Slowness, Politics, and Joy: Values That Guide Technology Choices in Creative Coding Classrooms

Andrew McNutt
University of Utah
Salt Lake City, UT, USA
andrew.mcnutt@utah.edu

Sam Cohen
University of Chicago
Chicago, IL, USA
samcohen@uchicago.edu

Ravi Chugh
University of Chicago
Chicago, IL, USA
rchugh@cs.uchicago.edu

Abstract

There are many tools and technologies for making art with code, each embodying distinct values and affordances. Within this landscape, creative coding educators must evaluate how different tools map onto their own principles and examine the potential impacts of those choices on students' learning and artistic development. Understanding the values guiding these decisions is critical, as they reflect insights about these contexts, communities, and pedagogies. We explore these values through semi-structured interviews with (N=12) creative coding educators and toolbuilders. We identify three major themes: slowness (how friction can make room for reflection), politics (including the lasting effects of particular technologies), and joy (or the capacity for playful engagement). The lessons and priorities voiced by our participants offer valuable, transferable perspectives—like preferring community building (such as through documentation) over techno-solutionism. We demonstrate application of these critical lenses to two tool design areas (accessibility and AI assistance).

CCS Concepts

• **Human-centered computing** → *Human computer interaction (HCI)*; • **Applied computing** → *Media arts*; *Education*; • **Social and professional topics**;

Keywords

Creative Coding, Interview Study, Power, Reflection, Arts

ACM Reference Format:
Andrew McNutt, Sam Cohen, and Ravi Chugh. 2025. Slowness, Politics, and Joy: Values That Guide Technology Choices in Creative Coding Classrooms. In *CHI Conference on Human Factors in Computing Systems (CHI '25), April 26-May 1, 2025, Yokohama, Japan.* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3706598.3713472

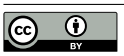
1 Introduction

Creative coding is a mode of computer-based work "that emphasizes the expressivity of computer programming beyond something pragmatic and functional" [54], in a manner usually associated with the production of art [24, 49, 89]. This setting has led to a huge number and variety of artistic works across numerous disciplines including visual art [54], music [35], Twitter bots [14], dance [71], Internet of Things devices [101], digital fabrication [94], and more. These artworks are mediated and shaped by a wide range of creative coding–specific tools, including Processing [80] (and its JavaScript descendent p5.js [66]), openFrameworks [64], Pure Data [77], among many others [22, 62]. These technologies often include custom editors, such as the p5 editor for p5.js, which provide affordances relevant to the domain. Though niche, this area is not obscure: creative coding platforms like OpenProcessing [65] have tens of thousands of users [95].

Beyond being a useful means of creating art, creative coding has been widely used as a context in which to teach computational thinking concepts [27, 55, 73, 107]. The immediacy and tangible nature of creative work provides straightforward incentives for novices to engage with the ideas of computation, compared to those taught in traditional introductory CS courses. Creative coding educators must select from a wide universe of tools, often developing tools to solve problems they have in the classroom or to give themselves greater expressivity in their artworks (which are often cycled back into the classroom). These custom tools often include standalone editors adapted to particularities of the domain (such as danceON [71] for choreography) or libraries that enhance an aspect of the domain (such as p5.sound [76] for making music).

While creative coding practices have been studied from the perspectives of artists [61, 79, 93, 100] and students [60], there has been little [10] consideration of how *educators* select and design tools for creative coding classrooms. Which tools are valued and why? What can be gleaned about this domain, the tool ecosystem, and the people working in this area from these preferences? Educators, in this context, necessarily take on the roles of guide and practitioner, making them keen observers of the hard edges tools have and how they effect students. Moreover, answers to these questions are likely to be of value beyond the classroom, as they reflect insights about art-centered tools and their surrounding communities which are not captured by individual interactions.

We explore these questions via a semi-structured interview study with practitioners (N=12) who have taught creative coding and have built tools, both broadly defined. We find that the reasoning for technology selection and usage varies widely, but that the concerns expressed in those considerations broadly center around three themes. First is SLOWNESS, referring to issues around the creation of friction-ful experiences meant to convey pedagogically important experiences or systems meant to a support directorial style of agency (i.e. forgoing craft in favor of artistic decision-making). Next is POLITICS, regarding the context of the classroom, its relationship to structures governing control and access to software, and the power of educators over their students. Finally, there is JOY, capturing playfulness and connecting with communities or practices. These themes are interwoven; for instance, the types



CHI '25, Yokohama, Japan

ACM ISBN 979-8-4007-1394-1/25/04
https://doi.org/10.1145/3706598.3713472

of JOY (and associated learning) that are accessible through SLOW tedious interactions with some systems may not be available in "fast" systems that do everything for you. We explore how these values play out in two application areas: the role of accessibility and AI in creative coding classrooms.

Beyond being a compelling community whose needs and interests are worthwhile to understand, we suggest that looking to the educators of creative coding is of particular value for HCI. These communities are often excluded or overlooked, leaving many without the longstanding collaborators and investment that other areas have enjoyed (e.g. creativity support tools). These communities have built their own tools, defined their own effective practices, taught thousands upon thousands of students, and produced myriad artworks. The lessons they have learned are valuable, such as the value of fostering community, building inclusive documentation, making some experiences friction-ful, and centering kindness over techno-solutionism. This work highlights these lessons and suggests that they might be useful in other areas of HCI.

2 Related Work

Creative coding possesses unique practices and practitioners. Here we review its practice as well as its use in the classroom, but first we situate our studied notion of creative coding among related fields.

2.1 Creativity, Coding, and Creative Coding

Creative coding is a broadly porous term, encapsulating many activities, tools, and techniques. Some definitions of the term identify broader qualities of such work and its creators. For instance, Verano Merino and Sáenz [100] find that creative coding is typified by uncertainty and an absence of requirements (in contrast to other types of programming), allowing it to incorporate a wide range of art forms and types. Vestergaard et al. [101] defined people who work in this area as being digital artists who are self-taught programmers. For our work, the single most suitable association of the term traces back to the formation in 2001 of Processing [80], which is commonly associated with the term. However, creative coding is not a monolith and can encapsulate many related areas such as CSS art [69] and the demo scene [84].

Closely related is the term "media computation" [27, 29], which captures a related set of qualities as creative coding (intentionally drawing a parallel with new media art). Although the boundaries are fuzzy, media computation is generally associated solely with pedagogical goals, whereas creative coding also has a predominant practical emphasis. Like most computer science–based creative coding courses [31], the underlying perspective of media computation curricula might be summarized as teaching computer science through motivating art applications. Computer science education researchers have studied potential benefits—regarding gender diversity, retention, class performance, and so on—of emphasizing computing with media in introductory programming courses [28, 29, 83, 91]. Centered in the classroom, our concerns are related to these but also draw on the wider field of associated artistic creative coding practice.

Also nearby are creativity support tools (CSTs), which support creative tasks through software applications [90]. For example, the commercial system Adobe Photoshop is a canonical CST for creating images. Research in this area is sprawling: a recent survey of CSTs taxonomized 143 HCI papers [21], while another analyzes 111 CSTs centered on art-making [11]. Creative coding also supports a particular flavor of creative tasks, but restricted to those that specifically involve coding—which only some CSTs encompass.

2.2 Creative Coding in Practice

Next, we consider works on creative coding in the context of practical art-making settings.

Artists. At the center of any art making practice are the artists making the work. Prior work has found a variety of different relationships between these practitioners and technologies.

Li et al. [51] interviewed 13 professional artists who use code in their work, identifying themes relating to motivation, choice, and process. Relevant to our work, many of these artists valued efficiency, for example, to rapidly explore different compositions (something our participants also valued), but manual control was preferred over automation for achieving aesthetic outcomes (related to mismatches between tool designers and users). Furthermore, many artists were motivated to make their own tools for reasons like a desire to align tooling with unique artistic styles and to support intellectual growth. Verano Merino and Sáenz [100] interviewed 5 artists who produce work with code, characterizing their working practices and highlighting a heterogeneity of approaches to tools, with some moving from simpler tools (e.g. p5.js) to more advanced or unconventional development environments.

Closely related are Li et al.'s [52] considerations of the role of power in CSTs. They highlight the ways in which tools—and their creators—express power over their users. For example, a CST may be designed to reduce tasks which the creators deem to be tedious, but which users might actually find to be useful steps toward their artistic goals. Such power dynamics are investigated by interviewing 11 artists (i.e. users of CSTs) and toolmakers who design CSTs. Vasudevan [99] interviewed 53 artists, curators, and administrators in the closely affiliated area of new media art—which breaks from creative coding by often, but not always, including code as a part. She highlights the burdens put upon artists by the tech industry (which will sometimes use them as a form of abstracted research and development department), while at the same time having their work forced into ephemerality due to changes from the industry (such as to APIs or platforms) beyond their control. Dovetailing with these works, we study creative coding technologies used in classrooms (e.g. as compared to CSTs used for art) by interviewing educators and toolmakers who design and use these tools.

Tools and Techniques. Several studies have focused more narrowly on individual tools or tool components.

For example, several works have considered adapting the idea of version control from software engineering to creative practices. Verano Merino and Sáenz's [100] participants observed that the exploratory nature of their work is not well supported by existing version control systems. Sterman et al. [93] interviewed 18 creative professionals (including several creative coders) about their practices around version control. Some highlighted that high-resolution git-style version control can sometimes impede JOYful creative expression by causing new work to overfit to older work. Rawn et al.

[79] find that code-based artists treat code much as how materials like clay are treated; this informs the design of a specialized version control system for Processing. Subbaraman et al. [95] study how creative coders interact with remixing (an abstracted form of version control), finding that reuse and restructuring are common. We do not dive deeply into a single technological facet as these works do, but instead focus on identifying high-level issues that might be considered in the context of any area of creative coding practice.

Technological interventions have also been designed to break free from conventional, text-based programming interfaces. Dynamic Brushes [38] allows custom brushes to be programmed (in a visual programming environment) in a way that retains stylistic elements from the artist's hand-drawn strokes (through a digital stylus). Stamper [8] and Field [19] provide nodes and wires for connecting (textual) code fragments on a two-dimensional canvas in a manner which may help support non-linear art-making.

Artists often act as toolmakers to support their work, and toolmakers pursue technological interventions they hope artists will find useful. As summarized in a 2018 report titled *Open Source Software Toolkits for the Arts (OSSTA)* [57], contributors, maintainers, and authors of such toolkits identified that social factors—such as inclusivity, funding, community, and leadership—were primary barriers to carrying out their work. Technical needs (e.g. software maintenance and documentation) were also areas of concern. Our concerns are related to OSSTA's but situated in the classroom.

2.3 Creative Coding in the Classroom

In differentiating media computation from creative coding, we suggested differences in *student* populations compared to creative coding courses. In parallel, the communities of people involved in *teaching* and *toolmaking* to support creative coding and media computation courses are mostly disjoint: the former appear largely in "the art world" whereas the latter is largely in "the computer science education world." As such, it is valuable to study the structure of creative coding courses, and the people involved with them, separately from students of intro programming in media computation or other more well-studied settings.

Curricula. Creative coding is a common means for teaching about programming and computation (e.g. [25, 55, 60, 97]). Hansen [31] surveys the curricula of 30 creative coding courses, finding that they predominantly centered learning to code in the context of artistic prompts, rather than learning to do art with code. In contrast, many of our participants did their teaching in more art-centered contexts. This type of course can provide an accessible introduction to computational thinking for students who might not be otherwise interested. Greenberg et al. [25] suggest that such introductions can be more appealing to those often left out by traditional CS curricula (such as women) by creating a more inviting environment. Participants in Chung and Guo's [10] study espouse similar positions, highlighting the value of new media art-centered introductions to computing for women, non-binary, and queer people.

Students. Naturally, creative coding technologies may be experienced differently by professional artists (in practical contexts) than by students (whether in a classroom context or not). Mitchell and Bown [61] performed a lab-based study with 9 creative coders, of varying levels of expertise, performing a task in Processing. They identified opportunities for tools that highlight and visualize program state, that "minimise the creative feedback gap" by shortening iteration cycles, and assist exploration. Through several offerings of a creative coding course with college and high-school students, McNutt et al. [60] probe student perceptions of several features in a slightly modified p5 editor, including custom features for manipulating colors and shapes via GUI interactions (rather than coding). Their students offered mixed responses regarding the iteration-shortening auto-refresh feature, as well as skepticism about tools doing too much work for them, allowing students to "skip the sweat" that leads to learning.

Educators. In addition to the students in creative coding classrooms, it is valuable to study the teachers of those classrooms. These educators are often toolmakers and artists, too. Such studies, thus, contribute to the growing body of work studying artists in relation to programming tools and practices. Levin and Brain [49] interviewed 16 creative coding educators to understand the types of assignments and practices that they draw on in their teaching, while Chung and Guo [10] interviewed 18 new media artists who teach workshops on computing topics about their motivations and practices. Our work is centered around the specific issue of tool selection and design, rather than on the creative coding classroom as a whole and the practical work that goes on there. That said, on subjects where our interviews overlapped, our findings broadly comport with these prior studies. For instance, Levin and Brain's [49] participants noted that their classrooms were often bimodal, consisting of novices and experts, as well as students skilled with art and not coding and vice versa. Similarly, Chung and Guo [10] observed that educators were often motivated to teach to make new culture and critically interrogate the world around them.

3 Interview Study

To understand the desires, beliefs, and expectations that educators in creative coding classrooms have for the technologies they use, we conducted an interview study with practicing educators.

The motivating goal of this study was to answer questions such as: *What are the barriers creative coding educators face, and how might they be resolved through technological or tool-level interventions?* Our interview guide thus included questions about languages and tools, the role of AI, live coding, course projects, and so on.

However, as we conducted the interviews, participants spoke at greater length about the *values* they brought to the consideration of new technologies and the relationships they had with those and surrounding tools, and less about the *technologies* themselves (although that did come up as well). These values fundamentally inform their technology selection and more broadly their classroom design. Given this stronger signal, we refocused subsequent interviews and our analysis of them around these more foundational perspectives—using questions about specific technologies as a platform to discuss the principles in play.

Below, we outline our study methodology, give an overview of who participated in our interview (to situate their perspectives in the context of our work), describe how we analyzed the interviews, characterize the limitations of our study design, and offer a positionality statement.

Id	Name	Most Strongly Aligned Labels			Self-Specified Labels	Example Tool Built
$\mathbf{P}_{\text{Willie}}$	Willie Payne		Educator	Toolbuilder		danceON [71]
$\mathbf{P}_{\text{Nick}}$	Nick Briz	Artist	Educator	Toolbuilder	Organizer	netnet [63]
$\mathbf{P}_{\text{GL}}$	*Pseudonymized*	Artist	Educator	Toolbuilder	Organizer	
$\mathbf{P}_{\text{Matt}}$	Matt DesLauriers	Artist	Educator	Toolbuilder	Creative Coder	canvas-sketch [17]
$\mathbf{P}_{\text{A}}$	*Anonymized*	Artist	Educator	Toolbuilder	Researcher	
$\mathbf{P}_{\text{B}}$	*Anonymized*	Artist	Educator	Toolbuilder		
$\mathbf{P}_{\text{Allison}}$	Allison Parrish	Artist	Educator	Toolbuilder	Poet	pytracery [70]
$\mathbf{P}_{\text{Baku}}$	Baku Hashimoto	Artist		Toolbuilder	Video Director, Graphic Designer	Glisp [32]
$\mathbf{P}_{\text{Cassie}}$	Cassie Tarakajian	Artist	Educator	Toolbuilder		p5 editor [67]
$\mathbf{P}_{\text{C}}$	*Anonymized*			Toolbuilder		
$\mathbf{P}_{\text{Tega}}$	Tega Brain	Artist	Educator			p5.riso [48]
$\mathbf{P}_{\text{Chris}}$	Chris Coleman	Artist	Educator	Toolbuilder	Arts Engineer, Technical Producer	Maxuino [12]

Figure 1: The background of participants in our study. Participants were asked how they would like to be referred to (anonymously, by name, by selected name)—for instance $\mathbf{P}_{\text{GL}}$ specifically requested to be presented that way. We also asked which of the labels, "artist, teacher, and toolbuilder," (relabeled here as educator) they identified with—gray labels indicate positive alignment while black labels indicate strong alignment. Many participants contributed to tools following the pluralistic notion of contribution espoused throughout: such as by contributing documentation or learning materials, which does not so squarely fit in a contribution column but we emphasize here. Participants are presented in the order we interviewed them.

3.1 Interview Methodology

The target user population for our study consists of creative coding educators, particularly those who also develop their own tools to support their pedagogical or artistic work. Studying educators who have "only" taught using existing tools would be valuable for assessing educational practices. However, we were specifically interested in the intersectional perspectives of artists, educators, and toolmakers.

In total, we contacted 33 potential participants via email. Participants were identified through a convenience sample guided by prominence on social media, participation in related studies [49], and referrals from other participants who completed our study (i.e. snowball sampling). To qualify, participants must have taught creative coding in some way, and also made, modified, or contributed to the tools they used for doing that teaching. Following the pluralistic definition of contributorship espoused by communities in this ecosystem (particularly in the Processing family [68]), our definition of "teaching" is broad, including classroom-based teaching, workshops, mentorship, and tutorial writing.

The study protocol—reviewed by a university institutional review board—included a semi-structured interview followed by a brief follow-up survey. Interviews were conducted over Zoom during Summer 2023 and lasted approximately one hour each. Interviews were recorded, automatically transcribed, and then reviewed for accuracy by the first author. Participants received $75 USD for their participation as an online gift certificate.

In the follow-up survey, participants were asked whether they would like to be explicitly named or remain anonymous; their responses inform our disclosure of identities throughout this work. We chose this approach to explicitly credit the thoughts of these artists and educators. A draft was sent to participants prior to submission for review and quote approval. The study instruments are in the appendix.

3.2 Demographics and Backgrounds

We interviewed (N=12) professionals broadly in the space of creative coding pedagogy and practice. The participants, referred to as $\mathbf{P}_{\text{name}}$ and quoted "like so", are summarized in Fig. 1. We now briefly review their backgrounds.

Two participants used *they/them* pronouns, three used *she/her*, and seven used *he/him*. Their educational backgrounds ranged from some college-level study to completed masters (6/12) or Ph.D.s (2/12). All but two are US-based. Many participants (7/12) have been practicing educators for more than six years.

Nearly all participants (10/12) have taught or actively teach courses at the university level, although only 7/12 participants have primary appointments as university faculty. These include courses for both graduate and undergraduate students. Among them, only one has an appointment in a computer science department (by courtesy). Instead, some participants teach in a range of departments including art, new media, cinema, design, and information sciences. Others freelance or have positions in software. Nearly all participants have taught workshops of various types. The length of these ranged from one day events to 15 week "internships", and covered various audiences including middle schoolers (e.g. $\mathbf{P}_{\text{Tega}}$) or the general public (e.g. $\mathbf{P}_{\text{B}}$). Teaching in non-traditional contexts or venues was common. For instance, in addition to teaching in university contexts, $\mathbf{P}_{\text{Willie}}$ taught at the Filomen M. D'Agostino Greenberg Music School, an alternative school focused on music education for people of all ages with vision loss (as documented in his FilOrk [72] work). $\mathbf{P}_{\text{Nick}}$ has taught at Marwen, an arts education non-profit focused on low income youth (high school and below). $\mathbf{P}_{\text{C}}$ has developed educational software. All participants have done additional forms of teaching, including production of documentation, creating tutorials, or mentoring.

3.3 Analysis Methodology

Interviews were analyzed via open theming [7]. The first author coded transcripts (identifying ~30 topical codes), and then developed themes, which were discussed with the project team. The code book is summarized in the appendix. We forgo quantitative analysis of, say, the frequency with which particular technological or curricular choices were made or discussed, because our sample of the population of creative-coding educators is small [102]. Our qualitative analysis demonstrates that our participants' opinions and perspectives are among the concerns surrounding such choices.

In forming our themes, we sought to identify salient connections between the many interrelated topics (i.e. codes) covered in the interviews, also reconciling these findings with existing studies of creative-coding artists [50–52, 79, 93, 95, 100], students [60], and educators [10, 49]. In addition to our themes, participants expressed a variety of practical concerns and difficulties, which were annotated with codes not directly mapped to any themes or topics in this paper. Some omitted codes correspond to topics well considered in prior work—for example, artists studied by Verano Merino and Sáenz [100] reflected on the "creative coding" terminology, and educators interviewed by Levin and Brain [49] discussed their classrooms being bifurcated between students skilled at either art or coding. Other omitted codes refer to specific technological concerns (such as the particularities of JavaScript) and ideas that, in our mind, did not easily synthesize into takeaway lessons or broader themes. As part of the interview process, the first author wrote memos summarizing the topics discussed (which the rest of the team read during the interviews, ~2 months), which informed the initial topic codes. Our key themes emerged towards the end of the interview process. These were held as thematic hypotheses until the full analysis was undertaken. Discussions were held slowly asynchronously (~8 months) and one synchronous discussion during which the thematic hypotheses were mulled. The first author then coded the data, iteratively grouping the codes into different possible themes and topics. These were iterated on (for ~3 months) asynchronously (via email and writing) using evidence drawn from transcripts and then agreed upon in a final synchronous discussion.

3.4 Limitations

Like any study, ours has a variety of limitations. For instance, many of our participants were—at the time of their interview, or previously—affiliated with the Processing Foundation, which may have affected their opinions and perspectives. Many participants were also based in the United States.

This work is broadly framed within the context of HCI research—rather than as an education, art, or design project—which affects our perspective (for example, the recurring theme of how tool-based interventions can be useful). Related studies might be conducted from other perspectives. Furthermore, we focused on creative coding as separate from creativity support tools or media computation (as outlined in Sec. 2). Although participants' perspectives about the term suggest that this distinction is justified, future work might reexamine similar questions from related user populations.

Furthermore, our focus on educators who have built their own tools may have biased our results. If we had instead focused on educators more generally (or only on those that have taught workshops, as Chung and Guo [10] do), we may have elicited a different set of views. For instance, adjunct professors teaching large courses may have less agency over tool selection than those with the control to build their own tools. Similarly, our results do not reflect the opinions or positions of all creative coders or creative coding educators. A more broad ranging study, such as a quantitative survey, might better capture a consensus of creative-coding related opinions.

3.5 Positionality

We are a group of US-based researchers situated in computer science with a focus on human-computer interaction. Each of us has taught creative coding (particularly as a means to introduce computing to non-majors or to high school students) and have researched creative computing or related support tools. Each of us identify as educators and tool builders, as well as novice artists or art enthusiasts to varying degrees. As noted, the original goal of this work was to use this study as need-finding for subsequent tool building. While our focus shifted over the course of the interviews to center the interrogation and consideration of sociotechnical values, this origin likely inflects many of our reflections. Further, our positions in US-based higher education lenses our focus. For instance, limiting our ability to evaluate the applicability of our findings to non-US-based contexts, in earlier stages of learning such as K-12 (save for our experience with high schoolers [60]), or in one-off education contexts such as workshops. Lastly, as researchers housed in CS we acknowledge that we possess a substantial power (often reified as funding) compared to our colleagues in areas such as the arts—a disparity we strive not to magnify via this research (see Sec. 7).

4 Technological Setting

Before characterizing the findings of our analysis, we consider the technical settings that participants spoke out about working within.

Participants described using a wide range of different tools and technologies in their teaching. These included tools for the production of visual art via text-based coding, such as Processing, p5.js, openFrameworks, and three.js. They also noted using tools for producing visuals through GUIs, such as through Touch Designer, Unity, nodes.io, and Quartz Composer. Tools for making music were also used, including Pure Data, MaxMSP, and Tidal Cycles [59]. $\mathbf{P}_{\text{Allison}}$ used Tracery [14] to create textual art. Less specialized coding environments are sometimes used to support creative coding. For instance, $\mathbf{P}_{\text{Matt}}$ occasionally used web-based IDEs like Glitch, instead of the p5.editor. Microcontrollers, such as Arduino, were also often used. Kelleher and Pausch [42] review programming environments and tools (in 2005) for novice programmers (in general, not merely in creative coding), dividing them into "teaching systems" and "empowering systems." Our participants did not express a comparable divide, instead viewing tools for both pedagogy and practice. This perspective may be related to our subjects largely being educators who are also toolmakers.

Participants made their own tools for various purposes, ranging from research, to artistic needs, to spite ($\mathbf{P}_{\text{GL}}$), to pedagogical needs—often forming a virtuous cycle in which technologies developed for art practice inform those developed for pedagogy and vice versa. As a research project, $\mathbf{P}_{\text{Willie}}$ made a tool called danceON [71],

Theme	Meaning
SLOWNESS	Tools meant for art-making and pedagogy walk a difficult line. If too slow and friction-ful, then they will not be useful; if too fast and seamless, then students may only be able to express directorial agency rather than learned craft.
POLITICS	Selecting classroom technologies can entwine students and educators in complex political dynamics. While there is no easy solution, exploring tools that center community, control, cost, and (educator-student) connection seem to be valuable.
JOY	Finding joy in creative coding means not only using tools that are fun, but also helping students hone their technical and creative practices.

Figure 2: Synopsis of themes.

which consists of a domain-specific language for augmenting choreography through computer vision. $\mathbf{P}_{\text{Nick}}$ made a web-based editor called netnet [63] that provides interactive tutor style [58] hints and tutorials to guide users through the production of web pages and net art as a pedagogical tool. $\mathbf{P}_{\text{Chris}}$ made Maxuino [12] to provide GUI access to sensors without having to master Arduino and MaxMSP programming for his own practice and pedagogy. See Fig. 1 for additional example projects, attenuated by anonymity.

5 Themes

We identified three central forms of values that educators consider when making pedagogical and technological choices in creative coding classrooms: SLOWNESS, POLITICS, and JOY. We highlight findings from these themes in Fig. 2

5.1 Slowness: On Interface Speed

There is a natural tendency in designing interfaces to try to make them as fast as possible, to rapidly, and seamlessly [36], automate tasks that are not essential facets of the task at hand. Yet, this speed can lead users to race past useful experiences, particularly ones that are artistically or pedagogically helpful.

5.1.1 Slowing Things Down. Reducing the rate at which different tasks can be performed gives space for both artistry and learning, giving users time for reflection and personal growth.

Reflection. SLOWNESS seemed to have value for the production of and critical engagement with art.

$\mathbf{P}_{\text{Tega}}$ suggested that "in the arts, there's a real value to slowing down and taking the hood off things. Because it lets you ask critical questions, [such as] if you're truly engaging with it as a medium". $\mathbf{P}_{\text{Baku}}$ argued for the value of integrating tedium into his workflow, noting that it is useful to "integrate procedural ways of thinking with more manual or repetitive or more tedious works". He went on to describe how a photographer friend intentionally used an older and slower computer to guide the type of works he could create. Emphasizing the importance of user agency in this context, he summarized: "I always want the computer to be slower than my brain". $\mathbf{P}_{\text{C}}$ stressed that UIs and tools that try to speed up development processes (such as autocomplete) often impeded that agency by being distracting, impeding the flow of thinking.

Vasudevan [99] observe that new media artists often specifically desire opportunities to slow the pace of their work in contrast to the "move fast and break things" attitude sometimes thrust upon them by tools and external pressures. Li et al. [52] observe that automation of (what are externally perceived as) tedious processes can forgo artistically useful experiences. Similarly, Hullman et al. [34] observe that some friction-ful design choices (in the context of visualization) can cause a viewer to slow down and pay attention in a manner they might not otherwise. These sentiments echo the substantial literature on slow technology [30] and its benefits.

Learning Opportunities. Participants stressed that making an interface seamless can preclude opportunities for students to learn how to do things. $\mathbf{P}_{\text{Tega}}$ explained that a "seamless interface has taken away all that friction which produces understanding". Growth that accompanies learning can be uncomfortable [44], as it requires restructuring and reshaping of previous mental models, and so it can be usefully educational to allow some friction-ful interactions. While integrating elements (like documentation) can make a given task easier, creating some friction to slow that task seems to create opportunities for learning—a tension known to be present in some documentation forms [108]. $\mathbf{P}_{\text{Willie}}$ elaborated that "we saw this really clear distinction between the participants that were really comfortable with coordinate systems and some of the basic math in 2D compared with the students who didn't have that comfort level". Technological interventions, such as on-screen rulers or direct manipulation–based code generation [60], may help those students to produce artworks during the course. However, that may prevent them from engaging with the underlying ideas in a way that promotes long-term learning. McNutt et al. [60] echo this position, observing that students sometimes find such interventions deleterious for their learning.

5.1.2 Speeding Things Up. Reciprocally, there can be value in making things move quickly. Participants noted that faster interfaces can help automate the boring parts unrelated to the art and enable rapid design space exploration.

Directing versus Coding. A key tension in many creative coding classrooms is their function as both an environment for learning to code and one for learning to practice art [31]. In the former an essential value is to impart the basics of computation (such as for loops, functions, and abstractions), whereas in the latter those mechanics are less important than the art work produced.

To this end, $\mathbf{P}_{\text{GL}}$ noted that in his courses, which were on the art side, they "almost never, like 99% of the time" look at code, because focusing on such mechanics "reinforces a technocentric perspective that distracts from a holistic view of the art". He continued: "It'll be like [in] a painting class asking, 'How did you make that paintbrush make that paint mark?'", noting that such questions are "incredibly blinkered". $\mathbf{P}_{\text{GL}}$ noted that he expects coding skills to improve "experientially, through daily practice, rather than through explicit instruction." This aligns with how, in other participant's courses, students were guided toward a directorial style of agency: it does

not matter how you do it as long as it gets done and it serves the art. $\mathbf{P}_{\text{Nick}}$ noted that "in that directorial kind of approach, students oftentimes have ideas and they want to develop these ideas" rather than focus on the craft of coding. This orientation of priorities shifts the focus from the mechanics of coding to the artistry of the work. While this can be beneficial, this can also lead to unrealistic expectations of how quickly students will gain skills as Chung and Guo [10] observe.

While beneficial for artistry, relentless pursuit of speed in the interest of directorial agency can be problematic: it may force students to become dependent on others to execute their artworks. $\mathbf{P}_{\text{Allison}}$ noted that if someone exhibits an art work in a public space, "it's going to break and they're going to have to hire somebody who actually knows what they're doing to fix their project, and that's not giving people the skills and the autonomy and the understanding they need to be successful in the field". $\mathbf{P}_{\text{Nick}}$ suggested that students should not be excused from needing to learn to code on their own, commenting that "your ability to communicate with collaborators that are writing that code for you is going to be very different" if you are unable to code yourself.

Rapid Exploration. A related concern was enabling smooth and rapid exploration of the design space. $\mathbf{P}_{\text{Matt}}$ summarized: "as a generative artist, it's kind of essential...to have a tighter and tighter feedback loop". $\mathbf{P}_{\text{Baku}}$ echoed this perspective, noting that liveness (generally referring to programs that can be modified and then immediately, or continuously, executed [96]) can be useful for "tweaking in a parameter or magic number of source code to make outputs to be more aesthetic".

To help bring such user interfaces into the world, $\mathbf{P}_{\text{Matt}}$, $\mathbf{P}_{\text{Baku}}$, and $\mathbf{P}_{\text{Willie}}$ (and others) had all developed tools (including canvas-sketch [17], Glisp [32], and danceON [71], respectively) that center liveness and a close synchrony between input and effect. $\mathbf{P}_{\text{Willie}}$ noted that "every single tweak happens immediately with danceON", and expressed that one of the key values of Tidal Cycles [59] (a live musical performance DSL used in FilOrk [72]) was how "the developer of Tidal Cycles, has put a ton of thought into its liveness".

Nevertheless, despite embracing the desire for liveness, $\mathbf{P}_{\text{Baku}}$ believed that the technological manifestations will always, perhaps beneficially, fall short of the ideal: "I think the imagination is my most real time media".

5.1.3 Challenge: Navigating Fast and Slow. Whether or not it is beneficial to speed up certain tasks is particularly difficult to judge when the users are students.

For instance, $\mathbf{P}_{\text{Tega}}$ observed that sandboxed frictionless interfaces like the p5 editor can lead to trouble after the course ends: "A lot of students have done a whole semester of p5 and they still don't understand how to embed a sketch into a website, because they've always just stayed in the editor. So the editor works really well. And I think one of the drawbacks of having it work so beautifully, is that it's very easy just to stay in that safe space, and then never actually ask: 'Oh, well, how do I run it locally? How do I stick it on my weird, esoteric website?'" Making it easy to make art in a course and building a sustainable art-making practice on the internet are naturally in tension, as teaching real-world practicalities may impede other topics. While tool-based solutions might make it easier to mount p5 sketches in new contexts, it is unclear if this benefits students' overall technical literacy compared to pedagogical solutions.

One technological design factor that may plausibly reduce friction is the use of graphical programming interfaces (e.g. Scratch), which rely less on conventional text-based programming and its "eccentricities" ($\mathbf{P}_{\text{Chris}}$). However, $\mathbf{P}_{\text{Willie}}$ noted that he avoided using GUI-based coding tools because "block interfaces don't feel like real coding". Similarly, $\mathbf{P}_{\text{Nick}}$ observed "I want [students] to learn how to ride a bike, because they're gonna get a lot further on a bike than in some esoteric, you know, device that I make, and then later [they] have to relearn how to ride a bike." Weintrop and Wilensky [104] explored student perceptions of block-based environments, finding that they are sometimes perceived as being less powerful, slower (something $\mathbf{P}_{\text{A}}$ mentioned), and inauthentic relative to tools used in practice [60, 86]. $\mathbf{P}_{\text{GL}}$ reified this authenticity in economic terms: "If I'm paying $60,000 a year, for a degree, I [expletive] well want to learn something that's going to be seen as useful when I graduate." Indeed, choices about which technology to use—in terms of its SLOWNESS, or any other technical characteristics—take place within a broader context of concerns and POLITICS.

5.2 Politics: Considering Power

When considering which art-making and pedagogical tools to use, creative-coding educators must also consider the POLITICS of the systems in question. As $\mathbf{P}_{\text{Nick}}$ explained it, "we're all human and we all have biases, and those work their way into the technology and get amplified and propagated". Technologies "are not just tools that we use, but environments that we're living in; they're affecting us" and guiding the types of work that are made. Our discussion is organized by how participants navigate power expressed over them, and how they in turn express power over others—reflecting Li et al.'s [52] considerations of power in CSTs.

5.2.1 Navigating Power Structures. With few exceptions, everyone who codes necessarily uses technologies designed by others. Less technical users (such as novices) are especially beholden to those external power structures, as they are rarely the authors of their own tools. In classroom settings, both students and educators are "users" of many such technologies.

Personal Obligation. A key aspect of power dynamics involves what is required of users to access these systems.

For instance, the cost of commercial art tools was a prominent concern. Standard professional tools, such as Adobe's Creative Suite, can cost more than $700 USD per year [2]. "I don't want my students to be paying rent on their software," said $\mathbf{P}_{\text{Allison}}$, voicing a position shared by others. $\mathbf{P}_{\text{Chris}}$ emphasized that his department was hesitant to select Adobe products, because doing so could be tantamount to "literally burdening your student with a lifetime of submission". (In a 2024 FTC filing [87], Adobe has allegedly made it more difficult to disentangle oneself from using their products.) $\mathbf{P}_{\text{GL}}$ noted that one of his main goals in teaching creative coding was "to reclaim computation as a medium of personal expression" from

corporate control. $\mathbf{P}_{\text{Baku}}$, $\mathbf{P}_{\text{Cassie}}$, and $\mathbf{P}_{\text{Nick}}$ suggested that building tools in the browser can address some of these issues, because of the ubiquity and openness of the web—at the same time, the tendency of such systems to bit rot (see below) was also a concern.

Beyond costs, another concern pertained to the types of works that are made easy by commercial tools. $\mathbf{P}_{\text{B}}$ observed that their aversion to the Adobe suite is "not even about the cost. It's about the agency behind the code"—how can an artist control and own what they have made if it is locked behind proprietary tools? $\mathbf{P}_{\text{GL}}$ echoed this sentiment: "I'm constantly fighting against corporate tools like Photoshop or Unity...how they try to make it possible to not have to use code to do it". He continued "Adobe cannot sell a tool that allows you to do things that you truly haven't been able to see before. If they did, nobody would buy it". Artists outside of the classroom are highly aware of these tensions, as in Dahnke et al.'s [15] recent collection of writings on creative resistance to technological dominance.

Reconciling these considerations directly with personal values—and indirectly with their students' values—can be difficult. $\mathbf{P}_{\text{B}}$ emphasized that they "tried to use software by foundations"—such as open-source software toolkits for the arts—which they see as being composed of works that follow the adage "software should be done by people, for people". While there are reasonable criticisms of a foundation model [6] they offer a model for governance and maintenance (which can be, but is not necessarily, sustainable), and a signal to potential users about the future and values of that tool. However, $\mathbf{P}_{\text{Allison}}$ noted that open source tools are rarely free from uncomplicated funding as well: observing that for many projects "you don't have to go far before you start hitting against corporate interests, government interests, military interests".

Instead of considering these issues directly, some participants favored practicalities in technology selection. For instance $\mathbf{P}_{\text{Chris}}$ observed that he ended up using MaxMSP (owned by Cycling '74) instead Pure Data (an open source close alternative), because "At the end of the day, and so I don't bring [Pure Data] into the classroom" because while Pure Data "was beautiful as open source, and it's sparse, but it's not friendly." Deciding between tools that are aligned with a classroom's values and ones that are pedagogically or creatively useful can be challenging. There is unlikely to be a single unambiguously good choice in this wicked [81] situation.

External Forces. One of the biggest challenges in making tools for creative communities is not the implementation of complex features, but rather addressing sociotechnical constraints.

Centralized technologies or services often change, leading to code-based artworks, teaching materials, and tools being unavailable or non-functional (or bit rot). For instance, $\mathbf{P}_{\text{Nick}}$ described how a change in a Firefox extension API killed a multi-year project. As a result, he modified his development posture is to prevent similar damage. $\mathbf{P}_{\text{GL}}$ offered a historical perspective: "I've seen a generation of artists, who worked with Macromedia/Adobe Director, make thousands of apps that can't be seen anymore. And then I saw another generation of people, five years younger than them, make a bunch of things with Flash—for example, Newgrounds.com—and they [made]...thousands of games and interactive artworks...that nobody can see anymore, because Flash doesn't work in the browser". Although projects like ruffle [82] have revived many Flash-based projects, $\mathbf{P}_{\text{GL}}$ noted that "you're at the mercy whether somebody has made an emulator, because they're that nostalgic... Good luck." Vasudevan [99] and Snodgrass and Soon [92] echo this perspective, finding that many artists have been forced to embrace this impermanence, due to changing APIs, system updates, or policies (e.g. auto-playing web audio no longer being allowed). Nevertheless, considering long-term maintainability as best as possible in the selection of tools seems an important consideration.

Beyond changes to platforms is how they are organized and run—that is, their governance. $\mathbf{P}_{\text{Cassie}}$ explained "It's really difficult to make decisions amongst a group of people. I really think the solutions are...structures of power... Governance is such a huge part of open source projects"—echoing OSSTA's findings [57] that sociotechnical issues like leadership and fostering community were critical the success of arts toolkits. While it is natural for toolmakers to develop technical interventions that might make these processes better, this kind of techno-solutionism may be poorly matched with problems that are better handled through policy.

Surrounding governance are issues concerning governments. $\mathbf{P}_{\text{B}}$, who works in a developing nation, observed that there were often governmental concerns to navigate which may have shifting priorities when it comes to supporting the arts, education, or even internet access. They observed that "I don't take for granted that they don't want to unplug me (again) from the internet." This observation guided an emphasis on using local hardware (e.g. Arduinos) in their courses. Designing tools with instability in mind, rather than assuming constant internet access on top-tier computers, may open the door to more contributors and communities.

Less drastically, these bodies also often impose bureaucratic and legal hurdles. $\mathbf{P}_{\text{C}}$ recalled how a cloud-based platform she works with "had to go through this form request from an educational board. And a lot of it was concerned about like cybersecurity", which included constraints like "if they request us to shut it down, then they expect then the agreement says that we should shut it down when they request it" which was broadly infeasible for tools with tens of thousands of users. This is a common issue in US K-12 schools [9], which can lead to selecting potentially suboptimal technology to accommodate local laws.

5.2.2 Interrogating Power Over Others. While navigating power structures associated with existing technologies or the design of new tools, educators are also by definition in a position of power in the classroom. Students yield power to educators by investing time and money at the cost of other opportunities.

Grading and Evaluation. As the contexts in which our participants taught varied, so too did grading and evaluation processes. Those who taught in less formal workshop settings tended to emphasize positive experiences over critical feedback. For instance, $\mathbf{P}_{\text{Willie}}$ taught a fifteen-week "internship" with young women of varied technical background, noting that "it's never ever, like, 'that's wrong, or that's bad'". Similarly, $\mathbf{P}_{\text{Allison}}$ noted that her program uses a "very gentle art crit... We always applaud at the end, and we never really dig deep into" students' pieces. She went on to observe that this style of critique was designed to support students emotionally, if not artistically: "I don't know if it has great outcomes, but I think the emotional outcomes of it, at least, are good.

Because I feel like people exit the class usually feeling okay about themselves". Building students' confidence in this subject seems particularly important, given the well-known obstacles that many students face when studying computer programming [78].

$\mathbf{P}_{\text{GL}}$ and $\mathbf{P}_{\text{Chris}}$ noted that the framing of creative coding as art classroom adds incentives unrelated to grades, but instead on more student-centered outcomes, such as the quality of the work. For instance, despite the lack of traditional point-based penalties in $\mathbf{P}_{\text{GL}}$'s courses, "there is a penalty in my course if you don't have your work in time for the crit, which is that you don't get [expletive] crit". Moreover, $\mathbf{P}_{\text{GL}}$ sometimes invites (and gives honorariums to) well-known artists to critique students' work. The penalty for not completing work on time is then an extrinsic interaction with power: students would not have the opportunity to learn how to make their work better, nor the opportunity to interact with those whose opinions they might value.

Participants were hesitant about automated critique (as Chung et al. [11] favor), arguing that such automation was mostly valuable in computer science courses and not in art. We suggest that automated evaluation might be more fruitful not on an artistic level (per the satirical critique found in the video game Art Sqool [74]), but instead on a craft level, i.e. guiding basic usage and best practices. Students in creative coding classrooms seem receptive to the utility of linters [60], indicating that tools could capitalize on learning opportunities. To wit, Davis et al. [16] explore an automated critique system that guides novice filmmakers on best practices.

Community. Power can be expressed by toolmakers over those who use and contribute to those tools. For instance, $\mathbf{P}_{\text{Chris}}$ noted a realization that "oh, actually, I have a whole bunch of responsibility that comes with" building and maintaining tools people use—responsibility being an artifact of power.

Participants broadly viewed the efforts of the Processing Foundation and those associated with it as being a success story in this regard. $\mathbf{P}_{\text{Cassie}}$ contrasted the intentional kindness espoused in Processing with other projects: "Have you ever tried to contribute to a different open-source project? People are vicious". $\mathbf{P}_{\text{Chris}}$ observed that its success is in part due to "how friendly and engaged is that community... They think deeply about community, they think deeply about what contributing is, they think deeply about all the steps that they take". For instance, contributions to the p5 editor [68] can come through code but also "through documentation or developing teaching resources" ($\mathbf{P}_{\text{Tega}}$). There is an intention to make contributing to the project as easy and inclusive as possible. $\mathbf{P}_{\text{GL}}$ explained "Lauren [McCarthy] and Taeyoon Choi taught me what a contributor is [or] can be...that we can decolonize our very notion of what a contributor is". Building tools and communities that are inviting and open to new contributors is challenging and without obvious technical solutions, but, we add, of pressing value.

5.2.3 Opportunities for Restructuring Power. Power does not need to be challenged directly, but can be approached subtly or conceptually. $\mathbf{P}_{\text{Cassie}}$ and $\mathbf{P}_{\text{B}}$ observed that the idea of a "sketch"—synonymous with "program" in the Processing/p5 ecosystem—was particularly useful. $\mathbf{P}_{\text{Cassie}}$ observed that this terminology "takes it from the idea of writing code from, like a business context [and] having users, to think[ing] of it like an artist's sketch... The decision to name it in that way was extremely intentional." Following these footsteps may lead to other opportunities for bending labels and conceptual models of coding and its predominantly professionalized toolchains. To wit, $\mathbf{P}_{\text{Allison}}$ observed that there has not yet been a satisfactory notebook model of a sketch—highlighting the potential of something like a *sketchbook*.

Creative coding tools are not merely means to facilitate rapid input (echoing SLOWNESS), but include the materials and communities that surround those tools, which make it possible to use them in a fulfilling manner. Verano Merino and Sáenz [100] observed that creative coders often value the quality of documentation over the quality of the surrounding tooling, which comports with these perspectives. While tasks like documentation are primarily sociotechnical labor, there may be some opportunities to ease the process for novices to contribute to documentation. For instance, $\mathbf{P}_{\text{C}}$ thought that making it simpler to contribute documentation (such as without using git) was potentially valuable. Touching on the merits of approachability, we suggest that one interface idea could be to leverage familiar notions of comments, suggestions, and change-tracking found in word processors, such as Google Docs, rather than the esoteric concepts used by version control systems.

5.3 Joy: Embracing Play and Process

Lastly, we explore the ways JOY can manifest in this context, through playful experiences that incentivize learning, as well as opportunities to connect with communities and practices.

5.3.1 Playfulness. A central appeal of creative coding in the classroom is that it allows learners to "relate to the topic at hand to their interest and got excited about it" ($\mathbf{P}_{\text{Cassie}}$). In addition to the selection of topics, playfulness can result from teaching practices and the tools themselves.

Playful Teaching. One approach was to make coding feel unintimidating and approachable. For instance, $\mathbf{P}_{\text{B}}$ emphasized the low stakes in their teaching making space for joyful practices, suggesting that students' attitudes should be of the mind "*you're not gonna be on the cover of The New York Times for this project... Let's have fun, let's not suffer*". Similarly, $\mathbf{P}_{\text{Willie}}$ noted that "it's important to me that they make something that they feel good about that reflects their aesthetics and their ideas at the same time."

$\mathbf{P}_{\text{Allison}}$ explained that "we teach p5.js with the idea of that—because it's visually oriented—lots of people can learn pretty well from visually oriented things." This idea, about intrinsically motivating applications of programming, is echoed in prior works on creative coding [25, 60, 73, 107].

A related strategy was to give students something concrete that they can own or share. For instance, $\mathbf{P}_{\text{Chris}}$ emphasized the value of OpenProcessing and the p5 editor, because works made with "both of those are super shareable". $\mathbf{P}_{\text{Willie}}$ highlighted that thinking carefully about course design can yield big payoffs, recalling that students in his dance internship had a final performance which "incorporates these really creative ideas and original choreography and animations that map to that choreography. And, you know, just like that, the technology facilitated that".

$\mathbf{P}_{\text{Allison}}$, $\mathbf{P}_{\text{B}}$, $\mathbf{P}_{\text{Cassie}}$, and $\mathbf{P}_{\text{Tega}}$ contrasted such approaches with their own more traditional CS education. $\mathbf{P}_{\text{B}}$ emblematically explained that "my experience was brutal. When I went to engineering

school first, I was miserable", describing it as "very violent teaching". Such sentiments are emblematic of "weed-out" courses, which, as Weston et al. [105] describe, are often found in STEM disciplines and which create (often needless) difficulties for students.

Outside of formal class settings, $\mathbf{P}_{\text{Cassie}}$ suggested that the success of a specific YouTube channel that teaches MaxMSP was "really popular because [the host] just had a playful energy", which they also noted was the signature of prominent creative coding educator Daniel Shiffman. $\mathbf{P}_{\text{Allison}}$ noted it was important for educators not just to seem like they are having a good time, but to select tools that are actually enJOYable for them: "if I'm not having fun when I'm teaching, then it's just that it's a nightmare for everyone involved". Building tools that are pleasurable to learn, to teach, and to use in practice is hard; highlighting a complex HCI and design challenge.

Playful Tools. An important form of play in this context involves experimentation. $\mathbf{P}_{\text{Nick}}$ explained that students come into the studio "without a plan, and they start experimenting, and they see where that experiment takes them". Supporting this type of unstructured exploratory programming [43] seems to be an essential part of creative coding technologies. Prior work [4, 79, 93] has considered how to support this type of exploration with creative version control, allowing programmers enough leeway to remain creative and to provide a historical view of their work to various ends.

A component of play is feeling welcome to do so, yet it is not always clear which tools or design choices might best help support the goals of approachability and inclusivity. For instance, $\mathbf{P}_{\text{Nick}}$ offered a historical perspective, observing that "JavaScript is a lot friendlier and an easier introduction than C++. And now, we're like: 'JavaScript, is that really friendly enough?'". In contrast, $\mathbf{P}_{\text{GL}}$ argued "JavaScript is not specifically really inclusive... There's intense use of non-alphanumeric characters, semicolons, and different" symbols, suggesting that inclusivity would require design choices that align programming languages with everyday speech. Beyond the tools themselves, $\mathbf{P}_{\text{GL}}$ observed that there are other factors: "What's inclusive is, are the faces of the people who are making these things. What's inclusive, is the documentation that's written in a way that maybe doesn't assume you already understand the documentation you're trying to read about." However joyful and inclusive system *features* may be, there are still the considerations about the surrounding POLITICS of the system—echoing our previous theme.

5.3.2 Process. The role of educator can be seen in many ways, such as conveyors of information, as arbiters of grades, or exemplars of ways of being. To the last of these, some participants saw their role as showing students a way of being—rather than a mere course-long engagement—and that demonstration of process was, in essence, a JOYful invitation to be a part of an artistic community.

Joy in Coding. Some participants highlighted JOYfulness as a specific endpoint through which engagement with a process may lead. In her teaching, $\mathbf{P}_{\text{Allison}}$ says "I'm trying to impart to my students some of that joy of doing computer programming, because it is a joyful thing. For me, that's the primary reason that I do it is that it brings me happiness". That is, she attempted to impart not just her love for coding, but rather the process in which that love manifests, so that students might replicate that feeling themselves. $\mathbf{P}_{\text{Cassie}}$ echoed this position, observing that they strove to help students "feel like part of the [creative coding] community and like [they] belong"—emphasizing that practices like these are rarely individualistic, but rather are part of something larger. This connects with prior work [10, 25] that describes how creative coding can facilitate a sense of belonging for people sometimes otherized in computing adjacent areas.

Others were more explicit about the process itself. An essential part of $\mathbf{P}_{\text{A}}$'s teaching is an emphasis on demonstrating process: "You gotta get your hands in the clay. And I use that literally and metaphorically... It's made me a pretty strong proponent of teaching students through doing and allowing them to develop their own practice and process". This practice is analogous to *live coding* (i.e. writing code during lecture rather than coding as performance) as increasingly used in traditional computing classroom settings [85].

This perspective governed not just attitude, but also selection and style of pedagogy. $\mathbf{P}_{\text{Chris}}$ noted that "they're going to have to keep learning for the rest of their life if they're going to stay in coding. And so I need to enforce that process right away... The actual technical competency is a little bit less important" compared to the experiential process of debugging and finding resources. Similarly, $\mathbf{P}_{\text{Cassie}}$ noted that they had "been super inspired by Daniel Shiffman. I really like his approach of being open to making mistakes and debugging in front of you". Demonstrating process then is not about perfection, but showing the holistic cycle of doing work.

Some participants noted that there was a relationship between JOY and rigor. For instance, $\mathbf{P}_{\text{Allison}}$ commented that "part of the joy of computer programming for me is, is its rigorousness, the fastidiousness with [which]...you need to approach it." She continued that "there's a certain joy in learning a programming language and learning to talk in the way that the programming language wants you to talk and learning to solve the problems in that way". Understanding the beauty in truth tables, binary, and the physical presence of a computer are accessible through concrete engagement with technical ideas and concepts.

Joy in Engagement. $\mathbf{P}_{\text{A}}$ observed that some levels of engagement were only possible with a certain level of proficiency and therein investment: "conceptual play that often happens when you talk about art at that level. Right? It's like, suddenly, you're having this kind of higher-level conversation about movement and signification." Echoing Shneiderman's [90] "low thresholds, high ceilings, and wide walls," an important component of tools is the facility to allow not just introduction but expanse. $\mathbf{P}_{\text{Chris}}$ summarized that in p5, "you can be expressing yourself, like day two, pretty easily... And then there's a whole other layer of 'Oh, holy [expletive], I can actually do things! I can like turn particles into graphics buffers, and do GPU compute stuff.' And there's a whole other layer that gets unlocked that maybe isn't there for something like p5.js". While tools like p5 may offer a welcoming invitation, its guard rails may impede access to richer topics and design experiences.

Connecting with elements of SLOWNESS, $\mathbf{P}_{\text{A}}$ noted that "it feels like art is continually this process of reflection in action. You're never not reflecting. And sometimes it can be hard. I think, maybe sometimes it's indicated by getting stuck". $\mathbf{P}_{\text{Willie}}$ similarly observes that "I really like reflection in this type of work of like, students do something and then they write about what was hard and what was easy and what they want to do next." We suggest that there may

be additional space for tool-based intervention in order to prompt reflection, a space explored by QuickPose [79] (in which reflection is prompted through a process change) and LitVis [106] (where reflection is directly prompted via linted questions).

5.3.3 Joyous, Not Toyish. Tools for creative coding need to avoid being so playful and toy-like that they cannot be used for serious work, while also being not so serious and feature-rich that they cannot be approached by beginners—as $\mathbf{P}_{\text{Cassie}}$ put it "you want a tool to invite you in". At one end of the spectrum is Compton's [13] notion of tools for casual creators, capturing those tools that enable their users to make fun and interesting artworks that are not expected to be of a production quality. While making it easy to get started is important (as $\mathbf{P}_{\text{Chris}}$ stresses), selecting the right balance of floor and ceiling height [90]—e.g. does a tool leave room to grow or just make it easy to get started—does not have a single answer.

Using multiple tools with different characteristics might help techniques learned in one context transfer to others. In one of $\mathbf{P}_{\text{GL}}$'s courses, code-based artwork with p5.js is sequenced before professionalized GUI-based tools like Touch Designer, so that the basic computational thinking from textual programming can still be gained while also learning industry-standard tools. This highlights an inherent difficulty in creative coding courses: many works made simple by tools like Processing are often also unimpressive compared to what is possible with professionalized tools. Paradoxically, however, the lessons about computational thinking available in tools like Processing are often more valuable than those in higher-end professional tools. Striking the right balance between enabling impressive designs and pedagogically valuable designs is a wicked problem with no one answer, but, we suggest there are opportunities to explore designs that support offer high-level joyful control while still imparting transferrable computing knowledge.

6 Tool Design Considerations

The themes established in our analysis can help understand the original question for this work: How should tools be shaped in the context of creative coding classrooms? Some aspects of this question have already been touched on. For instance, a high degree of liveness [96] can be a valuable way to make an interface feel fast and dynamic, fomenting JOYful experiences that allow flow states—but this can come at the cost of opportunities for SLOW, careful reflection about the values of the work at hand.

Next, we use the themes as critical lenses to consider two additional facets of tool design that frequently arose in our interviews: accessibility and AI. The types of questions that can be asked about these facets in the context of our themes are exemplified in Fig. 3.

6.1 Accessibility

A basic belief expressed by most participants was that students of all abilities and backgrounds should be able to use the tools used in class. Disabilities come in many different forms, kinds, and contexts.

For instance, a substantial amount of $\mathbf{P}_{\text{Willie}}$'s work focuses on improving the accessibility of creative technologies. For instance, his FilOrk [72] specifically aims to make live coding accessible and enJOYable to blind and low-vision students. He highlighted the specific value of the web to help them work "on their own devices. And this is like really important for the accessibility aspect of it, because the students with some vision are using iPads that they really like because they can zoom and move around the screen really well". For instance, $\mathbf{P}_{\text{Chris}}$ noted that his students valued coding from their own devices rather than through a browser. $\mathbf{P}_{\text{Willie}}$ added that the "benefit of moving from something that's installed with your hardware to something in a browser is that...it can be accessed across a ton of different devices. In this case, a ton of different screen readers". While this kind of democratization is valuable, it can be in tension with notions of authenticity. In designing courses and tools, those in power need to navigate these tensions, and ask (POLITICAL) questions about whose needs to need to be centered.

Platform support for accessibility has been slow going. Potluri et al. [75] observe a wide array of accessibility challenges in computational notebooks for blind and low-vision users. $\mathbf{P}_{\text{Allison}}$ echoed these findings, noting that "I've been teaching with Jupyter Notebook for a really long time, but they just barely in the past couple of years have been working on making it screen reader accessible". Referring to tools like Scratch, $\mathbf{P}_{\text{Willie}}$ similarly noted that "I didn't want to do blocks because of accessibility considerations". Developers of prominent projects are aware of this need. $\mathbf{P}_{\text{C}}$ noted that a priority for her in open-source work is "thinking about how [to make tool development] accessible and friendly for all the other users within the community" besides just able-bodied developers.

$\mathbf{P}_{\text{Matt}}$ observed that there seems to be a place for AI-driven technologies in making some creative coding tasks more accessible, such as by creating "a hands free coding system" mediated by AI—potentially being beneficial to those with physical disability. $\mathbf{P}_{\text{GL}}$ echoed this position, specifically valuing some of $\mathbf{P}_{\text{Matt}}$'s recent experiments. Beyond shifting attention towards creative or directorial style agency, reducing SLOWNESS latent to some tools can make those tools accessible to a wider audience—particularly those for whom the affordances of those tools are not well aligned.

Not all accessibility challenges arise from commonly considered disabilities like blindness. For instance, $\mathbf{P}_{\text{Allison}}$ observed that "another dimension of accessibility is neurodivergence and...especially how to make the kinds of programming that I think are joyful accessible to people with all kinds of with all kinds of different learning styles, and all kinds of different ways of thinking. And understanding that can be difficult, as we have this history of teaching computer programming in a particular way that kind of only works for certain kinds of people." Similarly, $\mathbf{P}_{\text{Nick}}$, $\mathbf{P}_{\text{Matt}}$, and $\mathbf{P}_{\text{Allison}}$ noted that language barriers were also common, with $\mathbf{P}_{\text{Nick}}$ observing that CSS "eventually starts to become intuitive for [English-speaking] students", whereas the vast array of properties can be hard to connect with for others (such as Spanish or Chinese speakers). Chung and Guo [10] echo this observation, noting that, for Korean students, the English aspects of common programming languages can be scary—although this feeling can pass with familiarity. Huang's wenyan [33] language interrogates the English-language bias of many programming languages by supporting ancient Chinese. $\mathbf{P}_{\text{Allison}}$ noted that some basic assumptions latent to creative coding technologies were not held by all cultures, observing that p5's model of "art is a two-dimensional surface that has shapes on it, right? But that's a drawing from a particular tradition of art. That

Theme	Generic	Accessibility	AI
SLOWNESS	Does this tool increase or decrease opportunities for reflection?	For whom does this system become unduly friction-ful?	How does this tool give or take agency?
POLITICS	Who does this tool empower or disempower?	Whose needs and abilities are centered? Whose are marginalized?	What values are trod upon through this tools usage?
JOY	How does this tool allow play? What new horizons of joy does it enable?	Whose joy is being prevented?	Does this tool boringly reproduce things that already exist?

Figure 3: Example questions about two application areas formed from our themes, as well as generic questions for any tool or area. We use our themes as critical lenses, although they could be used as generative or analytical framings [5].

is not universal". Similarly, $\mathbf{P}_{\text{B}}$ noted that there was an element of classism in who was allowed to use these technologies, highlighting the huge cost associated with institutional learning or just having the time to dedicate to pursuing free resources.

Accessibility is contextual (and POLITICal), and so a tool choice can be inclusive, be inviting, and be JOYful for one person, but be exclusionary to someone else.

6.2 AI

The role of AI-driven tools in the classroom loomed large with participants, eliciting strong reactions of dread (such as from $\mathbf{P}_{\text{Allison}}$), disinterest (such as from $\mathbf{P}_{\text{Baku}}$), and in some cases optimism (such as from $\mathbf{P}_{\text{Matt}}$). Participants touched on both tools for code (e.g. ChatGPT) and image (e.g. Adobe's Firefly) generation.

Several participants stressed that the type of work that AI-based tools can help produce was "boring" ($\mathbf{P}_{\text{GL}}$) and unJOYful. $\mathbf{P}_{\text{Baku}}$ observed that "AI can improve our efficiency or productivity, but it doesn't improve the joy or satisfaction of making something from scratch". He went on to observe that this was a key motivator in the design of Glisp [32], which is "oriented to totally opposite the way of efficiency or productivity is, it's just for making that process of drawing graphics more satisfying or more joyful". Similarly, $\mathbf{P}_{\text{Allison}}$ commented that "for me, [using AI] is completely joyless because you aren't engaging in those problem solving skills, and you aren't like engaging with the materiality of the programming languages themselves". $\mathbf{P}_{\text{B}}$ likened the usage of GPT to the POLITICAL problems associated with using Amazon's Mechanical Turk. Jiang et al. [39] explore the harms that the use of art in AI can cause, leading to reputational damage or copyright infringement.

Some participants added nuance to this view. $\mathbf{P}_{\text{Chris}}$ observed that LLMs can help students get to the creative part of creative coding and not linger in the SLOW parts: "Because maybe it means that actually everybody can get a little bit further to making art a little bit quicker, because they've got an AI assistant. I'm okay with that." Similarly, $\mathbf{P}_{\text{Willie}}$ observed that it can be useful for "the kind of menial tasks, like, you just need some drum groove that's interesting" and to "help you generate initial ideas". A participant with teaching experience in Verano Merino and Sáenz's [100] study noted that, even in beginner-friendly tools, the "blank window" and sense there is "a flood of things you have to know in the beginning" are obstacles that confront students—barriers which might usefully be addressed through AI-style tooling. Like us, Jonsson and Tholander [40] connect friction to automation, finding that AI systems can help reduce friction in the context of creative coding, but in doing so can lose some useful friction that prompts creativity. Circumventing the debate about AI tools, $\mathbf{P}_{\text{Nick}}$ noted "I've been using tools that generate code for as long as I've been coding", connecting generative coding systems with tools like Dreamweaver. Further, $\mathbf{P}_{\text{Nick}}$ stressed that his position as an educator was often to help convey literacies of different sorts, such as for code, for the internet, or for art. To this end, he noted "prompt engineering could be one of these new literacies," and "that in terms of my role in [conveying those literacies], I don't think it's changed".

Generative AI tools geared toward introductory coding (as in CodeAid [41]) and creative coding (as exhibited by Spellburst [4] or Keyframer [98]) are becoming more prominent. Ippolito [37] describes a recent creative coding-adjacent course in which AI was centered, finding that it was broadly useful. Google Labs has an upcoming Chrome extension called Shiffbot [46] that provides an LLM-based assistant in the p5 editor that emulates the teaching style of Daniel Shiffman [88]. This type of interactive tutor system seems promising, in that it may still allow users appropriate time for reflection prior to intervention (i.e. supporting appropriate SLOWNESS), which Wang et al. [103] observe is an essential component using LLMs for creative coding tasks. Similarly, by emulating Shiffman's playful style—much in the same way that $\mathbf{P}_{\text{Cassie}}$ observes that their style was guided—the JOY of coding might similarly be conveyed without exchanging it for an unintuitive black box and may cause students to forgo valuable learning opportunities (as Lau and Guo [47] find that some educators fear). We add that such a system is naturally liable to the whims of corporate winds. For instance, if the price of LLMs radically increases (such as due to regulatory or market shifts), the viability of this style of assistant as a free tool will be limited. $\mathbf{P}_{\text{Tega}}$ framed this pricing as an equity issue (reflecting POLITICS), noting that "because if you can pay for the better subscription with OpenAI, you get much better results than the generic free stuff that I'm sure all of our students are using." Lastly, $\mathbf{P}_{\text{C}}$ noted that perspectives can change with time, likening acceptance of AI-based tools to the gradual way she accepted autocomplete in her phone. Perceptions of AI systems will change as their capabilities and ubiquity change.

7 Discussion

This paper explores one facet of the creative coding landscape, namely, the values that creative coding educators use when considering what tools to bring to their classrooms and what tools to build for themselves. We identified several themes (SLOWNESS, POLITICS, and JOY) which can be used to critically reflect on the values a given tool might bring to the fore, highlighting takeaways for each theme (Fig. 2). In doing so, we considered different areas in which technical

intervention or innovation might be useful, but also a variety of areas where techno-solutionism might yield unnecessary tools. We applied these themes to considerations of tool accessibility and the role of AI in creative coding tools. We conclude by reflecting on the context of this work and its connection to future HCI research.

Humans and Their Identities. The core identities represented in this work are those of artist, educator, and toolbuilder—as well as their intersections. These often appear to coalesce naturally (with, as $\mathbf{P}_{\text{Allison}}$ and $\mathbf{P}_{\text{Nick}}$ described, artistic practice sometimes informing teaching or tool building and vice versa), but there are times when these characteristics come into tension.

For instance, one of the most common stances for tool builders is to metaphorically smooth the edges off interfaces (cf. Krug's Don't Make Me Think [45]), which can conflict with the friction (i.e. SLOWNESS) that is sometimes valuable for education (such as to learn craft) or artistry (per Li et al.'s [50] observation that artists prefer granular control over their tools). Disentangling these interwoven tensions involves reflection on the intent and domain of the work. As these are fundamentally human issues, we stress that there is no single preferred path through this thicket of concerns—just as how each of these identities carries with them different relationships to our themes. Instead, we suggest merely that these dimensions are worth at least considering in tool design and selection.

As a thought experiment, consider using Google Sheets as the foundation for a creative coding course—which is, at the very least, *a* plausible way to make art [20]. While there is arguably little perceived JOY in writing spreadsheet formulas, there is a recognizable kind of JOY in mastering the functional programming concepts latent to spreadsheets—and perhaps a useful SLOWNESS to interacting with those concepts. Similarly, there is JOY in being a part of the vast community of people who use spreadsheets and engaging with the rich ecosystem of resources (e.g. message boards) surrounding them. Some may object to the POLITICS of using corporate-owned technologies—however, the amount of resources behind such tools may make it more accessible. Or, if the course is online, it may preclude usage in parts of the world where Google is blocked. What then is the answer? Alternative spreadsheets could be used, new ones could be developed, or it could be decided that these tradeoffs are acceptable. We suggest considering how one's values connect with these sociotechnical concerns offers a valuable entry point for reflecting on and approaching these issues.

Technology and Their Features. Throughout, we have noted areas where additional (technological) research might be useful, however we stress that these tasks should be pursued in community. While $\mathbf{P}_{\text{Allison}}$ noted that "I do think the p5.JS tooling could be better", we suggest that these are predominantly community-level or maintenance-level tasks, given that "p5.JS, in terms of their mission statements, and so forth, they basically said, we're not introducing new features, unless they specifically address inclusivity" ($\mathbf{P}_{\text{GL}}$). That is, while things could always be better, we suggest that building the next great creative coding editor or tool is not the only critical task.

For instance, while $\mathbf{P}_{\text{Matt}}$ was enthusiastic about a professional quality set of generative art tools (e.g. something like Photoshop for generative artists), he expressed significant hesitancy: "practically speaking, it's really hard to imagine how that works. Because all these tools approach things like frame rate and render loops and things very differently."

Instead the most impactful approach seems to be centered on developing sociotechnical systems and improved affordances around those tools. For instance, making it easier to contribute to documentation by removing the barrier of GitHub pull requests (as $\mathbf{P}_{\text{Cassie}}$ and $\mathbf{P}_{\text{C}}$ highlight), or developing systems that simplify accommodating political or regulatory burdens. Similarly, creative coding is not a monolith, and there are many other associated practices which may have other needs, concerns, and values, such as in new media art [10], permacomputing [56], or the demo scene [84].

Creative Coding and Its Research. Despite some participants chafing at the term creative coding—$\mathbf{P}_{\text{B}}$ objected that "all code is creative"—we suggest that this is a useful area for HCI research. People in this area have complex challenges. They have had several decades of intricate and involved independent tool development. While academic HCI has recently demonstrated an interest in creative coding [4, 8, 10, 60, 79, 94, 95], $\mathbf{P}_{\text{GL}}$ commented on HCI's long disinterest in the code-based art, admonishing "creative coding has been around for 25 years. What took you guys so long?" We echo Li et al.'s [51] call to see artists as technical collaborators rather than merely subjects. It is easy to think of creative coding as yet another genre in which HCI researchers might appear, create tools, write a paper, and parachute away [53]. For this, or any other community, such behavior is an ineffective pattern of research that leads to undue maintenance burdens [3], tools not needed, tools that are too advanced or otherwise not usable by their intended users, or just tools that are not released—Frich et al. [21] report that most (>75%) of surveyed CSTs were unavailable to the public.

There is substantial space in which to support this community without being extractive. For instance, research on the p5 editor might include making enhancements to the public version of it, rather than merely creating irreconcilable forks. Making research tools open-source and freely available offers at least some remedy to these issues, as others can remake and remix those artifacts. The recent artifact badges at CHI [1] are a useful development for this type of engagement. Yet, free and public release of software, alone, is far from perfect: open-sourced academic abandonware is common, potentially creating non-trivial maintenance burdens [3, 26] and unkind online environments (per $\mathbf{P}_{\text{Cassie}}$) without careful community curation.

Echoing McCarthy et al. [57], we highlight that monetary funding is key area of potential support. Some corporate policies have made steps in this direction—to this point, $\mathbf{P}_{\text{B}}$ valued of Google's Summer of Code program being open to students outside of the USA [23]. Yet, in the USA there is little funding for the arts compared to other technical areas. Furthermore, as Vasudevan [99] observes, artists are often treated as a form of research and development for larger technological interests. Improving this situation might involve forming partnerships with artists and artist organizations (as advocated by Devendorf et al. [18]) in general and in grant proposals, or seeking policy changes to value the (technological) work that artists do.

There is a lot for researchers to learn from the creative coding community—about building tools that are useful for a wide range of people, about community-oriented initiatives, and about aligning technology with the values of the people who use them. How researchers might reciprocate and best support this community remains to be seen—so it is up to us to actively shape and direct our approach in thoughtful and meaningful ways.

Acknowledgments

We thank our participants for sharing their time and insights with us. We appreciate the helpful pointers and commentary given to us by Jean Salac, Brent Bailey, and Jeffrey Heer. We are also grateful for the useful suggestions provided to us by our anonymous reviewers. This work was supported by the Moore Foundation and the University of Chicago College Innovation Fund.

References

[1] ACM. 2024. Artifacts at CHI 2024. https://chi2024.acm.org/2024/02/08/artifacts-at-chi-2024/.
[2] Adobe. 2024. Adobe Creative Cloud. https://www.adobe.com/creativecloud/plans.html. Accessed: 5/22/2024.
[3] Derya Akbaba, Devin Lange, Michael Correll, Alexander Lex, and Miriah Meyer. 2023. Troubling collaboration: Matters of care for visualization design study. In *SIGCHI Conference on Human Factors in Computing Systems*. 1–15. https://doi.org/10.1145/3544548.3581168
[4] Tyler Angert, Miroslav Suzara, Jenny Han, Christopher Pondoc, and Hariharan Subramonyam. 2023. Spellburst: A Node-based Interface for Exploratory Creative Coding with Natural Language Prompts. In *ACM Symposium on User Interface Software and Technology*. 1–22. https://doi.org/10.1145/3586183.3606719
[5] Michel Beaudouin-Lafon, Susanne Bødker, and Wendy E Mackay. 2021. Generative theories of interaction. *ACM Transactions on Computer-Human Interaction* 28, 6 (2021), 1–54. https://doi.org/10.1145/3468505
[6] Ben Fry Resigns 2023. Ben Fry resigns from the Processing Foundation. https://www.reddit.com/r/processing/comments/1708ikb/ben_fry_resigns_from_the_processing_foundation/. This reddit post preserves context of an event that was splayed across several social media platforms..
[7] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative research in psychology* 3, 2 (2006), 77–101. https://doi.org/10.1191/1478088706qp063oa
[8] Cameron Burgess, Dan Lockton, Maayan Albert, and Daniel Cardoso Llach. 2020. Stamper: An Artboard-Oriented Creative Coding Environment. In *SIGCHI Conference on Human Factors in Computing Systems Extended Abstracts*. 1–9. https://doi.org/10.1145/3334480.3382994
[9] Jake Chanenson, Brandon Sloane, Navaneeth Rajan, Amy Morril, Jason Chee, Danny Yuxing Huang, and Marshini Chetty. 2023. Uncovering Privacy and Security Challenges In K-12 Schools. In *SIGCHI Conference on Human Factors in Computing Systems*. 1–28. https://doi.org/10.1145/3544548.3580777
[10] Alice Chung and Philip J Guo. 2024. Perpetual Teaching Across Temporary Places: Conditions, Motivations, and Practices of Media Artists Teaching Computing Workshops. In *ACM Conference on International Computing Education Research*. https://doi.org/10.1145/3632620.3671095 To Appear.
[11] John Joon Young Chung, Shiqing He, and Eytan Adar. 2021. The intersection of users, roles, interactions, and technologies in creativity support tools. In *ACM Designing Interactive Systems Conference*. 1817–1833. https://doi.org/10.1145/3461778.3462050
[12] Chris Coleman. [n. d.]. maxuino. https://web.archive.org/web/20230331113047/http://www.maxuino.org/.
[13] Kate Compton. 2019. *Casual creators: Defining a genre of autotelic creativity support systems*. University of California, Santa Cruz. https://www.escholarship.org/uc/item/4kg8g9gd
[14] Kate Compton, Ben Kybartas, and Michael Mateas. 2015. Tracery: an author-focused generative text tool. In *International Conference on Interactive Digital Storytelling, ICIDS 2015, Copenhagen, Denmark, November 30-December 4, 2015, Proceedings 8*. Springer, 154–161. https://doi.org/10.1007/978-3-319-27036-4_14
[15] Sarah Dahnke, Christina Freeman, Shawn Escarciga, Misha Foley, Caitlinand Rabinovich, Lydia Jessup, Tega Brain, Jasmine A. Golphin, and Lil Miss Hot Mess. 2024. *We Refuse, We Want, We Commit: Volume 1: The Manifestos for Creative Resistance in Technology*. https://book.strategictransparency.network/.
[16] Nicholas Davis, Alexander Zook, Brian O'Neill, Brandon Headrick, Mark Riedl, Ashton Grosz, and Michael Nitsche. 2013. Creativity support for novice digital filmmaking. In *SIGCHI Conference on Human Factors in Computing Systems*. 651–660. https://doi.org/10.1145/2470654.2470747
[17] Matt DesLauriers. [n. d.]. Canvas Sketch: A framework for making generative artwork in JavaScript and the browser. https://github.com/mattdesl/canvas-sketch.
[18] Laura Devendorf, Leah Buechley, Noura Howell, Jennifer Jacobs, Cindy Hsin-Liu Kao, Martin Murer, Daniela Rosner, Nica Ross, Robert Soden, Jared Tso, and Clement Zheng. 2023. Towards Mutual Benefit: Reflecting on Artist Residencies as a Method for Collaboration in DIS. In *ACM Designing Interactive Systems Conference*. 124–126. https://doi.org/10.1145/3563703.3591452
[19] Marc Downie and Paul Kaiser. 2021. Field. http://openendedgroup.com/field/.
[20] Excel Art 2024. r/Excel Art. https://www.reddit.com/r/excelart/.
[21] Jonas Frich, Lindsay MacDonald Vermeulen, Christian Remy, Michael Mose Biskjaer, and Peter Dalsgaard. 2019. Mapping the landscape of creativity support tools in HCI. In *SIGCHI Conference on Human Factors in Computing Systems*. 1–18. https://doi.org/10.1145/3290605.3300619
[22] Terkel Gjervig. 2024. Awesome Creative Coding. https://github.com/terkelg/awesome-creative-coding.
[23] Google. 2024. Google Summer of Code FAQ. https://developers.google.com/open-source/gsoc/faq. Accessed: 5/24/2024.
[24] Ira Greenberg. 2007. *Processing: creative coding and computational art*. Apress.
[25] Ira Greenberg, Deepak Kumar, and Dianna Xu. 2012. Creative Coding and Visual Portfolios for CS1. In *ACM Technical Symposium on Computer Science Education*. 247–252. https://doi.org/10.1145/2157136.2157214
[26] Philip Guo. 2021. Ten million users and ten years later: Python tutor's design guidelines for building scalable and sustainable research software in academia. In *The 34th Annual ACM Symposium on User Interface Software and Technology*. 1235–1251. https://doi.org/10.1145/3472749.3474819
[27] Mark Guzdial. 2003. A Media Computation Course for Non-Majors. In *Conference on Innovation and Technology in Computer Science Education*. https://doi.org/10.1145/961511.961542
[28] Mark Guzdial. 2013. Exploring Hypotheses about Media Computation. In *ACM Conference on International Computing Education Research*. https://doi.org/10.1145/2493394.2493397
[29] Mark Guzdial and Andrea Forte. 2005. Design Process for a Non-Majors Computing Course. In *Technical Symposium on Computer Science Education*. https://doi.org/10.1145/1047344.1047468
[30] Lars Hallnäs and Johan Redström. 2001. Slow technology–designing for reflection. *Personal and ubiquitous computing* 5 (2001), 201–212. https://doi.org/10.1007/PL00000019
[31] Stig Møller Hansen. 2019. Mapping creative coding courses: Toward bespoke programming curricula in graphic design education. In *Conference of the European Association for Computer Graphics*. The Eurographics Association, 17–20. https://doi.org/10.2312/eged.20191024
[32] Baku Hashimoto. 2021. Glisp: lisp-based graphic design tool. In *SIGGRAPH Asia 2021 Real-Time Live!* 1–1. https://doi.org/10.1145/3478511.3491312
[33] Lingdong Huang. 2019. wenyan. https://github.com/wenyan-lang/wenyan.
[34] Jessica Hullman, Eytan Adar, and Priti Shah. 2011. Benefitting infovis with visual difficulties. *IEEE Transactions on Visualization and Computer Graphics* 17, 12 (2011), 2213–2222. https://doi.org/10.1109/TVCG.2011.175
[35] hundredrabbits. 2021. Orca. https://github.com/hundredrabbits/Orca.
[36] Sarah Inman and David Ribes. 2019. "Beautiful Seams" Strategic Revelations and Concealments. In *SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1–14. https://doi.org/10.1145/3290605.3300508
[37] Jon Ippolito. 2023. AI versus old-school creativity: a 50-student, semester-long showdown. Still Water. https://blog.still-water.net/ai-versus-old-school-creativity/.
[38] Jennifer Jacobs, Joel Brandt, Radomír Mech, and Mitchel Resnick. 2018. Extending Manual Drawing Practices with Artist-Centric Programming Tools. In *SIGCHI Conference on Human Factors in Computing Systems*. https://doi.org/10.1145/3173574.3174164
[39] Harry H Jiang, Lauren Brown, Jessica Cheng, Mehtab Khan, Abhishek Gupta, Deja Workman, Alex Hanna, Johnathan Flowers, and Timnit Gebru. 2023. AI Art and its Impact on Artists. In *AAAI/ACM Conference on AI, Ethics, and Society*. 363–374. https://doi.org/10.1145/3600211.3604681
[40] Martin Jonsson and Jakob Tholander. 2022. Cracking the code: Co-coding with AI in creative programming education. In *Conference on Creativity and Cognition*. 5–14. https://doi.org/10.1145/3527927.3532801
[41] Majeed Kazemitabaar, Runlong Ye, Xiaoning Wang, Austin Z Henley, Paul Denny, Michelle Craig, and Tovi Grossman. 2024. CodeAid: Evaluating a Classroom Deployment of an LLM-based Programming Assistant that Balances Student and Educator Needs. (2024). https://doi.org/10.1145/3613904.3642773
[42] Caitlin Kelleher and Randy Pausch. 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM computing surveys* 37, 2 (2005), 83–137. https://doi.org/10.1145/1089733.1089734

[43] Mary Beth Kery and Brad A Myers. 2017. Exploring exploratory programming. In *IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE, 25–29. https://doi.org/10.1109/VLHCC.2017.8103446

[44] Robert J Kloss. 1994. A nudge is best: Helping students through the Perry scheme of intellectual development. *College Teaching* 42, 4 (1994), 151–158.

[45] Steve Krug. 2013. *Don't make me think, revisited* (3 ed.). New Riders Publishing, Upper Saddle River, NJ.

[46] Google Labs. 2024. Shiffbot. https://shiffbot.withgoogle.com/.

[47] Sam Lau and Philip Guo. 2023. From "Ban it till we understand it" to "Resistance is futile": How university programming instructors plan to adapt as more students use AI code generation and explanation tools such as ChatGPT and GitHub Copilot. In *ACM Conference on International Computing Education Research*. 106–121. https://doi.org/10.1145/3568813.3600138

[48] Sam Lavigne and Tega Brain. 2019. p5.Riso. https://antiboredom.github.io/p5.riso/.

[49] Golan Levin and Tega Brain. 2021. *Code as Creative Medium: A Handbook for Computational Art and Design*. MIT.

[50] Jingyi Li, Joel Brandt, Radomír Mech, Maneesh Agrawala, and Jennifer Jacobs. 2020. Supporting visual artists in programming through direct inspection and control of program execution. In *SIGCHI Conference on Human Factors in Computing Systems*. 1–12. https://doi.org/10.1145/3313831.3376765

[51] Jingyi Li, Sonia Hashim, and Jennifer Jacobs. 2021. What We Can Learn From Visual Artists About Software Development. In *SIGCHI Conference on Human Factors in Computing Systems*. https://doi.org/10.1145/3411764.3445682

[52] Jingyi Li, Eric Rawn, Jacob Ritchie, Jasper Tran O'Leary, and Sean Follmer. 2023. Beyond the Artifact: Power as a Lens for Creativity Support Tools. In *ACM Symposium on User Interface Software and Technology*. 1–15. https://doi.org/10.1145/3586183.3606831

[53] Alan Lundgard, Crystal Lee, and Arvind Satyanarayan. 2019. Sociotechnical considerations for accessible visualization design. In *Visualization Conference*. IEEE, 16–20. https://doi.org/10.1109/VISUAL.2019.8933762

[54] John Maeda. 2004. *Creative Code: Aesthetics + Computation*. Thames & Hudson.

[55] Mihaela Malita and Ethel Schuster. 2020. From Drawing to Coding: Teaching Programming with Processing. *Journal of Computing Sciences in Colleges* 35, 8 (April 2020), 245–246. https://doi.org/10.1145/3544548.3580683

[56] Aymeric Mansoux, Brendan Howell, Dušan Barok, and Ville-Matias Heikkilä. 2023. Permacomputing Aesthetics: Potential and Limits of Constraints in Computational Art, Design and Culture. In *Ninth Computing within Limits*. LIMITS. https://doi.org/10.21428/bf6fb269.6690fc2e https://limits.pubpub.org/pub/6loh1eqi.

[57] Lauren Lee McCarthy, Thomas Hughes, and Golan Levin. 2021. *Open Source Software Toolkits for the Arts (OSSTA): a Convening*. Technical Report. The Frank-Ratchye STUDIO for Creative Inquiry, Carnegie Mellon University. https://github.com/CreativeInquiry/OSSTA-Report.

[58] Bruce M McLaren, Krista E DeLeeuw, and Richard E Mayer. 2011. Polite web-based intelligent tutors: Can they improve learning in classrooms? *Computers & Education* 56, 3 (2011), 574–584. https://doi.org/10.1016/j.compedu.2010.09.019

[59] Alex McLean. [n. d.]. Tidal Cycles. https://tidalcycles.org/.

[60] Andrew M McNutt, Anton Outkine, and Ravi Chugh. 2023. A Study of Editor Features in a Creative Coding Classroom. In *CHI Conference on Human Factors in Computing Systems*. 1–15. https://doi.org/10.1145/3544548.3580683

[61] Mark C Mitchell and Oliver Bown. 2013. Towards a creativity support tool in processing: understanding the needs of creative coders. In *Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration*. 143–146. https://doi.org/10.1145/2541016.2541096

[62] Taru Muhonen and Raphaël de Courville. 2023. Creation technology database. https://available-anaconda-10d.notion.site/Creation-technology-database-053027df02ec49e8b3183571d3fcafca.

[63] netizen. 2022. netizen. https://netizen.org/netnet/.

[64] openFrameworks 2021. openFrameworks. https://openframeworks.cc/.

[65] OpenProcessing. [n. d.]. OpenProcessing. https://www.openprocessing.org/.

[66] p5 2021. p5.js. https://p5js.org/. Accessed 9/21/21.

[67] p5editor [n. d.]. p5.js editor. https://github.com/processing/p5.js-web-editor.

[68] p5.js editor developers. [n. d.]. Contributing to the p5.js Web Editor. https://github.com/processing/p5.js-web-editor/blob/develop/.github/CONTRIBUTING.md.

[69] pagespeed-pro. 2024. css-art.com. https://github.com/pagespeed-pro/css-art.com.

[70] Allison Parrish. 2016. pytracery. https://github.com/aparrish/pytracery.

[71] William Christopher Payne, Yoav Bergner, Mary Etta West, Carlie Charp, R Benjamin Benjamin Shapiro, Danielle Albers Szafir, Edd V Taylor, and Kayla DesPortes. 2021. Danceon: Culturally responsive creative computing. In *SIGCHI conference on human factors in computing systems*. 1–16. https://doi.org/10.1145/3411764.3445149

[72] William Christopher Payne, Xinran Shen, Eric Xu, Matthew Kaney, Maya Graves, Matthew Herrera, Madeline Mau, Diana Murray, Vinnie Wang, and Amy Hurst. 2023. Approaches to Making Live Code Accessible in a Mixed-Vision Music Ensemble. In *ACM SIGACCESS Conference on Computers and Accessibility*. 1–5. https://doi.org/10.1145/3597638.3614489

[73] Kylie Peppler and Yasmin Kafai. 2009. Creative Coding: Programming for Personal Expression. *International Conference on Computer Supported Collaborative Learning* 30 (2009), 7.

[74] Polygon. 2019. Art Sqool took me back to real-life art school. https://www.polygon.com/2019/3/1/18246354/art-sqool-impressions-pc-mac.

[75] Venkatesh Potluri, Sudheesh Singanamalla, Nussara Tieanklin, and Jennifer Mankoff. 2023. Notably Inaccessible—Data Driven Understanding of Data Science Notebook (In) Accessibility. In *ACM SIGACCESS Conference on Computers and Accessibility*. 1–19. https://doi.org/10.1145/3597638.3608417

[76] Processing. [n. d.]. p5.sound.js. https://github.com/processing/p5.sound.js.

[77] Miller Puckette. [n. d.]. Pure Data. https://puredata.info/.

[78] Yizhou Qian and James Lehman. 2017. Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education* 18, 1 (2017), 1–24. https://doi.org/10.1145/3077618

[79] Eric Rawn, Jingyi Li, Eric Paulos, and Sarah E Chasins. 2023. Understanding Version Control as Material Interaction with Quickpose. In *SIGCHI Conference on Human Factors in Computing Systems*. 1–18. https://doi.org/10.1145/3544548.3581394

[80] Casey Reas and Ben Fry. 2007. *Processing: a programming handbook for visual designers and artists*. Mit Press. https://doi.org/oclc/73993935

[81] Horst W Rittel and Melvin M Webber. 1974. Wicked problems. *Man-made Futures* 26, 1 (1974), 272–280.

[82] ruffle. 2024. Ruffle - Flash Emulator. https://ruffle.rs/.

[83] Adrian Salguero, Julian McAuley, Beth Simon, and Leo Porter. 2020. A Longitudinal Evaluation of a Best Practices CS1. In *Conference on International Computing Education Research*. https://doi.org/10.1145/3372782.3406274

[84] Vincent Scheib, Theo Engell-Nielsen, Saku Lehtinen, Eric Haines, and Phil Taylor. 2002. The demo scene. In *ACM SIGGRAPH 2002 conference abstracts and applications*. 96–97. https://doi.org/10.1145/1242073.1242125

[85] Ana Selvaraj, Eda Zhang, Leo Porter, and Adalbert Gerald Soosai Raj. 2021. Live Coding: A Review of the Literature. In *Conference on Innovation and Technology in Computer Science Education*. https://doi.org/10.1145/3430665.3456382

[86] David Williamson Shaffer and Mitchel Resnick. 1999. "Thick" Authenticity: New Media and Authentic Learning. *Journal of Interactive Learning Research* 10, 2 (December 1999), 195–215.

[87] Natalie Sherman. 2024. Software giant Adobe accused of 'trapping customers'. https://www.bbc.com/news/articles/c98825z8330o. *BBC News* (June 2024).

[88] Daniel Shiffman. 2024. The Coding Train. https://www.youtube.com/user/shiffman.

[89] Daniel Shiffman. 2024. *The Nature of Code*. No Starch Press.

[90] Ben Shneiderman. 2007. Creativity support tools: accelerating discovery and innovation. *Commun. ACM* 50, 12 (2007), 20–32. https://doi.org/10.1145/1323688.1323689

[91] Beth Simon, Päivi Kinnunen, Leo Porter, and Dov Zazkis. 2010. Experience Report: CS1 for Majors with Media Computation. In *Conference on Innovation and Technology in Computer Science Education*. https://doi.org/10.1145/2445196.2445214

[92] Eric Snodgrass and Winnie Soon. 2019. API practices and paradigms: Exploring the protocological parameters of APIs as key facilitators of sociotechnical forms of exchange. *First Monday* 24, 2 (2019). https://doi.org/10.5210/fm.v24i2.9553

[93] Sarah Sterman, Molly Jane Nicholas, and Eric Paulos. 2022. Towards Creative Version Control. *Proceedings of the ACM on Human-Computer Interaction* 6, CSCW2 (2022), 1–25. https://doi.org/10.1145/3555756

[94] Blair Subbaraman and Nadya Peek. 2022. P5.Fab: Direct control of digital fabrication machines from a creative coding environment. In *ACM Designing Interactive Systems Conference*. 1148–1161. https://doi.org/10.1145/3532106.3533496

[95] Blair Subbaraman, Shenna Shim, and Nadya Peek. 2023. Forking a Sketch: How the OpenProcessing Community Uses Remixing to Collect, Annotate, Tune, and Extend Creative Code. In *ACM Designing Interactive Systems Conference*. 326–342. https://doi.org/10.1145/3563657.3595969

[96] Steven L. Tanimoto. 2013. A perspective on the evolution of live programming. In *Workshop on Live Programming, LIVE*. IEEE, 31–34. https://doi.org/10.1109/LIVE.2013.6617346

[97] Teresa Terroso and Mário Pinto. 2022. Programming for Non-Programmers: An Approach Using Creative Coding in Higher Education. In *International Computer Programming Education Conference (Open Access Series in Informatics, Vol. 102)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 13:1–13:8. https://doi.org/10.4230/OASIcs.ICPEC.2022.13

[98] Tiffany Tseng, Ruijia Cheng, and Jeffrey Nichols. 2024. Keyframer: Empowering Animation Design using Large Language Models. *arXiv preprint arXiv:2402.06071* (2024). https://doi.org/10.48550/arXiv.2402.06071

[99] Roopa Vasudevan. 2023. High-Level Creativity: New Media Art and the Priorities of the Tech Industry. (2023). Digital Democracies Institute Summer Speaker Series, https://digitaldemocracies.org/dr-roopa-vasudevan-high-level-creativity-new-media-art/.

[100] Mauricio Verano Merino and Juan Pablo Sáenz. 2023. The Art of Creating Code-Based Artworks. In *SIGCHI Conference on Human Factors in Computing Systems Extended Abstracts*. 1–7. https://doi.org/10.1145/3544549.3585743

[101] Lasse Steenbock Vestergaard, João Fernandes, and Mirko Presser. 2017. Creative coding within the Internet of Things. In *Global Internet of Things Summit*. IEEE, 1–6. https://doi.org/10.1109/GIOTS.2017.8016223

[102] Jessica Vitak. 2024. Reviewer Critiques (Qualitative Methods) and How to Respond to Them. https://web.archive.org/web/20240524105030/https://docs.google.com/document/d/1jHiWJdkjm6Go683GIxi0tz8l-17rQQpadn9qb7zZDh4/edit#heading=h.e86h69sdez6d.

[103] Anqi Wang, Zhizhuo Yin, Yulu Hu, Yuanyuan Mao, and Pan Hui. 2024. Exploring the Potential of Large Language Models in Artistic Creation: Collaboration and Reflection on Creative Programming. *arXiv preprint arXiv:2402.09750* (2024). https://doi.org/10.48550/arXiv.2402.09750

[104] David Weintrop and Uri Wilensky. 2015. To Block or Not to Block, That is the Question: Students' Perceptions of Blocks-Based Programming. In *International Conference on Interaction Design and Children*. https://doi.org/10.1145/2771839.2771860

[105] Timothy J Weston, Elaine Seymour, Andrew K Koch, and Brent M Drake. 2019. Weed-out classes and their consequences. *Talking about leaving revisited: Persistence, relocation, and loss in undergraduate STEM education* (2019), 197–243. https://doi.org/10.1007/978-3-030-25304-2_7

[106] Jo Wood, Alexander Kachkaev, and Jason Dykes. 2018. Design exposition with literate visualization. *IEEE transactions on visualization and computer graphics* 25, 1 (2018), 759–768. https://doi.org/10.1109/TVCG.2018.2864836

[107] Zoe J Wood, Paul Muhl, and Katelyn Hicks. 2016. Computational Art: Introducing High School Students to Computing via Art. In *ACM Technical Symposium on Computing Science Education*. 261–266. https://doi.org/10.1145/2839509.2844614

[108] Junran Yang, Andrew M McNutt, and Leilani Battle. 2024. Considering Visualization Example Galleries. In *Symposium on Visual Languages and Human-Centric Computing*. IEEE, 329–343. https://doi.org/10.1109/VL/HCC60511.2024.00043

A Appendix

In this appendix, we include several additional materials. First, Sec. A.1 contains the interview guide for our semi-structured interviews. Second, Sec. A.2 contains our post-interview survey instrument. Lastly, Sec. A.3 summarizes the code book from our analysis of the interviews.

A.1 Semi-Structured Interview Guide

Thank you for agreeing to participate in this interview. My name is Andrew McNutt. Today we will spend around 60 minutes talking about your experiences with, and opinions about, several topics related to creative coding. Tomorrow, I'll send you a brief exit survey, which should take a handful of minutes, and will give you a place to tell us where to send your interview compensation.

Do you have any questions for us before we start recording the interview?

Okay, let's begin! We'll start recording now. <START RECORDING>

Let's start with several Introductory Questions.

(1) Do you think of yourself as an artist? A teacher? A programming tool builder? How do those roles manifest themselves for you?
(2) Next I have a few quick questions about your experience teaching. Have you taught creative coding? This could be in a classroom setting, or in a workshop, or by writing or recording tutorials, or something else. (If not, skip to Question 6. And ask questions in a way that reflects that background having not taught.)
    (a) What are the goals of your teaching (in teaching creative coding)?
    (b) What types of things do people make in this course? Visual art?
    (c) What does "good" creative code look like? Does that mean beautiful code? Beautiful output? Both?
    (d) What types of students enroll in the course?
(3) Do you grade or give feedback on student work?
    (a) How does grading work? (What are you looking to grade? Eg code correctness/ artistic ness etc)
    (b) Are you satisfied with how grading works? How do you wish it was different?
(4) What is the role of community in your classroom? How do students help each other on course work?
(5) What are some challenges you see students encounter in the classroom? What obstacles stand in the way of successful outcomes we talked about earlier?
    (a) Does a student's background (for example, being interested primarily in art and design rather than computer science) impact the type of challenges that they face? Are there specific topics or projects students with different backgrounds tend to struggle with?
    (b) Do you think any particular technological interventions might help with those challenges?

This brings us to our next section. We've talked about several aspects of your experience teaching creative coding. Next, let's talk more about the Technical Details of those experiences.

(6) What technologies—including programming languages, libraries, tools, and editors—have you used to teach? AND What have you liked and disliked about those tools?
    (a) What motivated you to choose these technologies and tools?
    (b) Are the tools you use to create course material different from those that students use to complete course work? Do you wish that was different?
(7) Have you customized any of these technologies — languages, libraries, editors, or other tools? Or have you made your own?
    (a) What motivated you to make custom tools? What did you do?
    (b) What types of tasks did you seek to improve via these custom tools? For instance, expressivity, learning goals, reducing syntactic errors.
    (c) Do you think of your system(s) as primarily a teaching system or an art making system?
(8) How does automated guidance work in your classroom / your tools? Like, how are students able to get help with your tools? Was their documentation or? Are you familiar with linters or auto-formatters? Do you think they have value in this environment? One reason I bring it up is because in our own courses we found that linters were often well liked and viewed as helpful. How well does that align with your experience?
(9) Are there other sorts of editor features (or tools) that you wish existed that would help you teach creative coding?

(10) What creative coding classroom tasks are currently hard or impossible? Like, in the absence of constraints (such as from technological, temporal, curricular, or financial sources) what would you change about your creative coding course?

Alright us to our next section, where we'll spend some time thinking about Particular Code Editing Features.

(11) In your teaching, do you do live coding? What does that process look like? What tools do you use?

(12) What is the role of liveness in your systems? Do any of the tools you use have auto-refresh or anything like that?

(13) What is your perspective on the role of AI-based tools in a creative coding classroom? For instance, tools like GitHub CoPilot or ChatGPT, which can generate code snippets? In an ideal world, what types of things could a digital assistant help with in a code editor for creative coding?

(14) What do you think about the role of pedagogical software in classroom settings versus so-called "real" software engineering tools? More provocatively: is it appropriate to teach classes using tools that are pedagogically valuable, but that the student will never use again?

(15) We sometimes saw students shying away from more advanced tools that they perceive as impeding their learning because they make things too easy. Other students perceived a difference between an "art tool" versus a "making art with code tool". What do you think about this tension between more feature-rich creative coding environments and their potential effects for students?

We've been talking for a while, so we'll start to wrap up with a few Closing Questions.

(16) We talked a lot about creative coding. Let's pop up a level. Can you reflect on what the term itself, "creative coding," means to you?

  (a) Is creative coding a way to make art with code, or is it a way to learn coding via art?

  (b) How do you know when some creative coding work is finished?

(17) Is there anyone you think I should ask about these topics?

(18) Do you have any other comments, or anything else you'd like us to know?

That was all really helpful. Thank you so much for sharing your time and thoughts with us! Now we'll stop the recording and talk about how to fill out the brief post-interview survey.

A.2 Post-Interview Survey

Thanks for completing our interview! We just have a few other questions for you.

(1) Do you consider yourself... **Matrix multiple choice**

  (a) An Artist: ○ Strongly Agree, ○ Agree, ○ Neutral, ○ Disagree, ○ Strongly Disagree

  (b) A Teacher: ○ Strongly Agree, ○ Agree, ○ Neutral, ○ Disagree, ○ Strongly Disagree

  (c) A Tool Builder: ○ Strongly Agree, ○ Agree, ○ Neutral, ○ Disagree, ○ Strongly Disagree

(2) Is there any other label (besides the ones listed above) that you most identify with? **Sentence response**

(3) Is there anything that you've made that you would like to show us? This might be a tool you've made, materials you've used to teach creative coding, a programmatic environment you've used, or something else entirely. Please include links if possible (feel free to use multiple lines). **Paragraph response**

(4) How long have you been teaching? **Multiple choice**: ○ 0-1 Years, ○ 1-3 Years, ○ 3-6 Years, ○ 6-10 Years, ○ More than 10 Years

(5) What is the highest academic degree you have attained (or are pursuing)? What field is it in? **Sentence response**

(6) What pronouns should we use to refer to you in analysis of these interviews? **Multiple choice**: ○ She / hers, ○ He / his, ○ They / them, ○ Other..., ○ Add option

(7) When quoting any of your responses in our analysis, do you prefer to be quoted anonymously? Or would you like quotations of your responses to be attributed by name? **Multiple choice**: ○ Refer to me anonymously, ○ Refer to me by name if possible

(8) Is it alright if we contact you with follow up questions, should they arise? **Multiple choice**: ○ Yes, ○ no

(9) Is there anything else you'd like us to know that you didn't get a chance to talk about during the interview? **Paragraph response**

(10) Is there anyone else you think we should talk to? **Paragraph response**

(11) (for record keeping) What is your name? **Sentence response**

(12) What email address should we send your gift certificate to? **Sentence response**

A.3 Code Book: Analysis of Interviews

Code	Explanation (e.g. *Relating to X*)
SLOWNESS	
Art/Craft	The difference between art and craft, particularly in the context of learning each of the components
Authenticity	The perception of tools being "realistic" or "practical"
Constraints	Impositions on students to cause them to produce art in a useful or different manner
Directorial Agency	Expressing an idea without needing to go through the craft of making it, analogous to declarative programming
Friction	Tedious or difficult processes
Live Coding	Liveness (per Tanimoto [96]) or live coding
Process	Demonstrating, conveying, or teaching activities involving processes. Like "getting hands in clay" to teach pottery
Reflection	Reflection, pausing, or considering the context, content, or scope of work being pursued
Slowness	Activities or processes that involve taking more time in order to benefit various situations or pursuits
POLITICS	
Bit Rot	The tendency for digital objects to become broken over time
Documentation	Documentation of software
Evaluation	Grading or feedback
Physical Computing	Creative coding applications and contexts that involve hardware, such as microcontrollers or Arduinos
Politics	Issues around power, relationships between individuals, corporations, or institutions
JOY	
Inclusion	Being part of a community, or being included or welcomed
Joy	Enjoyment, play, delight, or otherwise ludic experience
Process	(also in SLOWNESS; see above)
TOPIC: ACCESSIBILITY	
Accessibility	Accessibility broadly defined, such as physical disability, or being left out through choices that alienate a culture
TOPIC: AI	
AI Concerns	Commentary or concerns about systems generally referred to as AI, such as LLMs, text2images generators, etc.
Mediocrity	Things that are boring, uninteresting, or middling, particularly the production of work with such qualities
N/A	
Alternate Pedagogy	Different ways in which the classroom might be reorganized to better fit different educational goals
Browser	The browser, such as being a platform for tools
Feature Ideation	Ideas or suggestions on how things could be different without a themed alternate reason for that suggestion
JS Difficulty	Difficulties or issues with JavaScript
Meaning of Creative Coding	Definitions and meanings of the term creative coding
Role of Teacher	The structure and function of pedagogy
Software Ecosystem	Environment of tools available and how they relate to one another
Tool Pedagogy	Means by which tools can achieve pedagogical goals
Two Kinds of Student	Frequent occurrence in creative coding classrooms: students who are good either at programming or art, but not both

Figure 4: The final code book from our analysis of the interview transcripts. Each code is presented with the eventual theme that it is connected with. A number of codes were identified as being redundant with prior work, particularly those related to pedagogical topics (such as "Two Kinds of Student" being well captured by Levin and Brain [49]), and are thus omitted from our chosen themes. In addition to our main themes, we also highlight those codes that led to consideration of specific topics, such as AI and accessibility.