Designing Courses for Liberal Arts and Sciences Students Contextualized around Creative Expression and Social Justice

Mark Guzdial
Tamara Nelson-Fromm
mjguz@umich.edu
tamaranf@umich.edu
University of Michigan
Ann Arbor, Michigan, USA

Abstract

The goal of teaching everyone computing (explicitly including programming) predates the definition of the computer science (CS) major and even the prospect of a software development career. At the University of Michigan, we are creating courses for non-CS majors which are grounded in the computational practices of liberal arts and sciences faculty. These courses have no connection to the CS major curriculum or software development jobs. We focus here on two of the themes that those faculty valued (Computing for Expression and Computing for Justice) and the introductory courses that we designed around each theme. The courses emphasize gaining broad perspectives of computing, which serve the study of multiple disciplines. Student activities include readings, writing essays, classroom discussion, and open-ended programming homework assignments. This experience report describes our design process, the Creative Expression and Social Justice courses, and an initial evaluation of our design. Most of the programming assignments were written in the block-based programming language Snap!, with some in-class exercises using teaspoon languages. Several units ended with an ebook assignment to connect the Snap! programming to equivalent programs in Python, Processing, and SQL. Interview and survey findings suggest that students found this sequence and the courses *useful*, despite not counting toward a CS major or focusing on early software development skills. Students described usefulness in terms of developing general computing knowledge, preparation for a range of future careers, and introducing them to other course choices.

CCS Concepts

• **Applied computing** → **Computers in other domains**; **Arts and humanities**; **Education**; • **Social and professional topics** → **Computing education**.

Keywords

liberal arts and sciences, computational literacy, CS for All, computational thinking, digital humanities, critical computing, computational science



This work is licensed under a Creative Commons Attribution International 4.0 License.

SIGCSE TS 2025, February 26-March 1, 2025, Pittsburgh, PA, USA
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0531-1/25/02
https://doi.org/10.1145/3641554.3701896

ACM Reference Format:
Mark Guzdial and Tamara Nelson-Fromm. 2025. Designing Courses for Liberal Arts and Sciences Students Contextualized around Creative Expression and Social Justice. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE TS 2025), February 26-March 1, 2025, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3641554.3701896

1 Meeting the Goals of Computing for All

Computing education for *everyone* includes students who will not join the technology industry or become professional software developers. The goal of providing everyone with computing education, including learning to program, pre-dates software development as a career. Scholar C.P. Snow and ACM Turing Award laureates Alan Perlis and Peter Naur argued for teaching *everyone* how to program in the 1960s [18, 21]. Standards for a computer science (CS) undergraduate major were defined in 1968, and the linear sequence of courses starting with "CS1" was defined in 1978 [38]. Undergraduate CS standards today are focused on developing CS majors [25]. Snow, Perlis, and Naur saw a value in learning to program for other reasons and while studying other disciplines [3]. For example, Perlis argued that programming would develop students' *procedural literacy* and support them in domain-specific tasks like building models and simulations [30]. We are exploring the development of courses that meet the needs of *non-CS majors* as a step towards the goals of computing education for all in the form described by Snow, Perlis, and Naur.

The contribution of this paper is to provide an exemplar process for developing courses based around alternate endpoints for computing education [39]. We start with a participatory design process where we invited non-CS faculty to tell us what should be in the introductory courses for students in their discipline. We describe the components of two of our courses briefly, as a model for how we mapped the design process to implementation. We use student evaluation surveys and interviews to test our hypotheses about our design. We wanted to know if students found it useful to have a programming course outside of the CS sequence, how they felt about programming, and what computing topics they would want to learn about next.

Prior work: Our work is unusual for focusing on non-CS majors and on non-CS careers. Prior work has explored non-major programming courses which have been designed to fulfill a university-wide CS requirement [12, 23] or a department-specific degree requirement [26, 36]. Some prior examples have served as more accessible versions of a CS-major introductory course [2, 10, 42] (which

may focus on some disciplinary applications of programming). We are creating courses that are valuable to non-CS majors, but do not count as a credit towards a CS major. This is a challenging task – to set aside over 50 years of development of CS curricula for the CS course sequence, and to re-think what computing education might mean for majors other than CS, following Snow, Perlis, and Naur.

A 2017 edited volume explored computing education in the liberal arts with a focus on developing an interdisciplinary perspective for future computing professionals [14]. These contrast with our goals to provide computing education for future practitioners of arts, sciences, and humanities. As we will see in the results, some of our students want to work in computing, but in different ways than CS majors.

Our effort is closer to courses which teach general education mathematics using programming [17] . Our courses fit into science or humanities scholarship as general education electives, where the goal is to teach students about the computing needed to succeed within their chosen discipline. We are not designing "CS0" courses, as that numbering still suggests a course at the beginning of a linear sequence leading to a CS major [29] and focuses on preparing students for further CS courses.

2 A Participatory Design Process

Our process began with a task force in our College of Literature, Science, and the Arts (LSA) at the University of Michigan. The task force was charged by the LSA Dean to identify the needs for computing education within the College. The task force conducted several investigations, including interviews with faculty, meetings with the student government curriculum committee, and data mining across the College course catalog and syllabi [22]. The task force identified three themes that described how LSA faculty used and explored computing:

- **Computing for Discovery:** Computational methods provide new avenues for exploration in natural and social sciences, as Perlis described [34].
- **Computing for Expression:** Computing changes how we communicate and creates opportunities for new forms of expression. Liberal arts and sciences scholars use, invent, and study these new forms.
- **Computing for Justice:** Computing has influence on social justice and equity. To understand that influence and to imagine a different future requires computing education, as Snow and Naur argued [18, 21].

A survey across LSA faculty showed broad support and connection to these themes. Faculty particularly resonated with the goal of supporting their students as *conversational programmers*. Researchers have found that many non-CS majors taking introductory computing courses are interested in learning *what* professional software developers do and *how to talk* with them about programming, but don't want to become software developers or spend significant time programming [6, 7, 9, 41]. Besides becoming conversational programmers, LSA faculty wanted to support students who were going to use programming for art, for communication, and as a research method in humanities and sciences.

After the task force submitted its final report [22], the College decided to create a new program to offer courses and credentials supporting LSA students' needs for computing education, the *Program in Computing for the Arts and Sciences* (PCAS). The program set out to create introductory courses that included programming for each of the three themes. The first three courses were on Creative Expression, Social Justice, and Scientific Discovery, as useful subsets of the themes from the task force. We focus on just the Expression and Justice courses in this paper, because they were the first two courses established and because we used a similar process for each of them.

From the task force investigations, we could identify faculty who worked in those two themes. We held participatory design sessions [11] around Expression and Justice (separately) with faculty who saw their work as relating to that theme. We asked them to identify learning goals and useful homework assignments for these courses. These were on-line sessions where the participants organized a shared whiteboard with their goals for the courses.

In one activity, we filled a whiteboard with a couple dozen post-it notes describing activities or learning objectives that appeared in the task force report. We asked the participants to move to the *right* those activities or learning objectives that the participants felt were *important* for the Expression or Justice courses, and to move to the *left* those activities or learning objectives that were *less important*. They could also delete or add new learning objectives or activities. We captured the transcript of the discussion where the participants weighed the importance of the post-it notes as they moved them. Figure 1 is a snapshot of the upper-right corner of the "Justice" whiteboard[1].

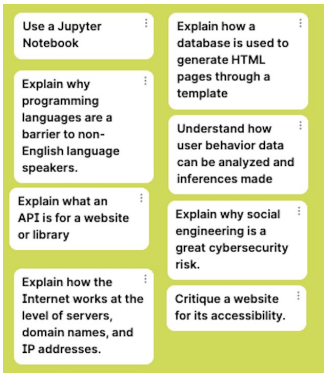


Figure 1: A snapshot of the upper right corner of the "Justice" learning goals whiteboard

We transcribe here the top five goals (in terms of proximity to the upper-right of the whiteboard) for Justice which were:

- Explain how a database is used to generate HTML pages through a template.
- Use a Jupyter notebook.
- Explain why programming languages are a barrier to non-English language users.
- Understand how behavior data can be analyzed and inferences made.

[1]The full image of the final-state whiteboard from both activities can be viewed at https://computinged.wordpress.com/2023/07/07/participatory-design-to-set-standards-for-pcas-courses/

- Explain why social engineering is a great cybersecurity risk.

We did a similar process for Expression. The top five goals (again, in terms of proximity to the upper right of the whiteboard) for Expression were:

- Build a web page in HTML and CSS.
- Build an image filter in some programming language.
- Know the difference between C++, Python, and Snap!.
- Write the program "Hello World" in a textual language.
- Describe how sounds and pictures can be represented in numbers or bits.

We note that these two lists are significantly different. For example, the goal *"Write the program 'Hello World' in a textual language"* is far-right for Expression, but was moved far-left for Justice. The lists are also different than what one would expect for CS0 or CS1 courses. One of the post-it notes reads *"Write secure, safe, and robust code"*, and both the Expression and Justice participants put that goal on the left half of their whiteboards.

The transcripts help us to interpret these goals. For example, learning how a database can be used to generate a website was not an obvious Justice goal for us. Discussions with the faculty helped us understand the goal. The Justice faculty know that much of the Web is generated from a database, and they wanted their students to think about how easy it is to generate a website from a database, who might create the database for a given website, and what agenda motivates those actors to generate the data and website.

Overarching messages from the participatory design sessions were "Don't overwhelm the students" and "They won't want mathematics." Our participants were not concerned about preparing students for future technology industry careers (though they were open to it). They were more concerned that students might be 'scared off' from ever wanting to program again.

3 Components of the Courses

The results from participatory design sessions (whiteboards and transcript) became the *standards* for the Creative Expression course ("Computing for Creative Expression") and Social Justice course ("Computing's Impact on Justice: From Text to the Web"). We did not fit all of the standards in a single course, but the standards inform the design of follow-on courses.

We describe the courses here, to show how we implemented the standards, but the contribution of this paper is not the courses themselves. Some readers may want to implement these exact courses. Our focus is on the design process (including participatory design) that implement stakeholders' input. We additionally present an evaluation that asks whether students might be interested in courses that are apart from the CS major and are driven by LSA computational needs instead of technology industry and CS research needs. This process could be replicated at other institutions to create *unique* courses that meet the needs of that institution's non-CS students.

Both courses required students to engage in bi-weekly quizzes, readings, and written reading reflections, as is common in liberal arts courses. In the Justice course, readings included the introduction to *Algorithms of Oppression* [32], a blog post on the myth of Target predicting a teenager's pregnancy, and a science fiction story about criminal chatbots. In the Expression course, students read about data artists [40] and the copyright issues around generative AI [35]. We included videos in the "readings" category. The Justice class watched a talk on why voice assistants should speak Spanglish [33]. The Expression class watched videos of computational artists describing their work, such as Golan Levin [27].

Our goal with the programming was to be accessible, even for students with little programming or mathematics background. We were less concerned with teaching professional programming skills. As our participants requested, we wanted students to gain the confidence to increase their programming skills in the future. Consequently, we chose to use non-traditional programming media. Programming activities in the courses had three forms, following the structure presented by Guzdial at ITiCSE 2023 [19, 20]. Some topics had in-class programming activities with a teaspoon language to introduce the computational ideas. Homework was in Snap! [1]. Students were given starter projects in Snap! for every homework, so that they had multiple worked examples [37] and had access to custom blocks to scaffold the activities. Many topics ended with the use of an ebook [13] in which students would see Snap! code from class, and then the same algorithm implemented in a textual language (like Python, Processing, or SQL). Students answered multiple-choice questions about the textual program, graded as participation rather than correctness. In this way, programming activities were scaffolded, and students were prompted to transfer their knowledge into more mainstream languages [24]. Some homework involved use of other computational tools, such as Twine [16] and CODAP [15], in response to faculty requests for Web adventure games and visualization tools, respectively.

We had no lectures explicitly focused on programming concepts such as variables, functions, or loops. Where we talked about programming concepts, we folded them into lectures on thematic concepts. We were using non-traditional programming media with low cognitive load (blocks and teaspoon languages), and we provided a rich set of examples in starter projects. Students could get started by modifying existing code, so loops, variables, and conditionals were an implicit part of the programming, rather than an explicit topic.

3.1 Social Justice

Table 1 lists the major units of the Justice course with the student programming activities. The premise of the class was to trace the humanities scholars' interest in text from ways of representing text and language on the computer (e.g., going back to ASCII and its limitations compared to Unicode), to the Web, and automatic processing of Web pages to generate data (which could then be analyzed and visualized). A challenging part of the design for this course was introducing database concepts and the generation of Web pages from a database – without dealing with servers and HTML syntax. We decided to make heavy use of Snap! by creating SQL-like blocks and HTML-generating blocks, which could be knit together to create Web pages from databases.

3.2 Creative Expression

Table 2 lists the units for the Expression course. Each unit started with an exploration of artists' work with the unit topics: drawing basic shapes, image filters, sound filters, interaction, video games,

Table 1: Topics and activities in the Social Justice course

Topics (weeks)	Lab and Homework activities	Ebook Chapter
Representing text and language (2)	Recognize and generate sentences using a teaspoon language Generate haiku in Snap!	
Chatbots (1)	Create a chatbot in a teaspoon language Create a chatbot in Snap!	Generate language and respond to user input in Python
Web (2)	Build a simple Web page using Snap! Build a Twine adventure game	HTML
Data (1)	Build a catalog Web page from a database using Snap!	Databases and SQL
Privacy (2)	Analyze a log file in Snap! to find clues to a mystery	
Visualizations (2)	Build a visualization to make an argument using CODAP	
AI (3)	Build a gesture recognizer in Snap!	Basics of programming in Python

and chatbots. Students were asked to sketch their output before they started, and to share their sketches and partial products in an online forum. The goal was to create the sense of a design studio. Many of these units involved using custom blocks to make digital sound, image, and text manipulation easier.

4 Evaluating the Design

In our first year of the course (two offerings of each course in each semester), we wanted to know if the *design* was working. In summative evaluation, we want to know about outcomes for students from the class, such as what students *learned* and whether the students increased their *self-efficacy* about programming. But during the first year (two terms) of the course, enrollment was low, both because they were new classes and because they were not a specific requirement for any majors. Enrollment numbers can be found in Table 3. Our first year evaluation was formative. We wanted to know if we were on a reasonable path, or if our idea of building courses completely separate from the CS major was failing.

The course enrollments matched the general demographics of the LSA College. We had a few computer science majors in the classes, but most of the majors came from across the LSA College, including philosophy, political science, music, and psychology. Our institution collects demographic data about enrolled students, which we were able to access as a part of our human subjects study. Among students enrolled in either course across the first two terms, 53% were female and 47% were male. Our institution collects gender data as binary. This likely does not fully represent our students, as a portion of students used pronouns other than she/her or he/him in class and one student self-identified as non-binary during interviews. Additionally, 20% of the students were reported as a member of a racial or ethnic group traditionally under-represented at our predominantly white institution.

The evaluation questions (EQ) that we address here are:

- EQ1: Would students find the class *useful*? Or would they see it as a toy course, a waste of time, or a dead-end?
- EQ2: Did students increase their sense of being a *conversational programmer*?
- EQ3: How would they find the use of *Snap!*? Would they see it as a toy, or a productive help?
- EQ4: What would they want to do next in terms of *courses* or *career*? Did they feel that they could *learn more programming later*?
- EQ5: Did students see the *ebooks* useful in terms of setting them up for later learning?
- EQ6: Did the students find the class too easy, or was it *harder* than we anticipated – or even *overwhelming* as some have reported for non-major programming courses [28]?

4.1 Methods: Survey and Interviews

We had two sources of data to answer these questions. First, we added our own questions to the institutional evaluation of teaching for both courses. Participation rates in the evaluation were high in both Expression (90% and 75% in the two terms represented) and Justice (61% and 70%). For several of our questions above, we included a survey question that directly responded to a question on a Likert-scale (where 5 is Strongly Agree and 1 is Strongly Disagree). We also included some open-ended questions with written responses, including "What would you tell students who are considering taking this course?"; "Which aspects of this course did you like best?"; and "Which aspects of this course did you like least?"

Second, we had a human subjects study approved as exempt by our institutional review board to interview students about their reasons for taking the courses and their processes of programming. We interviewed 17 students from across the two courses. That full study is yet unpublished. We have used the coded transcripts of those interviews in answering our more complicated questions. Transcripts were coded inductively by the second author, following the strategy defined by Coffey and Atkinson [8] which emphasizes generating conceptual categories based on the data as a part of the analysis, with the created codes not being the final output of the analysis but just the first step.

Our method for this report was a simplistic form of thematic analysis [5]. We examined the teaching evaluation survey comments and the coded data from the interview transcripts for references to the items highlighted in the questions in the previous subsection.

Table 2: Topics and activities in the Creative Expression course

Topics (weeks)	Lab and Homework activities	Ebook Chapter
Drawing shapes (2)	Generate shape-based art in Snap!	
Image Processing (2)	Build image filters with teaspoon language Create a collage with personal images and custom image filters built in Snap!	Drawing and repetition in Processing
Digital Sound (2)	Build an auditory scene in Snap!	Image processing in Python
Interactivity (2)	Build a drawing program that uses the mouse and either camera or microphone using Snap!	Turtle graphics in Python
Video games (2)	Build a side-scroller video game in Snap!	
Text and language (1)	Build a chatbot including image and sounds in Snap!	
Web (1)	Build a webpage using including CSS and interactive elements Snap!	Conditionals and loops in Python

Table 3: Course Enrollment and Interview Participation

	Term 1	Term 2	Total
Justice Enrollment	14	11	25
Expression Enrollment	10	39	49
Interview Participants	5	12	17

We used those quotes, as well as the conceptual categories which they had already been assigned, to identify themes in students' responses. Following the reflexive [4] school of thematic analysis, we developed themes in conjunction and reconciled any disagreements in interpretation, in contrast to categorizing data separately and recording numerical measurements of researcher agreement such as inter-rater reliability.

4.1.1 Limitations: One of the authors was the developer and instructor in both classes for the first terms, as well as directed the research. The other author is a graduate student who is studying the motivation and process of the students in the class, and was involved in the development of course assignments. We are biased observers, but we hope that our methods help us to identify weaknesses in the class, and not just the strengths we hope to see. The population of the courses is small, and students volunteered to complete the course evaluation or to participate in interviews. We cannot generalize our results to a broad population of students. However, we believe that these students responses are enough to tell us if we are going in completely the wrong direction.

4.2 Findings

Our evaluation questions about the students' ability to learn programming later (EQ4), being a conversational programmer (EQ2), and ebooks (EQ5) were directly addressed with Likert-scale questions (on a scale of 1 to 5) in the evaluation survey[2]. We report below the average score for each question in each of the four course offerings we are describing.

- **This course helped me feel more confident about my ability to learn programming**: Expression Term 1: 4.9, Expression Term 2: 4.5, Justice Term 1: 4.8, Justice Term 2: 4.8
- **This course made me more comfortable talking about programming**: Expression Term 1: 4.9, Expression Term 2: 4.5, Justice Term 1: 4.8, Justice Term 2: 4.8
- **The ebook chapters helped me see how what I am learning applies to the programming languages used by professionals**: Expression Term 1: 4.9, Expression Term 2: 4.6, Justice Term 1: 4.9, Justice Term 2: 4.6

We address the other evaluation questions with the thematic analysis which resulted in identified themes.

Useful: Students found their class useful (EQ1), in some cases because they wanted to gain specific programming skills. Some students appreciated developing new perspectives. Others liked just learning to "make cool things." Some students were explicitly driven by their career prospects, but few reported interest in a software development career. Some of these students (see second quote below) also pointed out how the courses connected to later courses (EQ4).

- "So this has definitely really helped see like how [computing] will be useful and like how it's like changed history and like how it will help like, make cool things."
- "I should know how like the web tracks me and yeah. How these algorithms work and stuff...I feel like a lot of the thinking I've done for this class I've done for, I've had to apply to my other classes. "
- "It seems like a very useful skill to have just going forward, if not for, um, specific jobs or anything, just for being able to navigate like online spaces and do stuff online."

Career: EQ4 was about whether students found the courses useful for future courses or career. The career answer is complicated because liberal arts and sciences majors are interested in a wide variety of careers, with several explicitly saying things like (direct quote): "Well then again, I don't want a computer science related career."

Some students expressed interest in a career that would involve some amount of computation, but they were still exploring their goals and how they wanted to have impact. The course helped students see perspectives that they hadn't considered:

[2]Our institution did not provide distributions on the Likert-scale questions

- "Hopefully I become like a data analyst/scientist at some environment place. But I mean, after that I hope I don't do that my whole life, you know? ...I mean like programming would be cool also and finding kind of like a way to tie like more art stuff kind like what this class is into that. Um, just because like all the TED Talks <*the instructor*> has made us watch, it's just like, whoa, I didn't even know you could like make your career out of this...So that's really cool. "
- "Yes. I just wanna like have a career like that's kind of techy but then like also know that it's like doing some good and so that's kind of why I want to do a sustainability minor, just so then it's like I know that I can do something that's good for the planet."

Courses that come next: EQ4 was also about whether students saw these classes helping with later classes. Students reported that they were excited by the ideas connecting these courses to later courses, not just in computer science.

- "I like the concepts related to data science. Like today in the lecture the professor told us about complex system courses that are going to be offered next semester. And those sounded very intriguing and interesting...the theories behind it and how to apply this to society."

Snap!: The choice to use a block-based language (EQ3) was a risky one. Some students explicitly complained that they did not develop "real programming skills" as exemplified by the first quote below (who clearly had prior programming experience). The second quote provides a more nuanced view, that the course led to learning some "practical things" and "general computer science understanding." The third and fourth quotes are from students who understand that Snap! was about learning the "principles"

- "I felt like the assignments we did have in Snap didn't really target programming skills....But it was rather just putting stuff together, which is not the core of programming to me from my experience."
- "If I'm entirely honest, I thought that it was gonna be quite simplistic in the beginning almost to a point where I was like, this is a little demeaning. But now I feel like I know a lot more practical things and I have not just like, of course understanding what Snap is itself but like, just general computer science understanding...That I feel like is much more in depth than I expected at least."
- "Snap has made, in terms of learning curve seems to be a pretty easy learning curve...Since the projects are so open ended, a lot of the times I'll just go off on a bizarre concept and try to make it. "
- "Snap feels like learning the principles and then executing that doesn't seem like such a far stretch anymore. Now I'm in a much more capable position than I was before. I wouldn't necessarily do it in Snap, but I feel like I could at least envision sort of what I want to do and then take it further."

How difficult the courses were: We were surprised that students gave advice to future students (on the institutional survey) about the difficult of the courses (EQ6), like "Prepare for homework being slightly harder than expected, especially if you're not a computing major." We also saw students saying that the courses *could* have been overwhelming: "This is an amazing way to dip your toes into a usually overwhelming subject. No matter your experience or personal learning curve, you will be able to produce something to be proud of."

5 Conclusion

In this paper, we have described a process for defining courses based on how liberal arts and sciences faculty used and explored computing, as opposed to creating a course for non-majors that would cover the introductory computer science concepts typically included in a CS0 or CS1 course. In this paper, we focus on courses that meet two of the themes, Computing for Expression and Computing for Justice. We used a participatory design process for defining the standards for these courses, and used a highly-scaffolded process when introducing programming to these students. We evaluated the course designs to raise the possibility that our approach was not working, asking if the students found the courses to be useful or to be unhelpful, and if they saw future value in the course material.

We are optimistic about these formative results. The students who participated in our evaluation did find the courses useful, even if their utility was not developing in-depth programming skills. They felt that they developed their ability to be conversational programmers, and that the course ebooks set them up for future learning. They saw how the courses would be helpful to them later, even if they did not plan to enroll in computer science courses in the future. The use of Snap! was controversial among students, but our sense from the evaluation is that it worked. Students reported learning "general" skills that they might develop later. The course was not *too* easy, though perhaps their concerns about the courses possibly being "overwhelming" are based on prior expectations of computing rather than the course itself.

Since we conducted these evaluations, enrollment in the courses has more than doubled. The new program has hired a lecturer to keep up with demand. We have important future questions to answer, like whether we are actually teaching what the liberal arts and sciences faculty want their students to learn. We designed the courses based on their input, but we have not yet solicited faculty review of the courses.

The contribution of this experience report is around the goals of computing within the liberal arts and sciences and a course design *process*. Our experience suggests that we can create useful courses that involve programming through a participatory design process with liberal arts and sciences faculty. Students are finding these courses useful even when they are not pathways into the CS major.

We see our experience as opening new directions for developing models of computing education. We hope that others can use similar processes for gathering input broadly from their LSA faculty, for involving LSA faculty as participants in a design process, and for incorporating non-traditional programming media (like blocks and teaspoon languages) to implementing courses that meet non-CS major needs. Despite the enormous growth of the CS major at many campuses [31], there are still more *non*-CS majors in higher education. Creating forms of computing education which serve the needs of non-CS majors helps to achieve the goal of computing education for all.

References

[1] Michael Ball, Daniel Garcia, Victoria Phelps, and Yuan Garcia. 2024. Snap! 9 - Support for Teachers and Programming with Data. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2* (Portland, OR, USA) *(SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 1924. https://doi.org/10.1145/3626253.3635437

[2] Anna Batra, Iris Zhou, Suh Young Choi, Chongjiu Gao, Yanbing Xiao, Sonia Fereidooni, and Kevin Lin. 2024. "It Can Relate to Real Lives": Attitudes and Expectations in Justice-centered Data Structures & Algorithms for Non-Majors. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 88–94. https://doi.org/10.1145/3626252.3630754

[3] Brett A Becker and Keith Quille. 2019. 50 years of cs1 at sigcse: A review of the evolution of introductory programming education research. In *Proceedings of the 50th acm technical symposium on computer science education*. 338–344.

[4] Virginia Braun, Victoria Clarke, Nikki Hayfield, Louise Davey, and Elizabeth Jenkinson. 2022. Doing Reflexive Thematic Analysis. In *Supporting Research in Counselling and Psychotherapy : Qualitative, Quantitative, and Mixed Methods Research*, Sofie Bager-Charleson and Alistair McBeath (Eds.). Springer International Publishing, Cham, 19–38. https://doi.org/10.1007/978-3-031-13942-0_2

[5] Virginia Braun, Victoria Clarke, Nikki Hayfield, and Gareth Terry. 2019. Thematic Analysis. In *Handbook of Research Methods in Health Social Sciences*, Pranee Liamputtong (Ed.). Springer, Singapore, 843–860. https://doi.org/10.1007/978-981-10-5251-4_103

[6] Parmit K Chilana, Celena Alcock, Shruti Dembla, Anson Ho, Ada Hurst, Brett Armstrong, and Philip J Guo. 2015. Perceptions of non-CS majors in intro programming: The rise of the conversational programmer. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 251–259.

[7] Parmit K Chilana, Rishabh Singh, and Philip J Guo. 2016. Understanding conversational programmers: A perspective from the software industry. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. 1462–1472.

[8] Amanda Coffey and Paul Atkinson. 1996. Concepts and Coding. In *Making sense of qualitative data: Complementary research strategies*. Sage Publications, Inc.

[9] Kathryn Cunningham, Barbara J. Ericson, Rahul Agrawal Bejarano, and Mark Guzdial. 2021. *Avoiding the Turing Tarpit: Learning Conversational Programming by Starting from Code's Purpose*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3411764.3445571

[10] Jessica Q. Dawson, Meghan Allen, Alice Campbell, and Anasazi Valair. 2018. Designing an Introductory Programming Course to Improve Non-Majors' Experiences. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18)*. Association for Computing Machinery, New York, NY, USA, 26–31. https://doi.org/10.1145/3159450.3159548

[11] Betsy DiSalvo, Jason Yip, Elizabeth Bonsignore, and Carl DiSalvo. 2017. *Participatory design for learning: Perspectives from practice and research*. Taylor & Francis.

[12] Zachary Dodds, Malia Morgan, Lindsay Popowski, Henry Coxe, Caroline Coxe, Kewei Zhou, Eliot Bush, and Ran Libeskind-Hadas. 2021. A Biology-based CS1: Results and Reflections, Ten Years In. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (Virtual Event, USA) *(SIGCSE '21)*. Association for Computing Machinery, New York, NY, USA, 796–801. https://doi.org/10.1145/3408877.3432469

[13] Barbara J. Ericson, Kantwon Rogers, Miranda Parker, Briana Morrison, and Mark Guzdial. 2016. Identifying Design Principles for CS Teacher Ebooks Through Design-Based Research. In *Proceedings of the 2016 ACM Conference on International Computing Education Research* (Melbourne, VIC, Australia) *(ICER '16)*. ACM, New York, NY, USA, 191–200. https://doi.org/10.1145/2960310.2960335

[14] Samuel B Fee, Amanda M Holland-Minkley, and Thomas E Lombardi. 2017. *New directions for computing education: Embedding computing across disciplines*. Springer.

[15] W. Finzer and D. Damelin. 2016. Design perspective on the Common Online Data Analysis Platform (CODAP). In *Paper presented at American Educational Research Association (AERA) conference*. Washington DC.

[16] Jane Friedhoff. 2013. Untangling twine: A platform study. In *Proceedings of DiGRA 2013 Conference*.

[17] Michelle Friend, Andrew W. Swift, Betty Love, and Victor Winter. 2023. A Wolf in Lamb's Clothing: Computer Science in a Mathematics Course. 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, 256–262. https://doi.org/10.1145/3545945.3569736

[18] Mark Guzdial. 2022. Providing Students with Computational Literacy for Learning About Everything. In *Computational Thinking Education in K-12: Artificial Intelligence Literacy and Physical Computing*, Siu-Cheung Kong and Hal Abelson (Eds.). MIT Press.

[19] Mark Guzdial. 2023. Scaffolding to Support Humanities Students Programming in a Human Language Context. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 2* (Turku, Finland) *(ITiCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 579–580. https://doi.org/10.1145/3587103.3594157

[20] Mark Guzdial. 2023. Scaffolding to Support Liberal Arts Students Learning to Program on Photographs. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 2* (Turku, Finland) *(ITiCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 577–578. https://doi.org/10.1145/3587103.3594158

[21] Mark Guzdial, Emma Dodoo, Bahare Naimpour, Tamara Nelson-Fromm, and Aadarsh Padiyath. 2023. Putting a Teaspoon of Programming into Other Subjects. *Commun. ACM* 66, 5 (apr 2023), 24–26. https://doi.org/10.1145/3587926

[22] Mark Guzdial and August Evrard. 2023. Identifying the Computing Education Needs of Liberal Arts and Sciences Students (Discussion Paper). In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*. 1–7.

[23] Mark Guzdial and Andrea Forte. 2005. Design process for a non-majors computing course. *ACM SIGCSE Bulletin* 37, 1 (Feb. 2005), 361–365. https://doi.org/10.1145/1047124.1047468

[24] Mark Guzdial and Tamara Nelson-Fromm. 2024. A purpose-first theory of transfer through programming in a general education computing course. In *Proceedings of the PLATEAU Workshop*. https://doi.org/10.1184/R1/25587501.v1

[25] Amruth N Kumar and Rajendra K Raj. 2022. Computer Science Curricula 2023 (CS2023) Community Engagement by the ACM/IEEE-CS/AAAI Joint Task Force. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2*. 1212–1213.

[26] Cynthia Bailey Lee. 2013. Experience report: CS1 in MATLAB for non-majors, with media computation and peer instruction. In *Proceeding of the 44th ACM technical symposium on Computer science education (SIGCSE '13)*. Association for Computing Machinery, New York, NY, USA, 35–40. https://doi.org/10.1145/2445196.2445214

[27] Golan Levin. 2009. Art that looks back at you. https://www.ted.com/talks/golan_levin_art_that_looks_back_at_you.

[28] Yinmiao Li, Melissa Chen, Ayse Hunt, Haoqi Zhang, and Eleanor O'Rourke. 2024. Exploring the Interplay of Metacognition, Affect, and Behaviors in an Introductory Computer Science Course for Non-Majors. In *Proceedings of the International Computing Education Research Conference 2024*.

[29] Albert Lionelle, Josette Grinslad, and J Ross Beveridge. 2020. CS 0: Culture and coding. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 227–233.

[30] Michael Mateas. 2005. Procedural literacy: educating the new media practitioner. *On the horizon* 13, 2 (2005), 101–111.

[31] National Academies of Sciences, Engineering, and Medicine. 2018. *Assessing and Responding to the Growth of Computer Science Undergraduate Enrollments*. The National Academies Press, Washington, DC. https://doi.org/10.17226/24926

[32] Safiya Umoja Noble. 2018. Algorithms of oppression: How search engines reinforce racism. In *Algorithms of oppression*. New York university press.

[33] Manuel Pérez-Quiñones. 2021. Why I Want my Voice Assistant to Speak Spanglish. https://youtu.be/ohXfjysQhx8?si=50k0_1crK5A3CnMI.

[34] Alan J. Perlis. 1962. The Computer in the University. In *Computers and the World of the Future*, Martin Greenberger (Ed.). MIT Press.

[35] Jessica Rizzo. 2022. Generative Art Is Challenging What It Means to Be Human. https://www.wired.com/story/generative-art-intellectual-property-law/.

[36] Bianca L. Santana and Roberto A. Bittencourt. 2018. Increasing Motivation of CS1 Non-Majors through an Approach Contextualized by Games and Media. In *2018 IEEE Frontiers in Education Conference (FIE)*. 1–9. https://doi.org/10.1109/FIE.2018.8659011

[37] B. Skudder and A. Luxton-Reilly. 2014. Worked Examples in Computer Science. In *Proc. Sixteenth Australasian Computing Education Conference (ACE2014), Auckland, New Zealand*, J. Whalley and D. D'Souza (Eds.). CRPIT, 59–64.

[38] Sónia Rolland Sobral. 2020. CS1 and CS2 curriculum recommendations: learning from the past to try not to rediscover the wheel again. In *Trends and Innovations in Information Systems and Technologies: Volume 3 8*. Springer, 182–191.

[39] Mike Tissenbaum, David Weintrop, Nathan Holbert, and Tamara Clegg. 2021. The case for alternative endpoints in computing education. *British Journal of Educational Technology* 52, 3 (2021), 1164–1177.

[40] Jacoba Urist. 2015. From Paint to Pixels. https://www.theatlantic.com/entertainment/archive/2015/05/the-rise-of-the-data-artist/392399/.

[41] April Y Wang, Ryan Mitts, Philip J Guo, and Parmit K Chilana. 2018. Mismatch of expectations: How modern learning resources fail conversational programmers. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–13.

[42] Austin L. Zuckerman, Lily Vo, and Ashley L. Juavinett. 2024. Student Motivations and Expectations for an Introductory Programming Course in Biology. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1* (Portland, OR, USA) *(SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 1498–1504. https://doi.org/10.1145/3626252.3630931