Reframe: Rethinking Versioning as a Creative Practice in Coding

Reframe: Rethinking Versioning as a Creative Practice in Coding

Juan Pablo Sáenz
Politecnico di Torino
Turin, Italy
juan.saenz@polito.it

Stefano Di Leo
Politecnico di Torino
Turin, Italy
stefano.dileo@polito.it

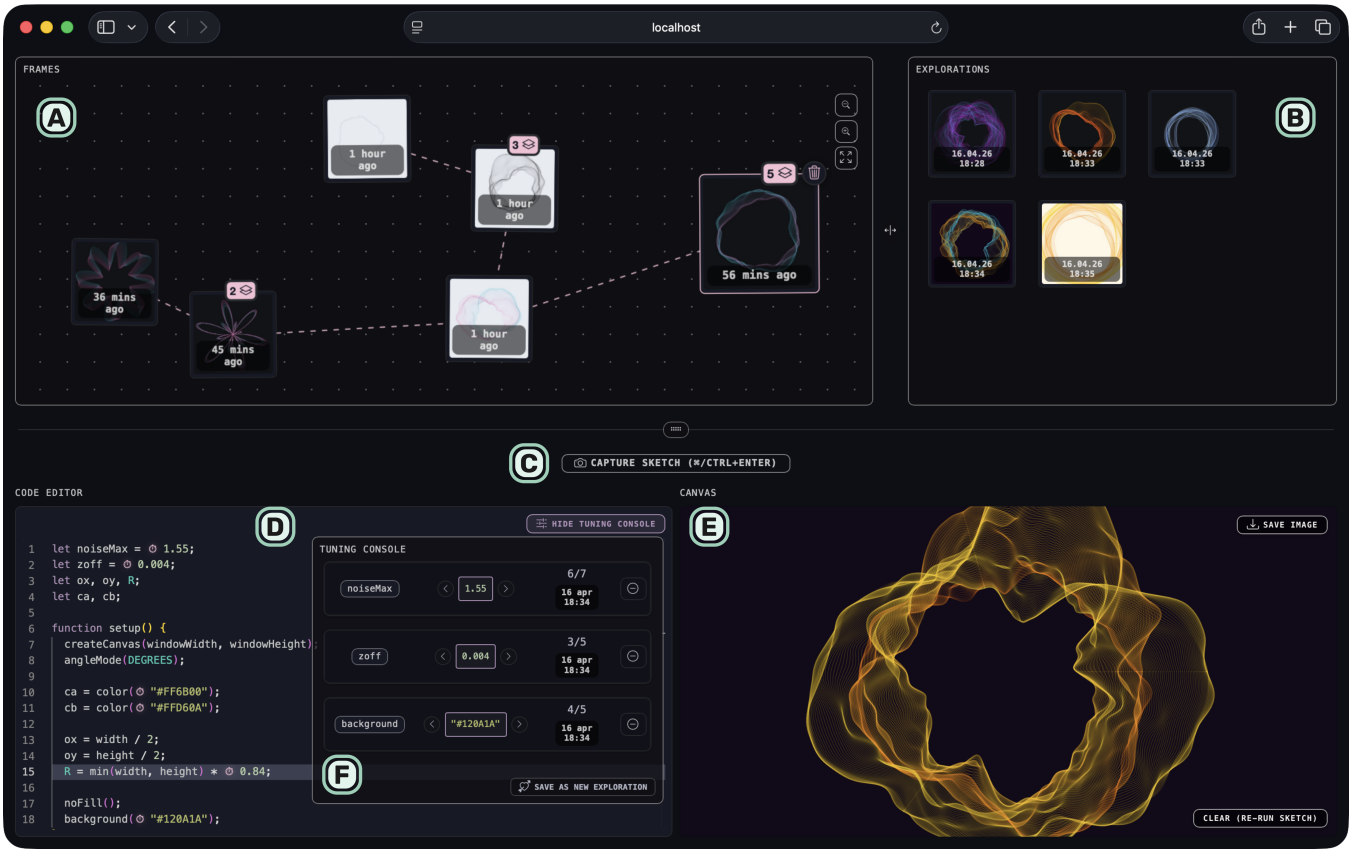

Figure 1: This article presents *Reframe*, a web-based Creative Coding environment for p5.js that rethinks versioning as a creative practice. The interface brings together non-linear navigation across frames and explorations, direct code editing, parameter-level exploration, and continuous visual feedback, supporting users in revisiting, comparing, and further developing evolving directions of a code-based artwork.

Abstract

Creative Coding is characterized by iterative and exploratory workflows oriented toward open-ended outcomes. Practitioners make small adjustments to parameter values and code structure, where even minor edits can produce drastically different visual results. Yet existing tools offer limited means for tracking this process, and creative coders rely on time-consuming strategies to save, inspect, compare, and revisit prior versions. More importantly, this is not only a practical limitation but also a conceptual mismatch: the tools used to manage versions can shape and potentially constrain the creative process itself. This article derives design considerations and presents Reframe, a web-based Creative Coding environment for p5.js that seeks to support creative work by rethinking versioning. By distinguishing between structural and value-level change and combining non-linear navigation, direct code editing, parameter-level exploration, and continuous visual feedback, Reframe is designed to support divergence, reflection, and continued iteration while keeping source code as the creative medium.

C&C '26, London, United Kingdom

ACM ISBN 979-8-4007-2583-8/26/07
https://doi.org/10.1145/3803784.3816814

CCS Concepts

• **Human-centered computing** → **Interactive systems and tools**; • **Software and its engineering** → *Development frameworks and environments.*

Keywords

creative coding, creativity support tools, creative version control, exploratory programming, programming environments

ACM Reference Format:
Juan Pablo Sáenz and Stefano Di Leo. 2026. Reframe: Rethinking Versioning as a Creative Practice in Coding. In *Creativity and Cognition (C&C '26), July 13–16, 2026, London, United Kingdom.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3803784.3816814

1 Introduction and Background

Creative Coding (CC) refers to the practice of using programming as a medium for artistic expression and creative exploration [11, 16, 25]. It is characterized by iterative workflows oriented toward open-ended outcomes, where coding does not simply implement a predefined specification but actively participates in the discovery of ideas, forms, and directions [15]. In this sense, CC aligns with what Beth *et al.* [2] describe as *exploratory programming*: a mode of programming in which code is used as a medium to experiment with ideas rather than to implement a predetermined solution [22].

In practice, this exploratory nature translates into workflows shaped by continual reuse, small adjustments, and situated experimentation. CC practitioners frequently work by modifying existing code-based works, or *sketches*, through parameter tuning and successive variations rather than composing each sketch from scratch [30]. In these workflows, even minor edits to parameter values or code structure can produce drastically different visual outcomes, making iteration unfold through multiple partial and alternative states of a sketch.

Existing tools offer limited support for saving, inspecting, comparing, and revisiting such states in meaningful ways, often leading practitioners to rely on manual and time-consuming strategies [4, 13, 21, 33]. More importantly, this is not only a practical limitation but also a conceptual mismatch: traditional version control systems are grounded in models of linear progression and stable revision histories that do not align with the exploratory and iterative nature of CC [20, 32]. Indeed, prior work has argued that versioning in creative contexts must be understood differently from linear software development, as the tools used to manage versions can shape—and potentially constrain—how creative work is explored, interpreted, and revisited over time [23, 29].

Reframing this problem requires approaching versioning not simply as a mechanism for storing and retrieving prior states, but as a design space for supporting creative practice. In this sense, versioning in CC should be understood within the broader context of Creativity Support Tools, as a question of how interactive systems might better support exploration, discovery, and the development of ideas over time [9, 14, 27]. More specifically, the notion of Creative Version Control Systems proposed by Sterman *et al.* provides a useful lens for understanding versioning in creative domains as a practice shaped by divergence, reuse, reflection, and alternative directions rather than by linear progression alone [29].

This perspective informed a set of design decisions aimed at aligning versioning with the exploratory character of CC. Rather than treating iteration as a linear sequence of revisions, the system is designed to support divergence, to distinguish between opening new directions and exploring within them, to foreground visual reasoning as part of the creative process, and to support exploration through direct engagement with code. Taken together, these decisions articulate an approach to versioning that aims to better support divergence, reflection, and exploration in CC. Against this background, we present *Reframe*, a web-based CC environment for p5.js [17] (a JavaScript library for CC inspired by Processing [7], reinterpreted for the web, and a central reference in CC practice) that embodies these design decisions.

2 Design Considerations

The design of Reframe is shaped by a set of design considerations that emerge from tensions in CC practice and from the need to rethink versioning beyond linear models of revision. We discuss these considerations below.

Supporting divergence rather than linear progression. A first design consideration was to avoid representing iteration as a single linear history. In CC, returning to a prior state does not necessarily mean reverting progress or recovering an earlier "correct" version; it can instead serve as a point of departure for a different creative direction. Representing version histories as a single sequence risks imposing a model of progress that is poorly aligned with the exploratory nature of the practice [23]. Reframe was therefore designed to support divergence as an ordinary part of iteration, allowing prior states to remain available not only as records of where a sketch has been, but as points from which new trajectories can emerge [29, 30].

Distinguishing new directions from exploration within them. A second design consideration was that not all changes in a sketch play the same role in the creative process. In CC, some changes introduce a new structural direction, while others explore possibilities within an existing one through parameter tuning and localized adjustments. Treating both as equivalent revisions risks flattening the exploratory character of iteration, obscuring how a sketch develops over time, and limiting opportunities for reflection and comparison across alternative directions [13, 29, 30]. Reframe was therefore designed to distinguish between opening a new direction and exploring within it, allowing different forms of change to be organized and revisited in ways that better reflect their role in practice.

Designing for visual reasoning during iteration. A third design consideration was that, in CC, visual output is not merely a final result but an active part of the iterative process. Practitioners do not only modify code; they also interpret, compare, and evaluate its visual consequences as they work. Prior work on creative versioning suggests that earlier states can function as a *palette* of materials for reflection, comparison, and reuse rather than as superseded steps in a linear progression [29]. Likewise, research on programming tools for visual artists shows that practitioners often reason about code by inspecting its visible effects as they work [12]. Supporting iteration therefore requires more than preserving code

states alone: it also requires treating visual outcomes as resources for reasoning, reflection, and comparison throughout the process.

Supporting exploration through direct engagement with code. A fourth design consideration was that supporting exploration in CC should not require displacing the user's engagement with code as the primary creative material. In this context, automating variation or proposing alternatives too early risks narrowing the space of possibilities before practitioners can explore and interpret them for themselves. Since even small modifications can lead to markedly different visual outcomes, seemingly minor changes may open unexpected creative directions that are difficult to anticipate in advance [30, 32]. A versioning tool should therefore support exploration not by generating or selecting alternatives on behalf of the practitioner, but by making the ongoing process more visible and revisitable while keeping creative action grounded in direct code manipulation.

3 System Overview

Figure 1 shows Reframe's interface. We present it here in terms of how its main regions support the design decisions outlined above and organize iterative work in CC.

A flexible workspace for history and action. Reframe organizes the workspace into two tightly coupled regions that support complementary aspects of iterative work in CC. The upper region is dedicated to navigating the evolving history of a sketch through *Frames* (Fig. 1A) and *Explorations* (Fig. 1B), while the lower region supports direct engagement with code and visual output through the *Code Editor* (Fig. 1D) and *Canvas* (Fig. 1E). A horizontal divider with a draggable handle between both regions allows users to redistribute space and shift attention between historical navigation and active work on the sketch as needed. This flexibility is important because visual representations play a central role across the system. Keeping prior frames and explorations continuously available together with their corresponding code and visual output requires giving users control over how much space is devoted to each view, so they can move across previous states while preserving a clear and consistent view of the sketch they are currently inspecting or developing.

Navigating frames and explorations. Within the upper region, Reframe distinguishes between *Frames* (Fig. 1A) and *Explorations* (Fig. 1B). Frames represent structurally distinct states of a sketch and are displayed as connected nodes that trace successive creative directions over time, while explorations capture value-level developments within a selected frame. A vertical divider between both panels allows users to redistribute space according to the density of frames or explorations they need to inspect. The Frames panel also supports direct spatial navigation, allowing users to drag nodes, pan across the workspace, and use zoom and recenter controls to keep growing nodal structures legible.

New states are created through the *Capture Sketch* action (Fig. 1C): when the current sketch is captured, Reframe determines whether it should be registered as a new frame or as an exploration, depending on whether the detected change is structural or value-level; this distinction is illustrated later in the next section through a simplified code example (Fig. 2). In both cases, captured states are represented through visual thumbnails of the canvas, allowing users to navigate the evolving history of a sketch through visual thumbnails coordinated with the corresponding code state. Clicking the visual thumbnail of either a frame or an exploration updates the lower workspace to display the corresponding code state and visual output, both of which remain directly editable. As a result, any previously captured state can become a new point of departure for further work, allowing the nodal structure to grow from multiple locations rather than unfolding as a single linear progression.

Exploring through code, tuning, and visual feedback. The lower region of Reframe combines the *Code Editor* (Fig. 1D) with the *Canvas* (Fig. 1E), keeping code manipulation and visual feedback tightly coupled throughout the process. The editor remains the primary site of creative action, while the *Tuning Console* (Fig. 1F) supports exploration within the currently selected frame by allowing users to add variables directly from the editor through the ⏱ icon displayed next to variables with previously explored values. Once added to the console, these variables can be adjusted individually or in combination, drawing on values from prior explorations associated with the active frame. In fact, the sketch shown in the canvas in Fig. 1 does not correspond to a previously captured exploration, but to a new configuration produced through the recombination of three variables in the Tuning Console. When such a configuration becomes relevant, it can be saved directly from the Tuning Console as a new exploration.

Across these interactions, the *Canvas* (Fig. 1E) remains the shared visual surface through which prior states are revisited and ongoing changes are interpreted. Whether users edit code, navigate frames and explorations, or adjust variables through the *Tuning Console*, the current visual state provides a continuous basis for comparison and further development. The canvas area also includes lightweight controls for saving the current visual state as an image and for rerunning the sketch from scratch when needed.

4 Implementation notes

Reframe is implemented as a web-based environment for p5.js, using React [28] and TypeScript [19] for the interface and application logic. Sketches are edited as JavaScript code and executed directly in the browser, keeping the code editor and canvas tightly coupled so that changes in code are immediately reflected in the visual output. When a sketch is captured, Reframe stores the corresponding code state, a thumbnail of the rendered canvas, a timestamp, and its relation to the currently selected frame, using IndexedDB [5] for local persistence and Yjs [10] for managing synchronized document state. This provides the technical basis for revisiting and navigating frames and explorations while keeping the system self-contained within the browser.

A central implementation requirement was to determine whether a captured state should be registered as a new frame or as an exploration within the current frame. To support this, Reframe compares the current code state against the relevant previously captured state and parses both into abstract syntax trees (ASTs) using Tree-sitter [31]. Structural differences between these ASTs are then computed with GumTree [6], allowing the system to distinguish structural edits from value-level changes. Reframe registers the former as new frames and the latter as explorations within the active

```
let radius = 120;
let step = 20;
let strokeAlpha = 80;

function draw() {
  background(240);
  stroke(0, strokeAlpha);
  translate(width / 2, height / 2);
  beginShape();
  for (let a = 0; a < 360; a += step) {
    let x = radius * cos(a);
    let y = radius * sin(a);
    vertex(x, y);
  }
  endShape(CLOSE);
}
```

(a) Base sketch

```
let radius = 120;
let step = 20;
let strokeAlpha = 80;

function draw() {
  background(240);
  stroke(0, strokeAlpha);
  translate(width / 2, height / 2);
  push();
  rotate(frameCount * 0.01);
  scale(0.9);
  beginShape();
  for (let a = 0; a < 360; a += step) {
    let x = radius * cos(a);
    let y = radius * sin(a);
    vertex(x, y);
  }
  endShape(CLOSE);
  pop();
}
```

(b) Structural change from the base sketch

```
let radius = 160;
let step = 12;
let strokeAlpha = 140;

function draw() {
  background(240);
  stroke(0, strokeAlpha);
  translate(width / 2, height / 2);
  push();
  rotate(frameCount * 0.01);
  scale(1.1);
  beginShape();
  for (let a = 0; a < 360; a += step) {
    let x = radius * cos(a);
    let y = radius * sin(a);
    vertex(x, y);
  }
  endShape(CLOSE);
  pop();
}
```

(c) Value-level change from the modified sketch

Figure 2: Simplified p5.js examples showing how Reframe distinguishes between structural changes to a sketch's code and value-level modifications within an existing structure. The former are registered as new frames, while the latter are registered as explorations.

frame. Figure 2 illustrates this distinction with a simplified example. Importantly, this classification is not intended to infer the user's creative intention or to measure the visual magnitude of a change. Instead, it provides an operational distinction for organizing captured states: structural edits open a new direction, while value-level edits remain available for exploration within that direction.

5 Related Work and Future Directions

Prior work in CC and related domains has explored alternative ways of supporting versioning and iterative exploration. In broader exploratory programming, *Juxtapose*, a system for creating and comparing parallel interface alternatives, supports runtime parameter tuning and visual comparison [8], while Mikami *et al.*'s micro-versioning tool, an in-editor versioning tool for exploratory programming, supports lightweight capture and revisitation of intermediate program states [20]. *Reframe* shares with these systems a concern for making intermediate states available during exploration, but differs in treating version history as the organizing structure for CC practice: structural changes become *Frames*, value-level changes become *Explorations*, and both remain tied to visual output as editable points of departure.

Within CC more specifically, *Quickpose* is a version control system for Processing designed for creative exploration through non-linear navigation and visual manipulation of prior program states [24]. *Reframe* shares this concern with creative versioning, but distinguishes between *Frames* and *Explorations*, allowing structural developments to remain identifiable as distinct directions while value-level modifications become material for further recombination through the *Tuning Console*. This allows captured changes to support new and sometimes unforeseen explorations.

A different approach to exploratory support in p5.js appears in *Perpend*, which supports exploration by generating possible next edits together with their outputs at the cursor location [3]. Our proposal, instead, leaves users free to manipulate code directly, while supporting exploration through the *Tuning Console* in a way that remains grounded in the user's own process, since it recombines values drawn from prior explorations produced by the user over time, rather than orienting exploration around system-generated next steps.

This shift becomes even more pronounced in *Spellburst*, a node-based CC environment that integrates prompt-based mechanisms for generating and transforming code [1]. The design of *Reframe*, by contrast, keeps exploration anchored in user-authored code changes, supporting revisitation and development of prior states rather than generating new ones through prompts. In this sense, our design decisions privilege a slower and more user-driven relation to exploratory work, aligned with accounts of CC practice that emphasize reflection, direct engagement, and values distinct from those of conventional software development [13, 18, 32].

Beyond these closer precedents, *Stamper* is an artboard-oriented CC environment for p5.js that responds to the limitations of showing only a single graphics frame per program [4]. *Reframe* addresses a related concern through versioning, by keeping structural and value-level states visible, selectable, and editable, and turning sketch history itself into a space for continued work. Across these contrasts, *Reframe* maintains source code as the primary site of creative action, while treating prior states as resources for revisitation and further development rather than as generated alternatives or detached representations.

A natural next step for this work is to study how CC practitioners appropriate *Reframe* in their own workflows. Such studies could

examine when and why users capture states, how they move between *Frames* and *Explorations*, how they interpret the distinction between structural and value-level change, and how the *Tuning Console* supports parameter-level exploration in practice. More broadly, this would help clarify which aspects of the system are most meaningful for revisitation, comparison, recovery of prior directions, and continued development of sketches over time. This would also align with broader discussions in CST research, which argue that evaluation should be grounded in the specific aims of the tool and in the practices of its intended users [26].

Future work could also extend the current model beyond value-level exploration as it is currently implemented, for instance by accounting for recurring patterns involving control flow, data structures, or other higher-level program constructs. Another promising direction is to explore collaborative workflows in which captured states can be shared, annotated, compared, or developed across multiple authors. Together, these directions would help assess how far the current design can be extended while preserving *Reframe*'s central commitment to keeping exploratory work closely tied to source code, history, and visual output.

6 Conclusion

This article presented *Reframe*, a web-based CC environment for p5.js that aims to approach versioning not simply as a mechanism for storing and retrieving prior states, but as a design space for supporting creative practice. In contrast to linear models of revision inherited from conventional software development, *Reframe* treats versioning as part of how creative work is explored, interpreted, and revisited over time. By distinguishing between *Frames* and *Explorations*, and by combining non-linear navigation, direct code editing, parameter-level exploration, and continuous visual feedback in a single workspace, the system is designed to support divergence, reflection, and continued development without displacing engagement with source code as the primary creative medium. In this sense, *Reframe* contributes to ongoing discussions on Creativity Support Tools by showing how versioning itself can be rethought as a creative practice in CC.

References

[1] 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 *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology* (San Francisco, CA, USA) *(UIST '23)*. Association for Computing Machinery, New York, NY, USA, Article 100, 22 pages. doi:10.1145/3586183.3606719

[2] Mary Beth Kery and Brad A. Myers. 2017. Exploring exploratory programming. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 25–29. doi:10.1109/VLHCC.2017.8103446

[3] Angela Bi, Eric Rawn, Justin Lubin, and Sarah E. Chasins. 2026. Programming By Scaffolded Demonstration with Perpend. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems (CHI '26)*. Association for Computing Machinery, New York, NY, USA, Article 891, 20 pages. doi:10.1145/3772318.3791327

[4] Cameron Burgess, Dan Lockton, Maayan Albert, and Daniel Cardoso Llach. 2020. Stamper: An Artboard-Oriented Creative Coding Environment. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI EA '20)*. Association for Computing Machinery, New York, NY, USA, 1–9. doi:10.1145/3334480.3382994

[5] Mozilla Contributors. [n. d.]. IndexedDB API. https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API. Accessed: 16-04-2026.

[6] Jean-Rémy Falleri et al. 2014. GumTreeDiff/gumtree: An awesome code differencing tool. https://github.com/GumTreeDiff/gumtree. Accessed: 16-04-2026.

[7] Ira Greenberg. 2016. *Processing: Creative Coding and Computational Art*. Apress.

[8] Björn Hartmann, Loren Yu, Abel Allison, Yeonsoo Yang, and Scott R. Klemmer. 2008. Design as exploration: creating interface alternatives through parallel authoring and runtime tuning. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology* (Monterey, CA, USA) *(UIST '08)*. Association for Computing Machinery, New York, NY, USA, 91–100. doi:10.1145/1449715.1449732

[9] Stacy Hsueh, Marianela Ciolfi Felice, Sarah Fdili Alaoui, and Wendy E. Mackay. 2024. What Counts as 'Creative' Work? Articulating Four Epistemic Positions in Creativity-Oriented HCI Research. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 497, 15 pages. doi:10.1145/3613904.3642854

[10] Kevin Jahn et al. 2017. Introduction | Yjs Docs. https://docs.yjs.dev. Accessed: 16-04-2026.

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

[12] 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 *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–12. doi:10.1145/3313831.3376765

[13] Jingyi Li, Sonia Hashim, and Jennifer Jacobs. 2021. What We Can Learn From Visual Artists About Software Development. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 314, 14 pages. doi:10.1145/3411764.3445682

[14] 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 *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology* (San Francisco, CA, USA) *(UIST '23)*. Association for Computing Machinery, New York, NY, USA, Article 47, 15 pages. doi:10.1145/3586183.3606831

[15] John Maeda. 2004. *Creative Code: Aesthetics + Computation*. Thames & Hudson, New York, NY, USA.

[16] John Maeda and Paola Antonelli. 2001. *Design by Numbers*. MIT Press, Cambridge, MA, USA.

[17] Lauren McCarthy, Casey Reas, and Ben Fry. 2015. *Getting Started with p5.js: Making Interactive Graphics in JavaScript and Processing*. Make Community, LLC.

[18] Andrew M McNutt, Sam Cohen, and Ravi Chugh. 2025. Slowness, Politics, and Joy: Values That Guide Technology Choices in Creative Coding Classrooms. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery, New York, NY, USA, Article 54, 16 pages. doi:10.1145/3706598.3713472

[19] Microsoft. [n. d.]. TypeScript. https://www.typescriptlang.org. Accessed: 16-04-2026.

[20] Hiroaki Mikami, Daisuke Sakamoto, and Takeo Igarashi. 2017. Micro-Versioning Tool to Support Experimentation in Exploratory Programming *(CHI '17)*. Association for Computing Machinery, New York, NY, USA, 6208–6219. doi:10.1145/3025453.3025597

[21] Mark C. Mitchell and Oliver Bown. 2013. Towards a creativity support tool in processing: understanding the needs of creative coders. In *Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration* (Adelaide, Australia) *(OzCHI '13)*. Association for Computing Machinery, New York, NY, USA, 143–146. doi:10.1145/2541016.2541096

[22] Nick Montfort. 2021. *Exploratory programming for the arts and humanities*. MIT Press.

[23] Santiago Perez De Rosso and Daniel Jackson. 2013. What's wrong with git? a conceptual design analysis. In *Proceedings of the 2013 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software* (Indianapolis, Indiana, USA) *(Onward! 2013)*. Association for Computing Machinery, New York, NY, USA, 37–52. doi:10.1145/2509578.2509584

[24] Eric Rawn, Jingyi Li, Eric Paulos, and Sarah E. Chasins. 2023. Understanding Version Control as Material Interaction with Quickpose. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 126, 18 pages. doi:10.1145/3544548.3581394

[25] Casey Reas and Ben Fry. 2006. Processing: programming for the media arts. *AI & SOCIETY* 20, 4 (01 Sep 2006), 526–538. doi:10.1007/s00146-006-0050-9

[26] Christian Remy, Lindsay MacDonald Vermeulen, Jonas Frich, Michael Mose Biskjaer, and Peter Dalsgaard. 2020. Evaluating Creativity Support Tools in HCI Research. In *Proceedings of the 2020 ACM Designing Interactive Systems Conference* (Eindhoven, Netherlands) *(DIS '20)*. Association for Computing Machinery, New York, NY, USA, 457–476. doi:10.1145/3357236.3395474

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

[28] Meta Open Source. [n. d.]. React. https://react.dev. Accessed: 16-04-2026.

[29] Sarah Sterman, Molly Jane Nicholas, and Eric Paulos. 2022. Towards Creative Version Control. *Proc. ACM Hum.-Comput. Interact.* 6, CSCW2, Article 336 (Nov. 2022), 25 pages. doi:10.1145/3555756

[30] 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 *Proceedings of the 2023 ACM Designing Interactive Systems Conference* (Pittsburgh, PA, USA) *(DIS '23)*. Association for Computing Machinery, New York, NY, USA, 326–342. doi:10.1145/3563657.3595969

[31] Tree-sitter. 2018. Introduction - Tree-sitter. https://tree-sitter.github.io/tree-sitter/. Accessed: 16-04-2026.

[32] Mauricio Verano Merino and Juan Pablo Sáenz. 2023. The Art of Creating Code-Based Artworks. In *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI EA '23)*. Association for Computing Machinery, New York, NY, USA, Article 271, 7 pages. doi:10.1145/3544549.3585743

[33] Wenhua Yang, Chong Zhang, Minxue Pan, Chang Xu, Yu Zhou, and Zhiqiu Huang. 2022. Do Developers Really Know How to Use Git Commands? A Large-scale Study Using Stack Overflow. *ACM Trans. Softw. Eng. Methodol.* 31, 3, Article 44 (April 2022), 29 pages. doi:10.1145/3494518