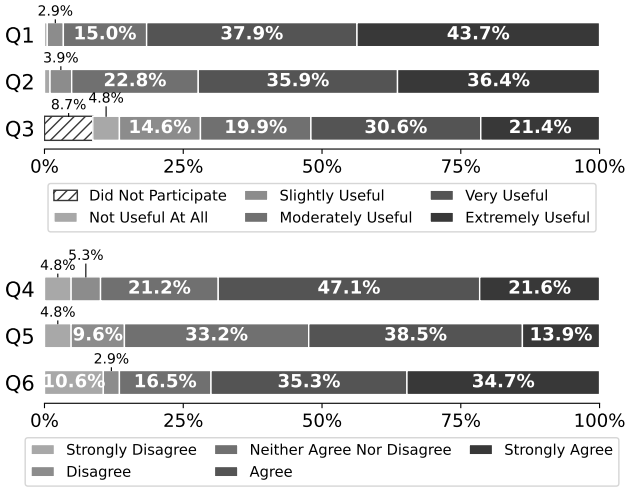


Figure 4: Student Responses to Likert Survey Questions

We surveyed students throughout the term, asking them to reflect on their experiences in the course. We asked seven questions:

- **Q1, Q2, Q3.** For each of the following course components, how useful was it for your learning in this course during the past 3 weeks? (Lectures (Q1), Homework (Q2), Study Groups (Q3)) **Week 4, Likert, 206 respondents.**
- **Q4.** Writing out my programming homework and "show your work" steps by hand helped me understand the underlying logic and structure of the code better than typing it directly on a computer. **Week 6, Likert, 208 respondents.**
- **Q5.** Documenting my progress in the process journal helped me approach programming problems more systematically. **Week 6, Likert, 208 respondents.**
- **Q6.** I would recommend this course in its current form (human-centered, process-oriented) to future students. **End of term, Likert, 170 respondents.**
- **Q7.** Please provide concrete, specific justification for the previous question (Q6). **End of term, Free response, 170 respondents.**

As shown in Figure 4, students generally felt positively about both individual course components and the course overall. Most students indicated agreement that our course components enhanced their learning with a few students expressing negative sentiments. Although it would have been interesting to compare similar student

sentiments from a previous version of the course before the design, we were not able to make this comparison as the previous version of this course did not have written homework, process journals, and study groups.

The fourth author coded the responses to Q7. We present the most commonly appearing codes for both positive and negative sentiments along with sample quotes from the students.

The most common positive responses appreciated the course without directly mentioning the human-centric, process-oriented design of the course. 38 students spoke positively about the course while offering limited reasoning, e.g., *"I had a lot of fun in this course and it helped me learn really well."* 30 students thought that the course was well suited for beginners, e.g., *"I would recommend this course to future students since it's a good starter course for learning how to code."* 21 students thought the course provided a supportive environment. A student responded, *"Basically this course is just my favorite course during this first quarter…I feel strongly accepted and welcome to this course, thanks to the professor and all the tutors and TAs. The course materials are very connected and organized for beginners to learn python, boosting my confidence of moving forwards into computer science :)."*

Some students explicitly described benefits of human-centered, process-oriented course design. 19 students indicated that the specific design of the course assisted their understanding. One student wrote, *"The human-centered approach held me accountable for actually learning and understanding what I needed to know, preventing me from crashing and burning during the midterm and previous homework assignments."* 12 students talked about how they enjoyed the different components of the course, such as *"I like how the process journals invite students to work together to make good work, it almost forces human-like collaborative work. The study groups also forces us to use our own brains, but in an entertaining and stimulating way."*

The most common criticism we received was from 13 students who disliked the grading system. One student wrote that *"I would disagree because the grading system is very very stressful and causes me a lot of anxiety,"* with several others commenting that it was too strict. Several students also wrote that the grading system was their only critique of the class. 9 students also mentioned that the course may not be well-suited to beginners due to the pace of the course and the difficulty of the assignments.

While some of our students may not have fully bought into our human-centered, process-oriented approach, the majority of students still felt positively about the course. A significant number of students expressed appreciation for these aspects, and we believe that this approach was able to accommodate our learners better. We end this section with a notable quote from a student: *"To be honest I took [and failed] this class last year and found that this year I truly began to understand the subject. Having the course be so human centered gave me some many chances to reach out for help. In comparison last year I felt I relied on the AI tutor extension the school gave and without realizing it I was depending on it for understanding. If someone were to ask if they should take [CS1] in this current form or another I would say this form."*

For our quantitative analysis, a GenAI tool (Claude Code) was used to generate the Python code that computed the Pearson correlation and produced Figures 3 and 4. The first author independently computed the Pearson correlation and confirmed it matched the value produced by the generated code, reviewed the generated code, and independently verified all data values depicted in both figures against the underlying dataset.

5 Reflections and Lessons Learned

We shared the redesign rationale, grounded in Kallia [12] and Hou et al. [9], with our course staff. A tight-knit team of TAs and tutors who believed in this mission was essential, turning new policies into a successful community effort.

We also explained in the first lecture that handwritten work and study groups were tools for deep understanding, not arbitrary hurdles.

Grading was a huge part of the course staff's time. Specifications grading helped keep the grading load manageable. Individual homework problems were graded using Pass or Not yet. Process journals were graded by giving feedback only at the first point where the student's work diverged from a correct approach.

Our approach is not against autograding. For the larger code-writing problems in Homework 5 and 6, tutors ran the submitted files through an autograder to check correctness and graded the journals by hand for process. Students did not see the autograder results.

The study groups were a resounding success. They replaced GenAI-driven isolation with a vibrant sense of community. Leading a small group was daunting for tutors who are primarily trained for 1-1 help sessions, though experienced staff managed it effectively.

Although the instructor had taught this course four times before, the process journals gave an unprecedented "window" into students' minds. Reviewing a few journals after each assignment revealed struggles that were invisible in the past, and let lectures adapt in real time. Process journals worked well in this course. They may not work as well in advanced courses, where tasks are far more complicated than writing a single function. In a course on working with large codebases, we had to modify the journal to make it more focused.

6 Conclusion

Our redesign was a direct response to two existential threats facing the modern classroom: eroding student communities and vanishing authentic engagement. By restructuring our course around TA-led study groups and process-oriented homework and exams, we pushed back against the trend of AI interfaces displacing human peers as students' go-to source of help [9]. We also valued the struggle of working through a problem over just producing code, so students stay connected to their own learning rather than becoming "silicon souls" [12]. Critically, the positive correlation between homework and exam scores suggests our redesign helped reduce the rate of near-perfect homework scores co-occurring with exam failure. Ultimately, by valuing human process over final product and community over convenience, we offer a rigorous, meaningful path for computing education in the age of GenAI.

Acknowledgments

This work was supported by the National Science Foundation under Award Nos. 2417787, 2417531, and 2506612. This work was inspired by Hou et al. [9] and Kallia [12].

References

[1] Christine Alvarado, Mia Minnes, and Leo Porter. 2017. Micro-Classes: A Structure for Improving Student Experience in Large Classes. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (Seattle, Washington, USA) *(SIGCSE '17)*. Association for Computing Machinery, New York, NY, USA, 21–26. doi:10.1145/3017680.3017727

[2] Carole Ames and Jennifer Archer. 1988. Achievement goals in the classroom: Students' learning strategies and motivation processes. *Journal of educational psychology* 80, 3 (1988), 260. doi:10.1037/0022-0663.80.3.260

[3] Mordechai Ben-Ari. 2001. Constructivism in computer science education. *Journal of computers in Mathematics and Science Teaching* 20, 1 (2001), 45–73.

[4] Quintin Cutts, Matthew Barr, Mireilla Bikanga Ada, Peter Donaldson, Steve Draper, Jack Parkinson, Jeremy Singer, and Lovisa Sundin. 2019. Experience Report: Thinkathon – Countering an "I Got It Working" Mentality with Pencil-and-Paper Exercises. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '19)*. ACM, 203–209. doi:10.1145/3304221.3319785

[5] Litong Deng. 2026. Failing grades soar as professors see greater AI usage, dwindling math skills in UC Berkeley computer science classes. https://www.dailycal.org/news/campus/academics/failing-grades-soar-as-professors-see-greater-ai-usage-dwindling-math-skills-in-uc-berkeley/article_16fad0bf-02cb-4b8c-8d88-888ffd9f8608.html

[6] Carol S Dweck. 1986. Motivational processes affecting learning. *American psychologist* 41, 10 (1986), 1040.

[7] Stephen H. Edwards. 2004. Using software testing to move students from trial-and-error to reflection-in-action. In *Proceedings of the 35th SIGCSE technical symposium on Computer science education (SIGCSE04)*. ACM, 26–30. doi:10.1145/971300.971312

[8] Katrina Falkner, Rebecca Vivian, and Nickolas J.G. Falkner. 2014. Identifying computer science self-regulated learning strategies. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education* (Uppsala, Sweden) *(ITiCSE '14)*. Association for Computing Machinery, New York, NY, USA, 291–296. doi:10.1145/2591708.2591715

[9] Irene Hou, Owen Man, Kate Hamilton, Srishty Muthusekaran, Jeffin Johnykutty, Leili Zadeh, and Stephen MacNeil. 2025. 'All Roads Lead to ChatGPT': How Generative AI is Eroding Social Interactions and Student Learning Communities. In *Proceedings of the 30th ACM Conference on Innovation and Technology in Computer Science Education V. 1* (Nijmegen, Netherlands) *(ITiCSE 2025)*. Association for Computing Machinery, New York, NY, USA, 79–85. doi:10.1145/3724363.3729024

[10] Wei Jin, Tacksoo Im, Xin Xu, Evelyn Brannock, and Hyesung Park. 2025. A Holistic Approach to Improve Learning and Motivation in CS1. In *2025 IEEE Integrated STEM Education Conference (ISEC)*. IEEE, 1–5. doi:10.1109/isec64801.2025.11147288

[11] David Johnson, Tangmit Johnson, and M Stanne. 2000. Cooperative learning methods: A meta-analysis. (01 2000).

[12] Maria Kallia. 2025. To Be, or to Be Otherwise? Silicon Souls in Search of Dasein and Authentic Human Engagement in the Age of Generative AI. In *Proceedings of the 2025 ACM Conference on International Computing Education Research V.1 (ICER '25)*. ACM, 240–255. doi:10.1145/3702652.3744223

[13] Jiwon Lee, Ayaan M. Kazerouni, Christopher Siu, and Theresa Migler. 2023. Exploring the Impact of Cognitive Awareness Scaffolding for Debugging in an Introductory Programming Class. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 1007–1013. doi:10.1145/3545945.3569871

[14] Joshua Martin, Stephen H. Edwards, and Clfford A. Shaffer. 2015. The Effects of Procrastination Interventions on Programming Project Success. In *Proceedings of the eleventh annual International Conference on International Computing Education Research (ICER '15)*. ACM, 3–11. doi:10.1145/2787622.2787730

[15] Ference Marton and Roger Säljö. 1976. On qualitative differences in learning: I—Outcome and process. *British journal of educational psychology* 46, 1 (1976), 4–11.

[16] Linda B Nilson and Claudia J Stanny. 2023. *Specifications grading: Restoring rigor, motivating students, and saving faculty time*. Routledge.

[17] Chris Park. 2003. Engaging Students in the Learning Process: The learning journal. *Journal of Geography in Higher Education* 27, 2 (2003), 183–199. doi:10.1080/03098260305675 arXiv:https://doi.org/10.1080/03098260305675

[18] Paul R. Pintrich. 2000. *The Role of Goal Orientation in Self-Regulated Learning*. Elsevier, 451–502. doi:10.1016/b978-012109890-2/50043-3

[19] Leo Porter and Daniel Zingaro. 2024. *Learn AI-assisted Python programming: with github copilot and ChatGPT*. Simon and Schuster.

[20] Noah Ravetch, Naomi Nayman, Jack Herberger, Theresa Migler, and Ayaan M. Kazerouni. 2026. Peer Networks and Instructor Resources as Complementary Support Systems in Six CS Classrooms. In *Proceedings of the 31st ACM Conference on Innovation and Technology in Computer Science Education V. 1* (Madrid, Spain) *(ITiCSE 2026)*. ACM, New York, NY, USA, 7. doi:10.1145/3803400.3809329

[21] Jeffrey A. Stone and Elinor M. Madigan. 2007. Integrating reflective writing in CS/IS. *SIGCSE Bull.* 39, 2 (June 2007), 42–45. doi:10.1145/1272848.1272881

[22] Vincent Tinto. 1975. Dropout from Higher Education: A Theoretical Synthesis of Recent Research. *Review of Educational Research* 45, 1 (1975), 89. doi:10.3102/00346543045001089

[23] Tammy VanDeGrift, Tamara Caruso, Natalie Hill, and Beth Simon. 2011. Experience report: getting novice programmers to THINK about improving their software development process. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (Dallas, TX, USA) *(SIGCSE '11)*. Association for Computing Machinery, New York, NY, USA, 493–498. doi:10.1145/1953163.1953307

[24] Maureen M. Villamor. 2020. A review on process-oriented approaches for analyzing novice solutions to programming problems. *Research and Practice in Technology Enhanced Learning* 15, 1 (April 2020). doi:10.1186/s41039-020-00130-y

[25] Lev S Vygotsky. 1978. *Mind in society: The development of higher psychological processes*. Vol. 86. Harvard university press.

[26] John Wrenn and Shriram Krishnamurthi. 2019. Executable Examples for Programming Problem Comprehension. In *Proceedings of the 2019 ACM Conference on International Computing Education Research (ICER '19)*. ACM, 131–139. doi:10.1145/3291279.3339416

[27] Benjamin Xie, Jared Ordona Lim, Paul K.D. Pham, Min Li, and Amy J. Ko. 2023. Developing Novice Programmers' Self-Regulation Skills with Code Replays. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1* (Chicago, IL, USA) *(ICER '23)*. Association for Computing Machinery, New York, NY, USA, 298–313. doi:10.1145/3568813.3600127