Forking a Sketch: How the OpenProcessing Community Uses Remixing to Collect, Annotate, Tune, and Extend Creative Code

Blair Subbaraman
b1air@uw.edu
University of Washington
Seattle, Washington, USA

Shenna Shim
shenns@uw.edu
University of Washington
Seattle, Washington, USA

Nadya Peek
nadya@uw.edu
University of Washington
Seattle, Washington, USA

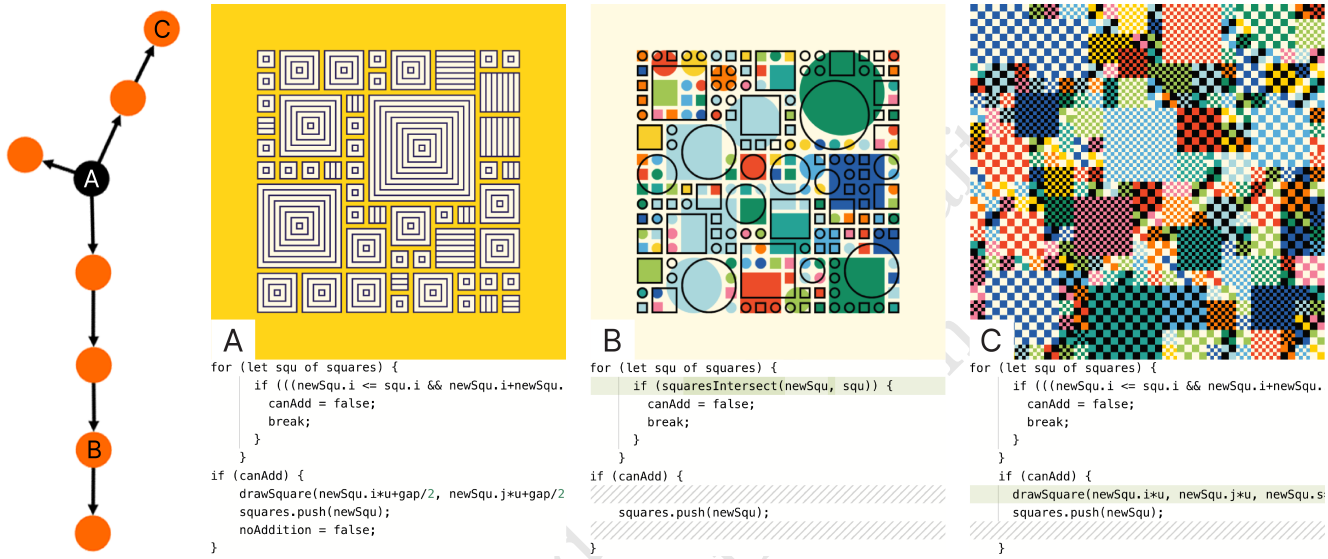


Figure 1: We pair network analysis with qualitative techniques to capture high-level patterns and meaningful details about how creative coders remix sketches. An excerpt of a remix subgraph is shown. *Roni Kaufman* iterates on their original sketch (A) in multiple ways (B, C). Code snippets highlighting example changes are shown below.

ABSTRACT

Creative coders create programs that generate visual output. Frameworks such as p5.js support sketching with creative code. Given the focus on expressivity over functionality, code reuse in creative coding practice is distinct from other programming contexts. Remixing facilitates iteration on existing code, but we have yet to understand how creative coders use remixing in practice. To understand creative coder remixing strategies, we studied the community of OpenProcessing, a site dedicated to sharing code-generated artworks. We found that 30% of the 1.2 million sketches in our data set were involved in remixing. For in-depth insight, we qualitatively analyze source code and visual output of 350 antecedent-remix pairs. We present on the diversity of ways that authors remix to curate projects, annotate process, explore variations, and transform existing sketches. We discuss the prevalence of these types and implications for supporting a multiplicity of remixing strategies in creative work.

CCS CONCEPTS

• **Human-centered computing** → *HCI theory, concepts and models*; *Empirical studies in HCI*.

KEYWORDS

Creative Code, Remixing, p5.js, Processing, Creativity, Digital Art

ACM Reference Format:
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 *Designing Interactive Systems Conference (DIS '23), July 10–14, 2023, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3563657.3595969

1 INTRODUCTION

In the early 1970s at Bell Labs, artist-in-residence Lillian Schwartz was advising statistician John Chambers on the use of color in his visualizations [50]. The visuals, intended for a scientific audience, were produced using an early domain-specific language for creating still and moving images with code [26]. In their work together,

Chambers identified lines of the program for Schwartz to edit. Reflecting on the experience in a 2014 interview, Schwartz recalled how "miraculously, just changing that one line changed the whole image" [58]. The ensuing collaboration turned into "Papillons", one of Schwartz' many computer-generated films which are now genre-defining early works of computational art [49]. Schwartz was able to explore visual outputs by creatively reusing existing code, transforming a mathematical visualization into a novel artwork.

We can view Schwartz' story as an example of remixing: repurposing an existing media artifact into something new. At the same time that Schwartz was creating her early computer visuals, music producers in Jamaica were stripping vocals from songs and re-recording them with new sonic effects. It is this creative practice that informs contemporary understandings of a remix culture which champions the free exchange of ideas [39]. Remix in music hit popular culture in the 1980s concurrently with the first personal computers, and the ability to easily edit digital media catalyzed remixing practices across a range of content creation areas. Competing social theories advocate for remixing as a site of open innovation [28] and deride it for generating unoriginal content [23]. Following this work, HCI researchers have empirically investigated what makes some projects more suitable for remixing than others [4, 15, 41], when a remixed creative project is of higher quality than the original [14], and how remixing can support computational thinking with code [8]. This prior work successfully evaluates motivations for and effects of remixing. By contrast, our research is focused on how creative practitioners use remixing in their work. Artists value quickly exploring new ideas [31], and remixing has emerged as a way to iterate on existing code. As our starting insight for this work, we hypothesize that creative practitioners' approach to code reuse is distinct from other programming contexts.

Since Schwartz's early explorations, artists have developed numerous domain-specific software tools to support programming for expressivity over functionality, including Processing, p5.js, and openFrameworks [32, 46, 57]. While each is anchored around a specific creative focus, all combine artistic practice with general purpose programming to offer new opportunities for creative expression. These software tools have evolved into communities of practitioners who together have come to be known as creative coders: "artists, designers, architects, musicians, and poets who use computer programming and custom software as their chosen media" [30]. In this paper, we focus on OpenProcessing: an online community dedicated to sharing p5.js and Processing projects. Authors on OpenProcessing can browse artworks, post new projects, and remix existing work. The site therefore offers an empirical setting to investigate remixing in the context of creative practice.

We aim to shed light on the diversity of current remixing practice. To this end, we ask: *What remixing strategies do creative coders employ to reuse code?* While code reuse has been studied in various software engineering [9] and novice programming [54] contexts, we focus on creative coders. HCI researchers have increasingly considered how digital tools can support expressive practices [20]; our intent is not to classify what a remix can or cannot be, but rather to help situate the development of useful tools by understanding the actions of existing communities.

To examine creative code remixing strategies, we designed a three phase analysis. We first conducted a network analysis of OpenProcessing, an existing creative coding community. Using a comprehensive data set of 1.2 million projects, we recreated the network of all projects which are remixed or remixes. We leveraged this remixing graph to surface subgraphs relevant to our research questions, including the most remixed projects and the longest chains of remixes. In the second phase, we used these subgraphs as field sites for reflexive thematic analysis. Using traditional qualitative coding approaches in conjunction with file comparison tools, we analyzed remixed code and conceptualized four themes which we believe speak to current remixing practice. While remixing is commonly appreciated for the generation of new artifacts, we find creative coders *collect* artifacts without making any code changes; we see a variety of *annotations* which use inline code comments to log personal process and informally version code snippets; small code edits can have large visual consequence, and we see creative coders *tune* existing parameters to explore variations in output; and finally, we explore the range of ways that creative coders *extend* sketches with precise stylistic interventions and larger reinterpretations of existing code. Beyond a single remix, we moreover see how remixing is used to manage families of changes which pursue multiple aesthetic directions. In a final phase, we measure the prevalence of these strategies in the community using our themes as codes in a content analysis. The results provide an additional layer of insight to our initial community analysis, indicating that over half of all remixes tune pre-existing parameters while comparatively fewer add code or inline comments. We discuss the implications of both our methodology and findings for building systems which support creative code and other exploratory programming community. As creative code is increasingly used to support computational education, we finally consider the implications of our remixing strategies for understanding and facilitating informal learning.

In summary, our overall contributions are:

- An interpretive analysis of remixing on OpenProcessing;
- A set of remixing strategies and their prevalence;
- Design provocations for HCI systems which seek to support creative community through remixing.

2 BACKGROUND & RELATED WORK

Our work contributes to two areas of HCI research: (1) studies of online remixing behavior, and (2) inquiry into creative practice. Building on existing remixing research, we complement community-scale network analysis with qualitative techniques to analyze code changes between remixed programs. Doing so grants insight into specific creative code reuse practices. We further situate our work against related software engineering and end-user programming research to distinguish creative coding from previously studied programming settings. In this section, we provide an overview of related work and detail how our approach builds upon prior studies.

2.1 Remixing in HCI

While remixing has been the subject of theory and analysis across disciplines from the humanities to the social sciences, we focus on HCI research to contextualize our contributions. Large remixing communities which grew online over the last 15 years offered an opportunity to empirically investigate social theories at a scale not

previously possible. One thread of this research has focused on organizing and quantifying community-scale data. Cheliotis and Yew [5] undertook a network analysis of a music remixing community to understand structural patterns. They found, for example, that artifacts which inspire multiple new remixes (i.e. branching patterns) were the most common in the community. While we use tools from social network analysis, we also use the results to surface field sites for subsequent qualitative analysis. In this way, we take inspiration from Oehlberg et al. [41]. Following a network analysis of remixing on the digital design file sharing site Thingiverse, the authors thematically cluster highly remixed source files to identify patterns. Our approach differs in that we are concerned with specific changes made between remixed code files. Textual programming files additionally permit qualitative analysis techniques not possible with 3D design files. We undertake an inductive thematic analysis to conceptualize remixing strategies.

A related body of work empirically tests social theories around remixing. The Scratch online community features heavily in this scholarship. Scratch offers a visual, block-based programming language targeted at novice programmers and students. Scratch is the largest online remixing community, composed of a core audience of 8-16 year olds [47]. Hill and Monroy-Hernández [14] utilized Scratch projects to test the theory that remixed artifacts are of higher quality than individually authored projects. They found that remixes receive lower peer-ratings than single-authored works. Related work also reveals a trade-off between generativity and originality in remixed works on Scratch [15]. Given Scratch's focus on young programmers, researchers have additionally studied ways remixing can support informal learning [8]. Our research is similarly interested in understanding how remixing plays out in a real-world community. We differ in employing inductive, descriptive methods to speak to the diversity of ways that practitioners remix. Additionally, we propose a novel group of practitioners (i.e. creative coders) in a novel community setting (i.e. the creative coding platform OpenProcessing). To this end, our study of remixing intersects with accounts of collaborative art practice [22], error and surprise in creative practice [21, 53], and the repurposing of found objects [18]. In our analysis, we focus on specific remixing practices distinct to creative code.

2.2 Creative Coding Tools and Community

Frustrated by the incongruence between their creative goals and the software available to them, visual artists have developed domain-specific software tools surrounded by vibrant communities [32, 35, 46, 57]. Our work focuses on a related set of creative coding tools and associated communities. Processing is a popular Java-based programming software started in 2001 [46]. The project has since been reinterpreted for the web as p5.js, which has over 1.5 million users [34]. p5.js (or p5 for short) provides a Javascript library to make sketching with code as intuitive as sketching with paper and pen. These tools are used diverse settings, from computer science classrooms to professional artworks [43]. The website OpenProcessing is an independently created social website which supports sharing and remixing projects made with Processing and p5.js. To our knowledge, it is the largest online community for sharing creative code projects.

Prior HCI work has engaged creative code in a variety of ways. Li et al. [31] set out to understand how artists use and develop custom software. To do so they interview visual artists, several of whom report the use of Processing or p5.js. Their results surface frictions between the priorities of commercial tools the goals of artists, and suggest collaboration opportunities between artists and systems designers. Related work has similarly argued for pairing art production with tool production, as artists already actively shape the tools they use [19]. Verano Merino and Sáenz [62] further reflect on the particularities of creative coding through interviews with code artists. The intimate relationships between artists and their technical tools motivates our interest in creative coders' remixing practices. Recent research also probes how creative practitioners across a range of mediums use version histories in their process [56]. The authors find that conventional version control systems are misaligned to creative practice. In line with this work, our findings suggest ways that artists appropriate remixing as a method of informal version control. Related systems support creative coders' creative process through integrated version control systems [45] and screenshot driven version control approaches [33]. Overall, our study builds on this rich body of work concerning code and creative practice. Where previous work focuses on practitioners' individual practice through interview-based methodologies, we focus on the computational artifacts themselves. In particular, we isolate remixing as one key aspect of creative coding practice which is particularly important in building and sustaining community.

2.3 Tailoring and Customizing Software

In the HCI literature, creative coders have been cited as end-user programmers [e.g. 25, 27]. We follow Ko et al. [27] in their definition of end-user programming as coding for personal, rather than public, use. We similarly note that this distinction does not denote inexperience. In addition to domain expertise in the visual arts, creative coders can have years of programming experience and might also work as professional software developers. OpenProcessing hosts projects from creative coders with a variety of backgrounds and expertise including students, hobbyists, and professional artists.

The framing of creative coders as end-user programmers allows us to make connections to prior literature. Mørch [38] defined customizing, integrating, and extending as three levels of end-user software application tailoring. Only the final category, extending, involves adding new code. This context differs from remixing on OpenProcessing, where all changes involve editing code. In reference to this prior work we renamed our final theme "extending". The definition of our theme did not change in the renaming process, consistent with our inductive approach to thematic analysis. Creative coding has also factored in more contemporary literature around exploratory programming. Kery and Myers [25] define exploratory programming as a task wherein (1) programming is used as a medium to experiment with new ideas, and (2) the programmer is not coding to a predefined specification. Research around exploratory programming has largely focused on data scientists. Our focus on creative coders thus contributes to this existing body of literature.

Finally, our focus on coding practices intersects with prior work in software engineering. Prior work has studied the relationship

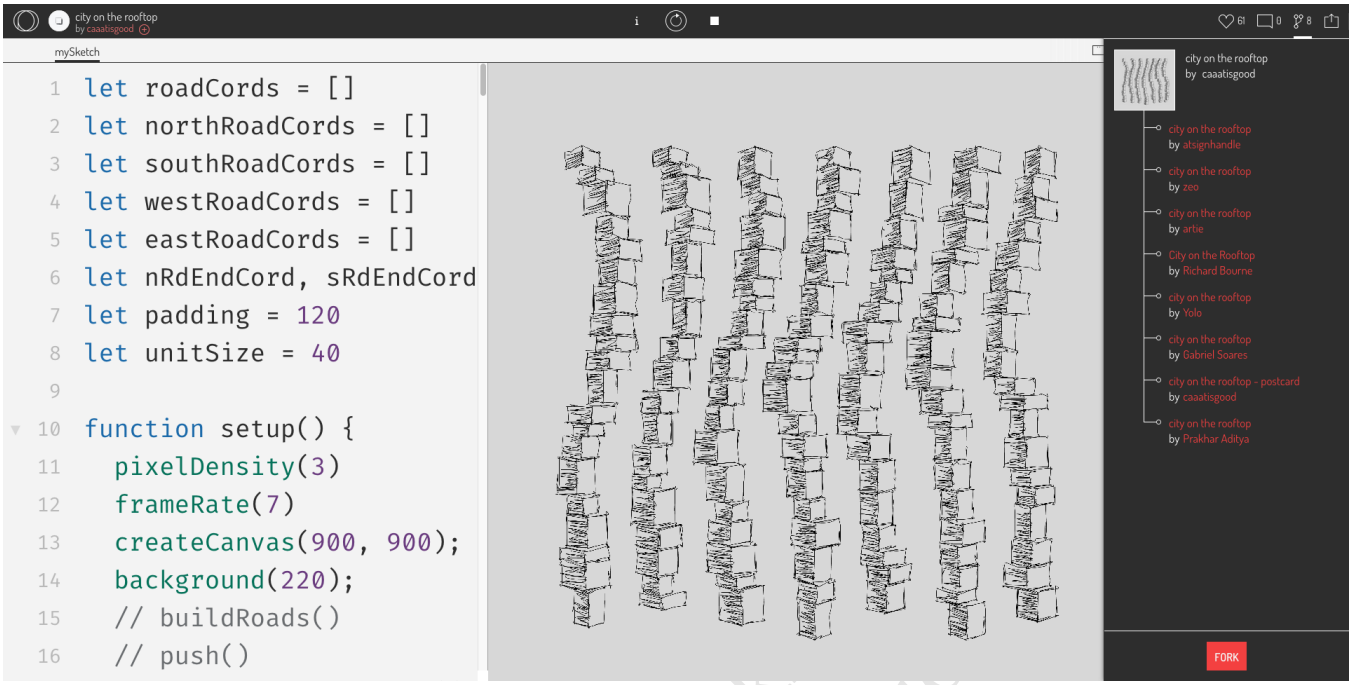


Figure 2: The OpenProcessing browser-based interface. Users can look at both the code (left) and corresponding visual (middle). Edits can be made directly to the code to observe output. These changes can be saved and published as a fork, or remix. Users can view all projects which fork the current sketch in a sidebar (right). The sketch shown is by *caaatisgood*.

between inline code comments and documentation [13], how software developers forage for relevant information [9], and how to automatically detect sections of code which might benefit from refactoring [61]. In its pursuit of creativity over functionality, we are interested in how creative coding compares and differs from traditional software engineering contexts.

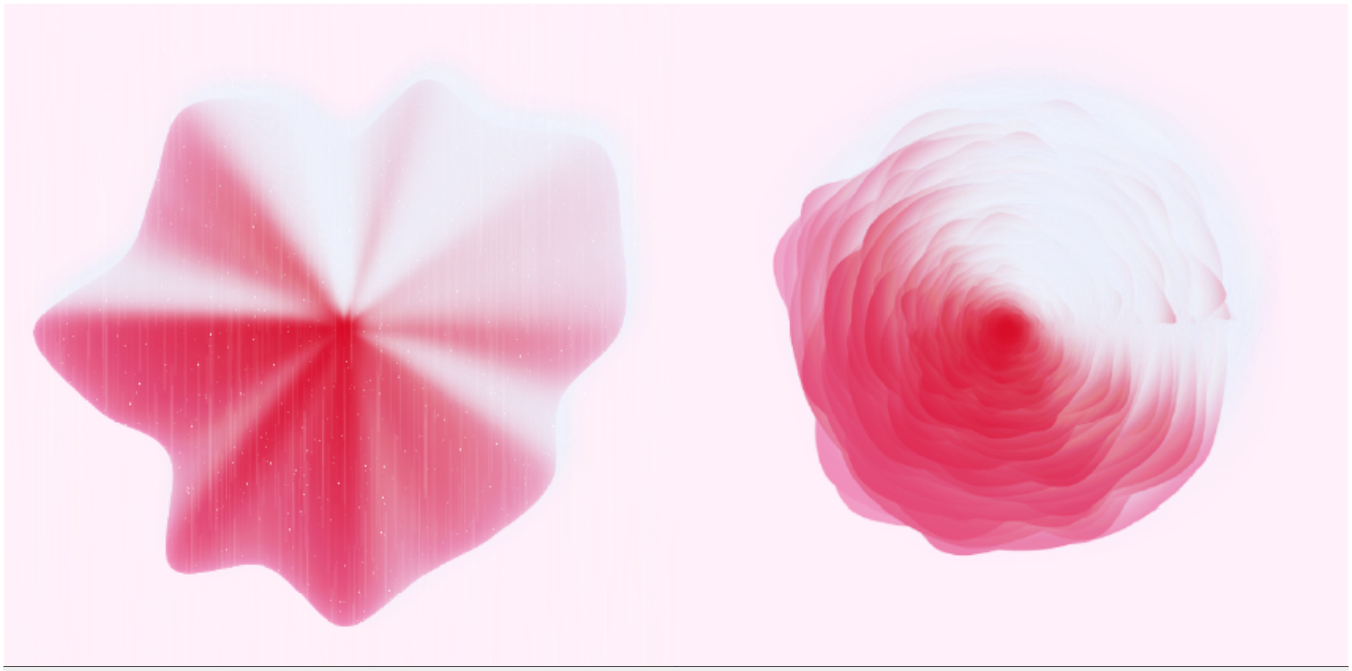
3 EMPIRICAL SETTING: OPENPROCESSING

To gain insight into creative code remixing strategies, we turn to an existing online community. OpenProcessing is an online community for creative coders to write and share projects. It supports code written using the popular creative coding libraries p5.js and Processing. OpenProcessing was independently founded by Sinan Ascioglu, separate from the development of the p5.js library itself [52]. Ascioglu continues design and development of the website. The site has accumulated over a million creative code projects since launching in 2008. It is free to make an account and paid features are also available with particular relevancy for educators and students. The site development is active and the creator regularly adds new features, such as a recent ChatGPT integration to help debug code errors.

We walk through the OpenProcessing interface, defining several key terms along the way. A creative code project is called a *sketch*. A sketch is comprised of a visual output and the code used to generate the visual. Sketches on OpenProcessing are uploaded by authors with optional descriptions. The site's landing page shows trending projects; upon selecting one, users are shown the visual (Figure 2 center). Importantly, users can not only view a sketch's source code (Figure 2 left) but also *fork* it. A common feature in software engineering contexts, an author who forks a project duplicates the sketch to their own account. They can then edit the code and re-publish it under their username. In doing so, the relationship between the fork and its source is preserved. When navigating to a forked sketch, a pop-up will appear linking to the parent project. Forks are contrasted with de-novo sketches, which are projects directly published by the author. To consolidate language, we will refer to the original sketch as the *antecedent* and the sketch which is a fork as a *remix*. Notably, these actions can be chained. One project's antecedent might be another project's remix. A tree representing all sketches derived from the current project is navigable from a sidebar (Figure 2 right), though traversing up the tree must happen manually.

4 METHODS

Our analysis of remixing on OpenProcessing was conducted in three phases. Phase one uses the tools of social network analysis to discover all sketches involved in remixing. In addition to presenting community-level data, network analysis helps us select productive field sites for subsequent qualitative investigation. Phase two consists of a reflexive thematic analysis wherein we sample antecedent-remix pairs from subgraphs identified in phase one. We make use of source code comparison software to make easily visible the changes between a remix and its antecedent. We subsequently use this 'code diff' as a key piece of data to qualitatively analyze in our thematic analysis. We conceptualize themes which we present as distinct remixing strategies; phase three uses these themes in a



```
function myCircle(x, y, rad) {
  let numLayers = 200
  for(let i = 0; i < numLayers; i++) {
    let vertices = []




    for (
      let theta = 0;
      theta < TAU;
      theta += TAU / 20) {
        ...
```

```
function myCircle(x, y, rad) {
  let numLayers = random(100, 300)
  for(let i = 0; i < numLayers; i++) {
    let vertices = []
    let flick = random(10,11)
    // for(
    // let theta = 0;
    // theta < TAU;
    // theta += TAU / random(10,30)) {
    for (
      let theta = 0;
      theta < TAU;
      theta += TAU / flick) {
          ...
```

Figure 3: Illustrating our qualitative coding process using an antecedent (left) and remix (right) pair from our data set. In remixing a sketch from user *Taiki Saito*, *Owaun Scantlebury* demonstrates tuning (blue), creative code extensions (purple), generic extending (green), and annotating (yellow).

content analysis to measure their relative frequency. We detail our methods in this section.

4.1 Network Analysis

Sketch metadata on OpenProcessing can be retrieved by querying an API with the sketch ID. In the case of a remix, this includes the ID of the sketch's antecedent. We collected data from all possible identification numbers in May 2022. As our analysis is interested in identifying remixing patterns which might be atypical, comprehensive data collection is necessary over representative sampling.

A social network is commonly abstracted as a graph wherein nodes represent individuals and edges communicate a relationship between them [64]. Our goal is to create the remixing graph where each node is a sketch, and edges point from an antecedent sketch to a remixed sketch. We used custom Python code to prepare the collected data for analysis, the NetworkX library [11] to analyze the data, and the open-source software Gephi [1] to visualize the network. Howard [17] contends that network analysis is useful to justify case selection for subsequent qualitative analysis. We take such a network ethnography approach, using the remixing graph to identify sources of remixing which would be otherwise invisible.

4.2 Thematic Analysis

We follow Braun and Clarke [2] to conduct a reflexive thematic analysis. While the relevant data for a thematic analysis is traditionally text such as an interview transcript, our unit of analysis is a antecedent-remix pair of OpenProcessing sketches; that is, the program and visual output from both an antecedent sketch and a fork of the antecedent. Our full data set was comprised of 350 antecedent-remix pairs identified in our network analysis.

Two researchers assigned qualitative codes[1] to the data using a browser-based tool. This tool provided links to remix and antecedent sketches so that all relevant information could be inspected online, including: the sketch program, its associated visual, author descriptions, and user comments on the platform. After an initial period of data familiarization, we observed that changes to the program are crucial to understanding what occurred between an antecedent and its remix. We therefore added a 'diff' view to our tool which highlights the differences between the remix and antecedent programs. Such file comparison utilities are common in software engineering contexts to easily identify what has changed between two files.

Over 8 weeks, we developed a codebook using an inductive open-coding approach. Over time, qualitative code categories increasingly focused on changes made visible by the program diff. We wrote memos to produce text that worked across the code set. We ultimately conceptualized four themes which we believe each capture a distinct remixing strategy. After naming and defining our themes, we found they could productively be put into conversation with prior research in end-user programming. In a final meeting, we updated our theme names to align with prior work where relevant. The definition of our themes did not change in this process, consistent with our inductive approach to thematic analysis.

4.3 Content Analysis

Each of our themes involve specific program edits. These include no change to the original code (*collecting*), adding inline comments (*annotating*), editing pre-existing parameters (*tuning*), using functionality exposed by the creative coding library (*creative code extensions*), and adding new code which does not require the creative coding library (*generic extensions*). After we contextualized these strategies in a thematic analysis, we sought to measure the prevalence of each on OpenProcessing. Two human coders analyzed a randomly sampled collection of antecedent-remix pairs from the remix graph. We coded sketches for the presence or absence of each theme. In accordance with Neuendorf [40], we begin with a norming stage wherein each researcher coded the same 100 projects and disagreements were discussed. These disagreements were subsequently accounted for in our coding process. In a reliability phase, the coders then coded the same set of 200 projects. We found we were able to code for each theme reliably with the following Krippendorf's alpha values: collecting ($\alpha = 0.97$), annotating ($\alpha = 0.92$), tuning ($\alpha = 0.88$), creative code extending ($\alpha = 0.91$), and generic extending ($\alpha = 0.92$). An additional 100 projects were then coded by a single coder for a total of 400 projects.

4.4 A Worked Example

While we further discuss the meaning and nuance of each theme in our findings, we walk through an example to clarify our qualitative coding process. Figure 3 shows an antecedent-remix pair from our data set alongside a matching excerpt from each program. We use a unique color to differentiate each change based on the qualitative code we assign it. Line two edits the value of the pre-existing variable `numLayers` (*tuning*), by way of the p5-specific function `random()` (*creative coding extension*). The remix goes on to declare the new variable `flick` (*generic extension*) and comments out a `for` statement (*annotation*). We would therefore assign each of these qualitative codes to this remix. Our qualitative coding process does not measure intensity; although there are multiple instances of tuning in Figure 3, we code only for presence or absence. Moreover, we treat all inline comments as annotations and do not assign tuning or extension qualitative codes to their contents. Finally, we do not assign any qualitative codes to lines of the program which have been removed.

[1]To avoid confusion, we use 'code' in this section to refer to qualitative codes, and 'program' to refer to computer code.

4.5 Limitations

We note several limitations regarding our chosen methods. Our focus on sketches themselves does not grant us insight into why an author decided to remix. While we pair network analysis with qualitative techniques to capture both high-level patterns and meaningful details, interview based studies would complement our approach. An analogous study might create the network of authors who remix each other; the resulting author remixing graph can guide researchers in recruiting interview participants. Tseng and Resnick [60], for example, found that most readers on the project documentation website Instructables are searching for project ideas and new techniques rather than recreating a project directly. Understanding the reasons why an author chooses to remix can add additional depth to our analysis. Moreover, OpenProcessing does not necessarily reflect the practices of all creative coders. Communities built around other tools might have correspondingly different interests and values. The practices of creative coders offline might be different from the ones demonstrated in a public platform. With these limitations in mind, we aim to understand how the code reuse practices of this community can inform future systems and studies.

5 UNDERSTANDING HIGH-LEVEL REMIXING PRACTICES

At the onset of our research, the members of the OpenProcessing community had shared over a million sketches on the platform. We use the tools of social network analysis to make sense of this data. Of the 1,500,800 queried sketches, 75% had publicly available metadata, while the remaining 25% of sketches were either private, deleted by the author, or removed for violating the terms of service. We use the remaining 1,119,988 sketches for our analysis. We find that 30% (356,946 sketches) of the accessible sketches were either sources for remixing, remixes themselves, or both. This number excludes 123 erroneous 'self-loops' in which sketches are their own remix, likely due to an error at the time of upload. This percentage speaks to the prevalence of remixing within the community.

The remixing graph is built as shown in Figure 4a, where nodes are sketches and edges are directed from antecedents to remixes. The full network is visualized with Gephi to make clusters of remixes visible (Figure 4b). We note key takeaways from the remixing graph. The entire graph is composed 79,453 subgraphs. Most of these subgraphs are small, with a mean size of four nodes. This data tells us that most sketches that are not heavily remixed. An example of a smaller subgraph is shown in Figure 4c, where a de-novo sketch is remixed five times, one of which generates an additional two remixes. Our network analysis allows us to filter and find subgraphs

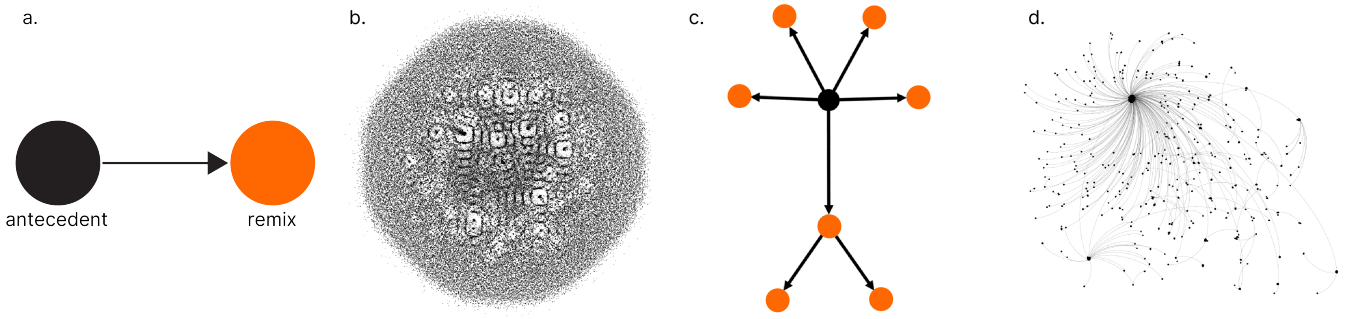


Figure 4: (a) We conceptualize remixing as a directed graph wherein nodes are sketches and edges points from antecedents to remixes. (b) The OpenProcessing remixing graph consists of almost 80,000 subgraphs. 30% of accessible sketches are involved in remixing. (c) An example subgraph consisting of 8 sketches. (d) We filter the remixing graph to find subgraphs with specific features. Shown is the largest subgraph in our data set consisting of 1594 total sketches. (b)-(d) were made using Gephi. [1]

by size. For example, the largest subgraph is shown in Figure 4d and is made up of 1,594 sketches. Altogether, this data suggests that sketches are remixed in a diversity of ways.

The proportion of sketches which are remixes (30%) aligns with current statistic from the the Scratch community [65]. However, we note two key differences in our OpenProcessing data set. First, Scratch does not allow remixes which do not make any changes to the original sketch, whereas OpenProcessing does. Moreover, on Scratch it is not possible to remix your own sketch. While there are workarounds noted in the forums [65], the fact that it is against community guidelines hinders this behavior. No such guidelines exist on OpenProcessing. In fact, we find it is common: we assembled the graph of remixes which have the same author as its antecedent on OpenProcessing and found that 49% of all remixed sketches (14.7% of all publicly available sketches) are involved in such self-remixing. We take note of this behaviour throughout our analysis.

We collate a selection of antecedent-remix pairs sampled from the remixing graph for in-depth analysis. In addition to randomly chosen antecedents and remixes, we identify a set of superlative subgraphs which we hypothesize are productive sites for qualitative investigation. These include the largest overall subgraphs, the sketches with the most direct descendants, and the longest remix chains. Our final data set for qualitative analysis consisted of 350 antecedent-remix pairs sampled from these field sites, including: 100 pairs randomly selected from the remixing graph, 5 pairs randomly selected from each of the 20 largest subgraphs, 5 pairs selected from each of the 20 most remixed sketches, and 1 pair selected from each of the 50 longest remixing chains. This data set was used to conduct a reflexive thematic analysis.

6 CONCEPTUALIZING REMIXING STRATEGIES

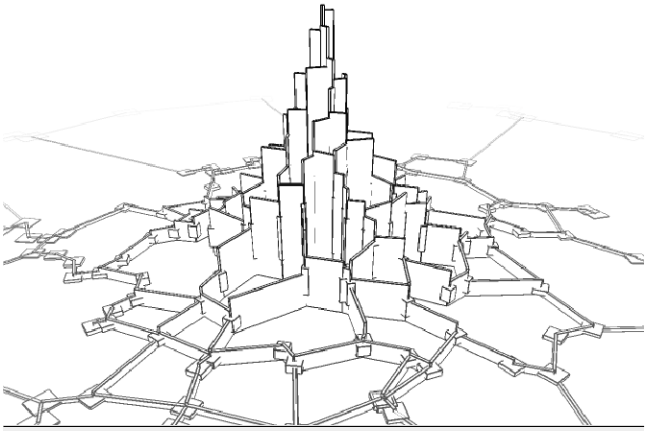
We conceptualize four high-level remixing strategies, each of which we tie to specific code edits. While broadly applicable to any edited code, we provide illustrative examples to distinguish the use of these strategies in a creative coding context. While remixing is often appreciated for its ability to create new artifacts, we see creative coders *collecting* sketches without making any edits. We see a variety of *annotations* which use code comments to log personal process and informally version code. Small code edits can have large visual consequence, and we observe creative coders *tuning* existing variables to explore a range of visual output. We finally see a broad set of *extensions* which add new code. In particular, we observe remixes which make targeted changes to build on a sketch in specific ways, remixes which use the antecedent as conceptual inspiration in larger changes, and families of remixes which explore multiple creative directions. A single remixed sketch might demonstrate several of the behaviors we describe, and we therefore consider the productive interplay and frictions between each strategy.

We have decided not to anonymize the OpenProcessing usernames of the authors whose work we include in this paper. We have contacted or attempted to contact the 20 accounts whose work we include. All eleven who have responded have asked for their usernames to be included alongside their work. Given that community members have a preference for public attribution, we believe it is still appropriate to use the usernames of the accounts from whom we did not receive a response– for an in-depth discussion of when not to anonymize in internet research, see [12]. We have also added all usernames to this paper's acknowledgements section. We present all code as it was published on OpenProcessing, with the exception of small formatting changes for clarity in presentation.

6.1 Collecting Sketches Without Making Changes

Conventional understanding of remixing focus on the creation of new artifacts. However, we find that many remixes on OpenProcessing contain identical code and therefore visual output. We call this behavior *collecting*. The most straightforward of our themes, we define collecting as a remix with no changes made to the code. It is therefore the only mutually exclusive remixing behavior we present.

While users of OpenProcessing have the ability to 'like' sketches, we see some accounts dedicated to collecting. Several usernames in our data set include "Best Sketches" or similar language in their



```
function drawJoiningWalls() {
/* for each pair of agents for whom there is no
other agent nearer to either, draw a wall */
```

Figure 5: *Neill Bogie* explains how walls are being drawn in *Naoki Tsutae*'s sketch.

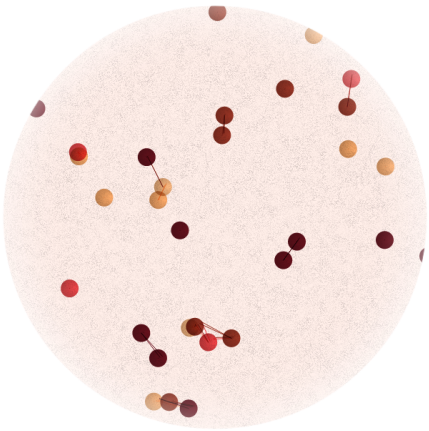
handle. These accounts feature only collected sketches, appropriating the remixing features of the platform to curate projects. Prior to manual investigation, there is no way to know what has changed between an antecedent and remixed program. This can make navigating the remixing history of a sketch difficult, as it is not apparent if a past or future version is any different that the current one.

6.2 Annotating Sketches with Inline Comments

Not all changes to code necessarily affect the visual output. We find that *annotating* code using inline code comments frequently occurs in our data set. Code commenting is well-studied in software engineering [13] and other exploratory programming contexts such as data science [24]. Here we present examples from our data, noting overlaps and differences from prior work. In particular, we see annotations used to learn about others' sketches, log personal process, and informally version lines of code.

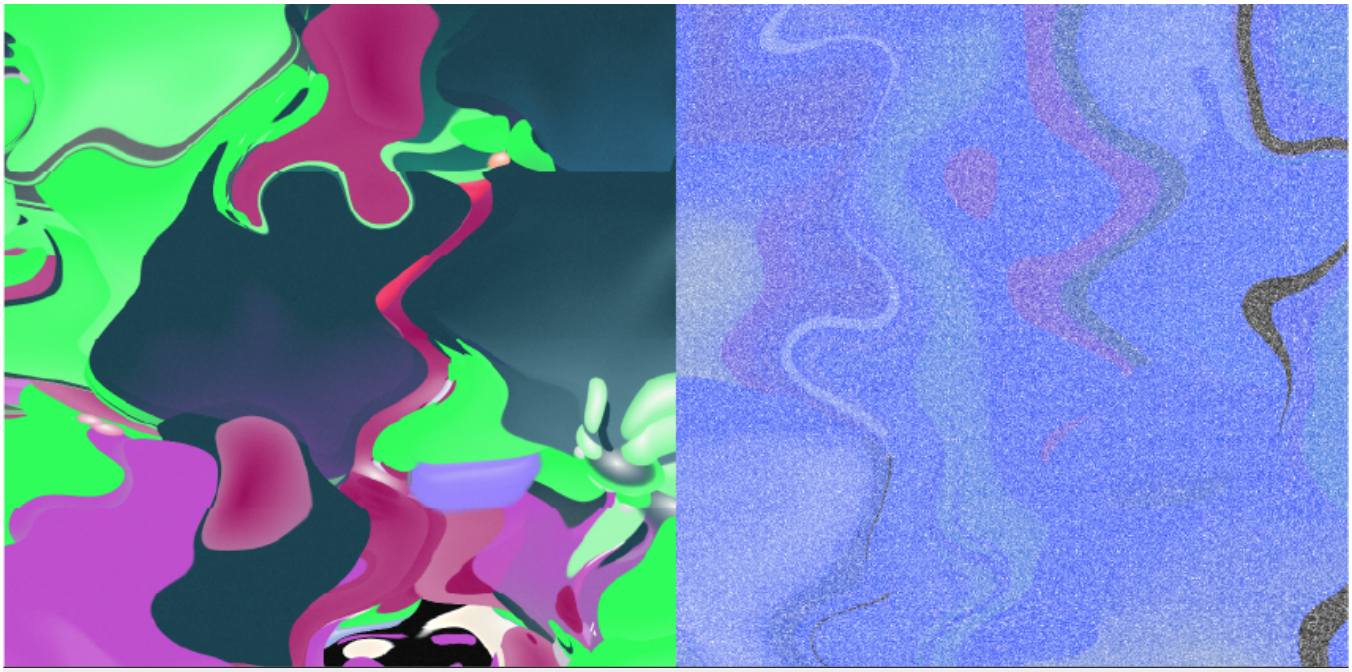
In our analysis, we see comments used in remixes to annotate sketches as the remix author learns how it works. In Figure 5, an antecedent visual is shown above a remixed code excerpt, with additions made in the remix presented in orange. *Neill Bogie* remixes a sketch by *Naoki Tsutae*. In the description of the sketch, they write that they are *"breaking down Naoki Tsutae's work to learn from it."* Throughout the code, the remix adds comments which provide high-level explanations of various functions. The original sketch animates the position of various particles, or 'agents'. The remixed excerpt shown notes how the walls are being drawn between these particles. After annotating this sketch, the remix author goes on to create several de-novo sketches in which they re-implement this algorithm themselves. While leaving explanatory comments in code is not new behavior, remixing serves as a way for *other* authors to annotate a sketch as they learn how it works. We additionally see examples of annotations which extend inline comment explanations and fix grammatical errors. We also saw several examples of remixes which translated comments into different languages.

We contrast explanatory annotations with process-oriented ones. Process-oriented comments log personal process in ways distinct to creative code. In our data set, process-oriented comments were often brief expressions of an issue or frustration left by authors remixing their own sketch. A particularly illustrative example is shown in Figure 6. A year after posting their original sketch, *Aaron Reuland (a_soluble_fish)* remixed it to annotate their process. These comments include links to references they draw from including other OpenProcessing authors and online examples, explanations of what various sections of code accomplish, and reflections on what they have learned in the past year. Throughout, these comments make transparent the author's process. In the excerpt shown, they describe where they sourced the code snippet to create a *"papery"* texture. While they say they copy-pasted it at the time of the original sketch, in their annotation they take time to explain its use. In other sections, they clarify what techniques are original and similarly cite inspirations. Comments such as these would not usually be found in production code. In a creative coding context however, such comments reveal process in a public setting. We moreover see these annotations as a way to reckon with attribution; a remix can only have a single antecedent, but code might be inspired by many prior works. The effort of manual credit-giving has been shown to be valued by community members in prior research [37].



```
/* ok, this texture algorithm I definitely stole. 98% sure
it was from **Che-Yu Wu (openprocessing.org/user/139364)**
an amazingly talented artist, who also adds lots of
in-progress stuff to openProcessing- nice to learn from
(not that I learned from this at the time I made this,
so much as I copied and pasted it) creates a nice papery
texture by applying noise to the pixel array, that is
blended with the rest of the 'art' later on. */
```

Figure 6: Annotations can log personal process in addition to providing explanations. *Aaron Reuland (a_soluble_fish)* remixed their own sketch a year later, adding explanatory details, sources of inspiration, and reflections.



```
margin = mySize / 100;
for (let i=0; i<int(random(50, 100)); i++) { ... }
theShader.setUniform(
  'u_time',
  millis() / 1000
);
let version = random([1,2,4,6,8])*100;
let c = random(2000, 5000);
colorMode(HSB, 360, 100, 100, 100);
```

```
margin = mySize / 10;
for (let i=0; i<int(random(500, 100)); i++) { ... }
theShader.setUniform(
  'u_time',
  millis() / 1
  );
let version = random([200,150,77,50,140]*100;
let c = random(1000, 2000);
colorMode(HSB, 21, 10, 10, 10);
```

Figure 7: Remixes can explore variations in visual output by tuning existing parameters. A remix by *naha* (right), achieves distinct visual output through editing pre-existing paramters set in the original by *SamuelYAN* (left). All of the changes made are shown in the accompanying code block, with edits in orange. Note that these tuned lines appear throughout the sketch, and are presented sequentially here to illustrate the effects of tuning.

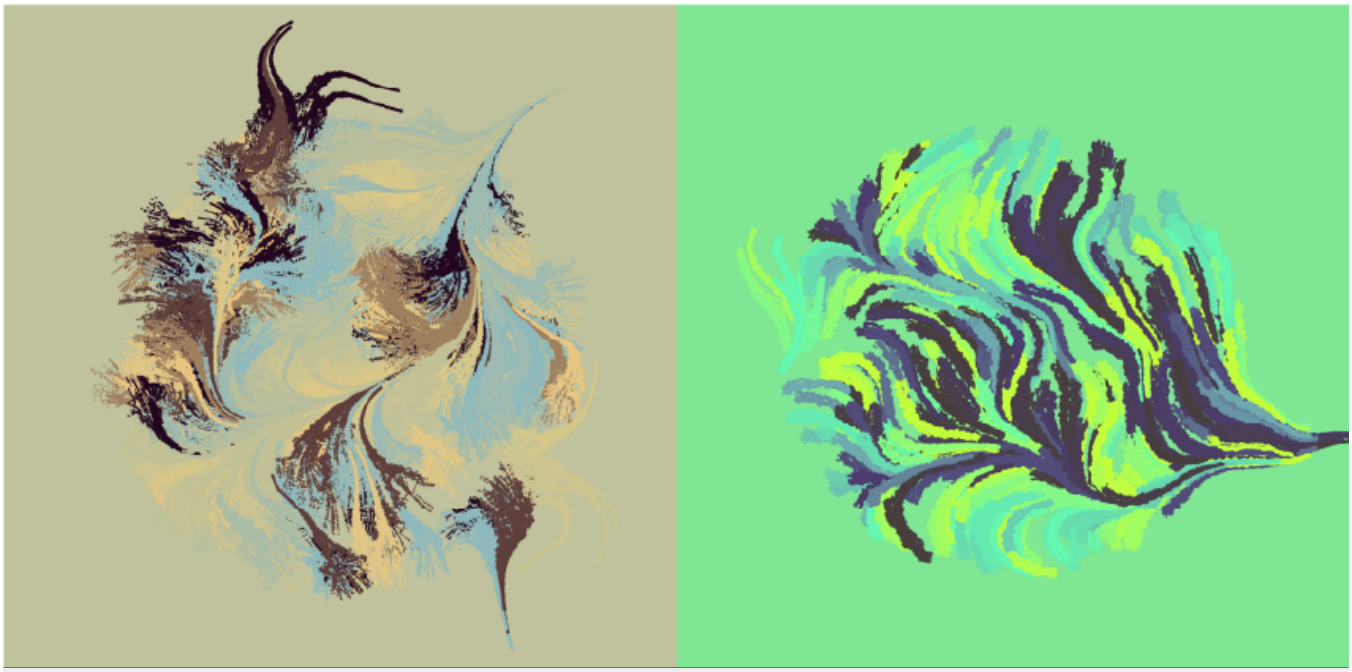
While the explanatory and process-oriented comments above are straightforward to interpret, others require contextualization in the remixing history. Figure 8 shows a matching line from an antecedent and remix in our data set. In the remix, it is not immediately clear where the number 150 comes from. By consulting the antecedent sketch, we see that this was the previous value for this variable. This strategy to archive previous values before changing them recurs throughout our data set. We also see new values left as comments. We infer that such comments are values which yielded output that authors wished to save. In this behavior, we surmise that authors of these remixed sketches are using inline comments to quickly backup, or version, individual lines of code. This informal version control aligns with previous studies of data scientists [24]. A key difference distinct to OpenProcessing is how archiving parameters can become collaborative. It is often not the original author who is versioning the previous line of code but others building from it.

Finally, comments can directly affect visual output. By commenting out lines with visual or interactive consequence, remixes can affect the look and feel of a sketch. Examples in our data set include

```
var length = 150;          var length = 100; // 150
```

Figure 8: The value `150` is left as a comment in the remix (right). Comparing the code with the antecedent shows us that this was the previous value for the variable `line`.

commenting out lines which draw shapes to the screen or limit the number of frames to be drawn each second. Deliberately commenting out lines can produce different output; by commenting instead of deleting the line entirely, the remix retains a strong trace to the antecedent. Notably, this permits exploring visuals without writing any new code. This use of comments aligns with prior studies of computational notebooks for data science wherein comments are used to control program flow [48].



```
pal = bigpal(3, pal);
br.disp(150)
this.swarm = new swarm(40, 29, 20)
```

```
pal = bigpal(6, pal)
br.disp(110 + random(-50, 50));
this.swarm = new swarm(10, 29, 20)
```

Figure 9: By tuning existing variables in code, authors can generate a set of possible visual outputs. After implementing a base algorithm, *Trrrrrr* remixes their own sketch multiple times to tune a set of relevant variables. The visual differences shown are a result of manipulating existing values. The code showing the tuned variables is shown. Note that these tuned lines appear at different locations in the sketch code and are shown here sequentially to illustrate the effects of tuning.

6.3 Tuning Existing Parameters to Explore Visual Outputs

One way to explore visual outputs is to manipulate values in existing code. We observe such *tuning* frequently in our data set. In Figure 7, we see *naha* remixing a sketch by *SamuelYAN*. A distinct visual output is reach solely through tuning existing paramters; all of the changes made in the remix are shown in the accompanying code block. In particular, we see the remix manipulating the values of pre-existing variables, the arguments to functions such as `colorMode()`, and the end condition of a `for` loop. We take all of these changes to be instances of tuning, defined by the explicit manipulation of an existing value in code.

In our data, we see that highly remixed sketches feature many tuning remixes. For example, the longest remixing chain on OpenProcessing at the time of our data collection was 106 sketches long. The source sketch consists of a face drawn with simple shapes, whose eyes follow the mouse as it moves. Almost all of the remixes of this sketch involve other authors tuning colors.

In addition to tuning others' sketches, we see that remixes by a single author can be used to manage variations. Figure 9 shows a set of sketches by *Trrrrrr*. After implementing a base algorithm, they manipulate various parameters to generate a set of distinct outputs. The code edited between two of these variations is shown; while these lines appear throughout the code, we present them sequentially to highlight the effects of tuning. *Trrrrrr* remixed the original sketch several times, making slight changes to the base algorithm each time to explore the possible outputs.

We emphasize the close relationship between tuning and annotating. In Figure 8, we saw how previous values of tuned variables were archived with inline comments. In other cases, we see tuning can make inline documentation obsolete. One such example is shown in Figure 10. In the remixed sketch, the RGB values which define the color of lines drawn on the screen are changed from a shade of red to black. The comment, however, is now out of alignment.

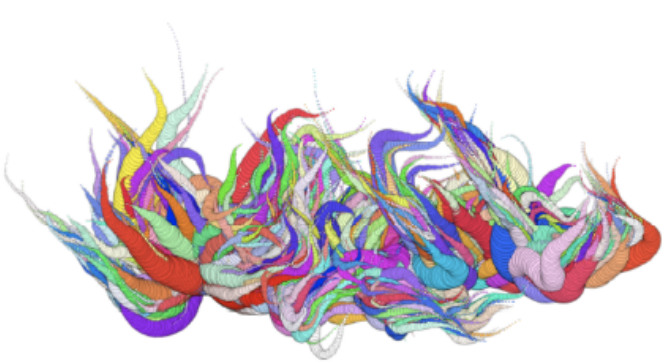
```
stroke(244, 37, 37, 60); // red
```

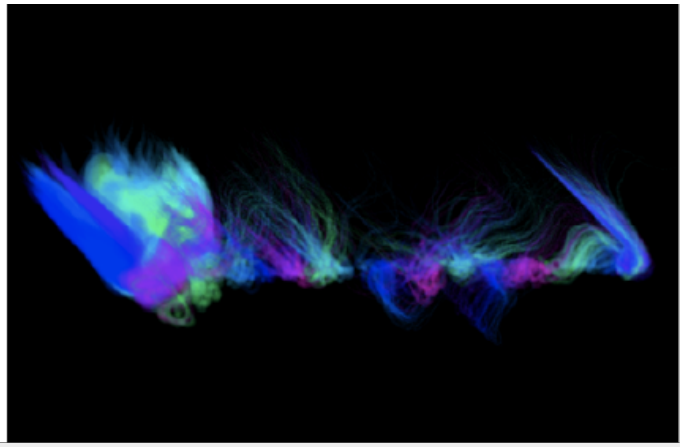
```
stroke(0, 0, 0);  // red
```

Figure 10: Tuning can result in inline documentation becoming outdated.

6.4 Extending Sketches with New Code

Our themes so far have not involved writing new code. In analyzing our data set, we found that many remixes make visually impactful additions by using Processing and p5.js-specific functionality. These creative coding libraries provide functionality to aid graphics



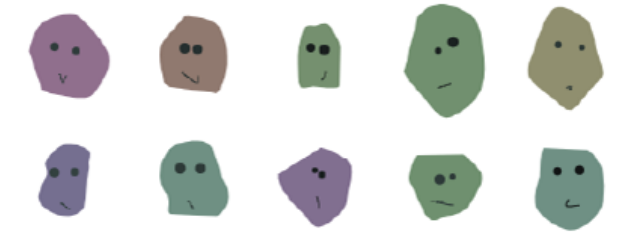


```
acc = new PVector(0,0);
lifeSpan = int(random(30, 90));
decay = random(0.75, 0.9);




c = color(random(255),random(255),255);
```

```
acc = new PVector(0,0);
lifeSpan = 90;
decay = 0.75;
this.h = h;
h += 0.5;
if (h > maxH) {
  h = minH+h-maxH;
}
c = color(h, 255, 255, 10);
```

Figure 11: Extensions involve adding new code in a remix. *Jason Labbe* (right) remixes *Raven Kwok*'s (left) sketch. The remix stays close to the original source material; in the code excerpt shown, the remix tunes several values and specifies color. The remix tunes and adds additional code in other locations.

```
vertex(xPosition, yPosition);
```



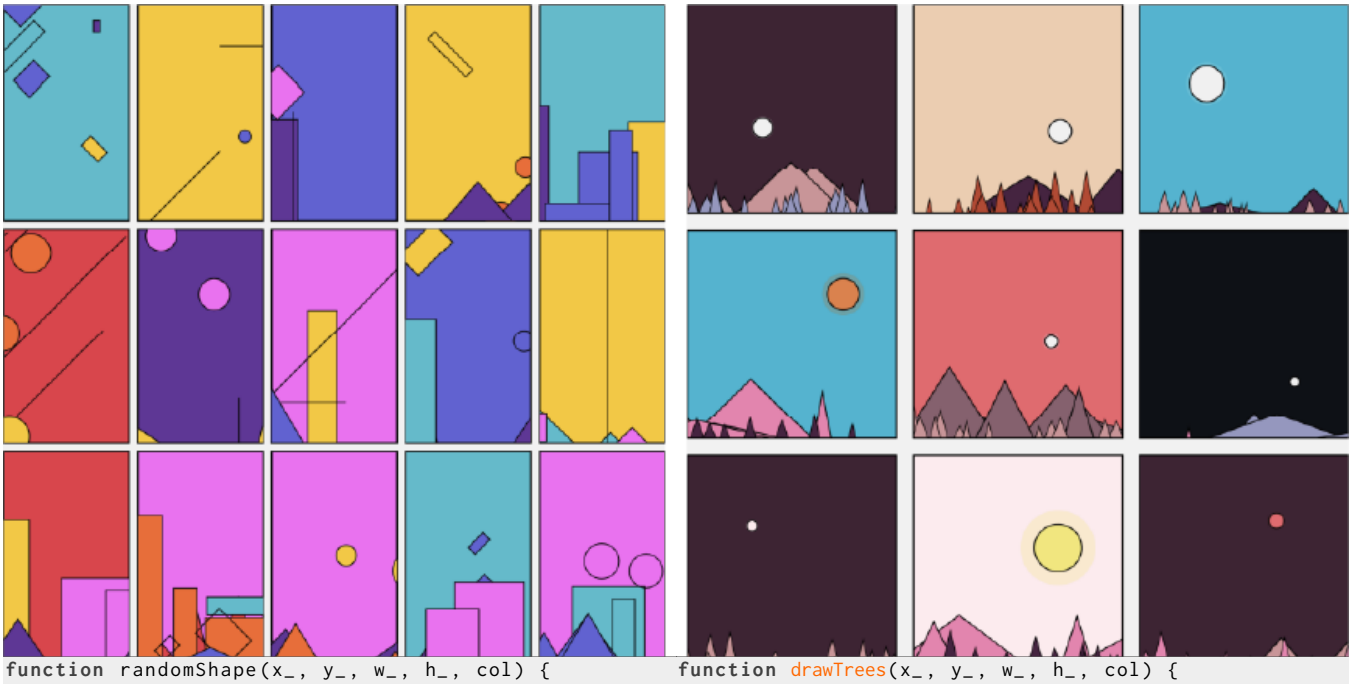
```
curveVertex(xPosition, yPosition);
```

Figure 12: The only change between *Sasha T.*'s antecedent (top) and *shrike*'s remix (bottom) is the use of the `curveVertex` command to join points using curved splines rather than straight lines. This results in rounded shapes in the face generator.

programming. Among these include functions to render lines (e.g. `line()`, `curveVertex()`) and shapes (e.g. `ellipse()`, `box()`) to the screen, change and blend colors (e.g. `fill()`, `stroke()`), and handle device input events (e.g. `mouseClicked()`, `keyIsDown()`). All of this functionality is tailored to the task at hand: creating visuals and interactivity using code. We define changes which require the creative coding library as *creative coding extensions*. For example, *shrike* changed a single function in *Sasha T.*'s face generator to make smooth, rounded curves (Figure 12). Instead of connecting points directly with `vertex()`, `curveVertex()` is another p5.js function which will generate a spline between points. By taking advantage of the creative coding library, the remix is able to make a small but visually substantial change to the sketch.

We distinguish creative coding extensions from changes which do not require specific use of the creative coding library. Instead, *generic extensions* rely only on the general purpose programming language and can be run without the use of the creative code library. Common examples of generic extensions in our data set include declaring new variables, writing custom functions and classes, and implementing control flow statements (e.g. `for` loops) to specify behavior. We distinguish between creative code and generic extensions to gain analytic insight in our content analysis; here, we consider sketches which make use of both.

Extensions can be used to achieve a desired result through targeted interventions. One example is shown in Figure 11. In remixing a sketch by *Raven Kwok*, *Jason Labbe* left an inline comment that they *"Changed how it renders to feel more stylized"*. To accomplish



```
function randomShape(x_, y_, w_, h_, col) {
  let grfx = createGraphics(w_, h_);
  let rnd = int(random(6));
  let num = int(random(1, 4));
  ...
  for (let i = 0; i < num; i++) {
    let w = random(5, w_ * 0.35);
    let h = random(5, h_ * 0.35);
    let x = (random(1.4)-0.2)*grfx.width;
    let y = (random(1.4)-0.2)*grfx.height;
    ...
}
```

```
function drawTrees(x_, y_, w_, h_, col) {
  let grfx = createGraphics(w_, h_);
  count = int(random(30));

  ...
  for (let i = 0; i < count; i++) {
    let w = random(2, 10);
    let h = w * random(2, 5);
    let x = (random(1.4)-0.2)*grfx.width;
    let y = (random(1.4)-0.2)*grfx.height;
    ...
};
```

Figure 13: *@Okazz* created a generated series of panels (left). In each, random circles, triangles, and quadrilaterals are drawn according to a specified set of rules. @JFrench reuses and expands parts of the original sketch to create landscapes within each panel (right).

this goal, the remix edits pre-exiting variables in the original particle simulation (i.e. tuning), adds a function to cycle through preset colors instead of using a random number generator (i.e. generic extending), and makes use of various built-in math functions (i.e. creative code extending). Much of the antecedent code is left in tact with specific sections changed to achieve the desired effect. Figure 11 shows a section of the remixed code. Specific variables are tuned to explicit values, and additional code has been added to set the color and size of the of the shapes drawn. These edits stay close to the source material to *"stylize"* the sketch in a new way.

In other examples, we see targeted changes used to add functionality to the antecedent sketch. Common examples include adding camera orbit controls, binding keystrokes to reset the elements of the sketch, or adding mouse interactivity. Author *Richard Bourne* has over 11,000 sketches on OpenProcessing, many of which are forks. Consulting their user page, we see they frequently remix sketches to add built-in functions for saving still images from the sketch when a user clicks their mouse. We finally note that remixes which make targeted extensions can occur by the same author as the original sketch. For example, Figure 14 (left) shows how *garabatospr* remixed their original sketch into a grid of outputs. Among other small edits, the remixed code abstracts elements of the original code into functions which can then be called multiple times.

Extensions can also reuse, repeat, and reinterpret the antecedent. *Okazz*'s sketch in Figure 13 (left) creates a grid of panels. Each panel is filled with a different set of random shapes. *JFrench* remixed this sketch to create generative landscapes. In the code excerpts shown, we can see how the same code which draws random triangles has been slightly modified to give the effect of trees; the same technique is used with larger triangles to create the mountains. While the code shown highlights how the remix repurposes this particular excerpt, the remix additionally makes larger changes to the overall code organization. For example, it also adds a function to style the sun

Figure 14: A set of antecedents (top) and remixes (bottom). In each, we see the remixes repurposing and repeating code from the antecedent. (Left) *garabatospr* remixes their own sketch to create an array of outputs. (Center) *Hans Peter* remixes *Sayama* to turn generative birds into owls. (Right) *Naoki Tsutae* remixes themselves to add circle packing to their collision-free paths.

and moon in the sky, ensuring exactly one circle is drawn. Similarly, *Hans Peter* reworks *Sayama*'s generative birds into owls (Figure 14 center), and *Naoki Tsutae* remixes themselves to add circle packing to their collision-free paths (Figure 14 right). Across these examples, we see the remix stays conceptually connected to its antecedent through reusing prior code, even in code-intensive changes.

The remixes considered till now have involved one antecedent and one remix. We see authors can explore multiple creative directions in a family of remixes. Figure 15 shows a subgraph of remixed sketches beginning with *"Square packing study"* by *Roni Kaufman*; however, we only show sketches by the original author. We see the author remixes the original sketch in three different ways. They then elaborate on the resulting sketches. We highlight two takeaways from this subgraph.

First, the relationship between sketches in distinct chains is obscured without a top-level view of the whole subgraph. Second, in lieu of presenting the code changes between each sketch, we note the edit distance between each remix and antecedent. The edit distance between two texts is quantified by the number of character insertion, deletions, and replacement must be made to transform between the two [29]. It is has been widely used in software engineering contexts as a metric of originality. We point out that the edit distance decreases over each chain. These measurements map to larger bursts of extending over the first generations of remixes, followed by shorter extensions and tuning. We noted in our network analysis that 49% of all remixed sketches are involved in self-remixing. Filtering the self-remix graph by subgraph size, we find that 48,600 sketches (14.4% of all remixed sketches) are a part of self-remix subgraphs with five nodes or more. This statistic speaks to the frequency with which authors manage families of versions through remixing.

7 MEASURING THE PREVALENCE OF REMIXING GENRES

Our themes provide an interpretive analysis of remixing behaviors in the OpenProcessing community. We seek to measure the prevalence of the remixing strategies in our data set. To do so, we conducted a content analysis. Table 1 reports the overall frequency with which we observe each strategy in a content analysis of 400 projects. Notably, tuning occurs in over half of all remixes. In our

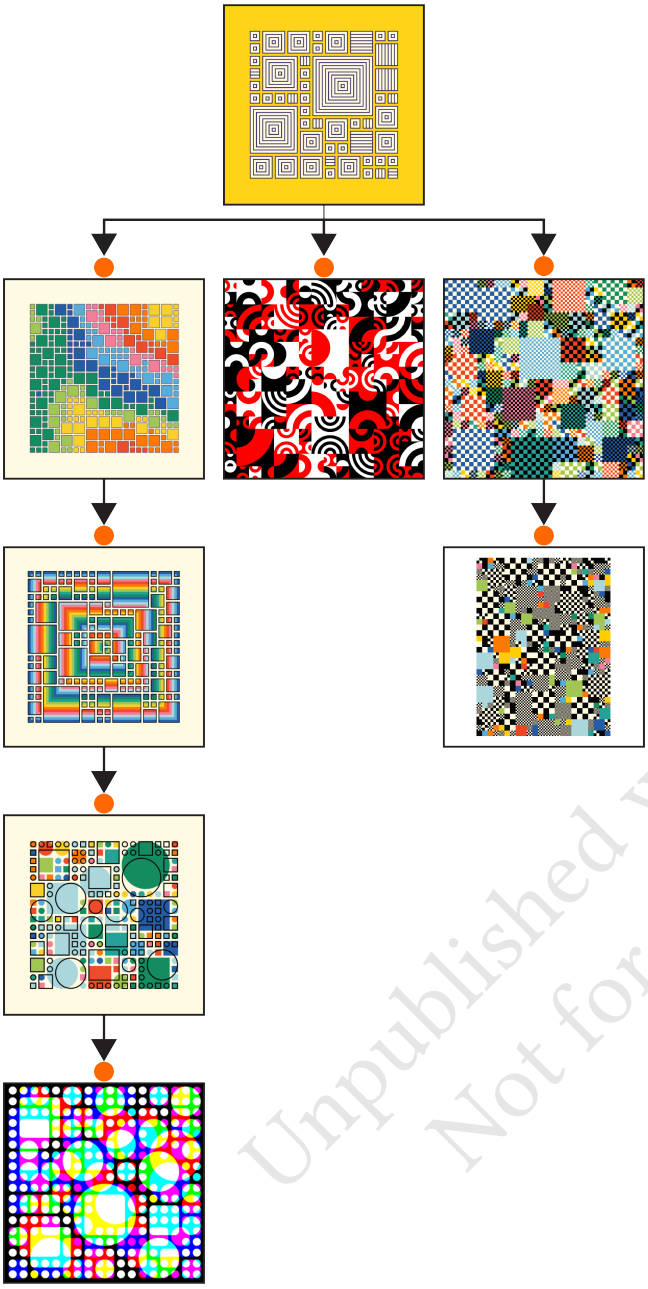


Figure 15: A self-remixing subgraph with sketches by *Roni Kaufman*. The original sketch is taken in three distinct directions. Each of these ideas is then further pursued. Components of this graph are also shown in Figure 1.

content analysis, we divide extending into the subcategories of creative code extensions and generic extensions, as discussed in Section 6.4. This distinction shows that code additions which makes use of the creative code library are more common than those which rely solely on the general purpose programming language. Finally, approximately a quarter of remixes are collections with no changes.

Theme	Frequency
Collecting	26.3%
Annotating	29.8%
Tuning	55.3%
Creative Code Extending	40.3%
Generic Extending	32.5%

Table 1: The frequency with which we observe each remixing strategy in a content analysis. With the exception of collecting, multiple strategies can be present in a given sketch.

While this is the least frequently occurring of our themes, we note that each of the other strategies might differ in scope and scale. All collections, on the other hand, are remixes with no changes made to the code.

8 DISCUSSION

Our analysis of OpenProcessing sheds light on a diversity of remixing practices in the community. Practitioners collect, annotate, tune, and extend code in ways which are difficult to discern without examining code directly. In this section, we discuss important takeaways from our work. (1) We reflect on lessons we can learn from OpenProcessing and our analysis, offering design provocations for HCI researchers interested in building systems which support creative code and related practices through remixing. (2) While our findings are intimately tied to creative coding, we discuss the possible benefits of our methodology in the study of other remix and coding contexts.

8.1 Design Provocations to Support Creative Community through Remixing

Our study of OpenProcessing provides insights relevant to the development of HCI systems which seek to support creative process and community. We outline design provocations derived from our analysis. These provocations include incorporating code diffs within the interface to rapidly discern changes, tagging remixes for refined discovery, supporting collaborative annotation of remixing graphs, and decreasing the relevant unit of analysis from sketches to individual lines of code when considering what makes a remix. For each, we highlight how such a feature might support individual creative coders as well as cultivate creative community. Rather than informing the design of OpenProcessing–which already makes design improvements guided by the interests of their community–we do so to bring empirical insight to prior HCI research. Recent research notes that only 25% of HCI systems which support creativity are made publicly available [10], and just 5% are intended to support a specific population [7]. OpenProcessing inverts this landscape, and we focus on lessons we can learn from an active community.

8.1.1 Diff in the Loop Remixing. In our analysis, we found it was impossible to know what changed in a remix without consulting the code for both the antecedent and remix. This can be tedious, particularly when it is unknown if there are any changes to be found and when changes are scattered among several files. "Diff in the Loop" explores visualizing differences in data sets to aid exploratory analysis in data science contexts [63]. Our research

approach to consult code diffs suggests ways to make differences between remixed code more easily legible. In particular, we can consider a live code diff view incorporated within the editor. With such a view, a user might open a pair of sketches to immediately see the differences in both code and visual output. For creative coders, this can help identify sections of code which produce desired visual effects.

8.1.2 Tagging Remixes. In addition to examining code diffs directly, the remixing strategies we identified can be used to tag remixes. Currently, discerning the changes made among remixes requires manually opening each sketch. By contrast, tags can be leveraged to filter remixes which are collections, only show remixes which extend the current one, or show all remixes which provide annotations to help make sense of the sketch. This feature can aid creative coders in the discovery of remixes of interest. Tagging posts when sharing a project is already supported in OpenProcessing and other similar online communities; our analysis suggests that tagging remixes in particular can aid exploration.

8.1.3 Collaborative Annotation of the Remixing Graph. While remixing is often presented as a method for collaborative peer production, our analysis suggests ways that current remixing interfaces limit collaboration. Linear navigation of remixes can hinder exploration by obscuring relationships between related sketches (as shown in Figure 15). To elaborate possible correctives, we recall the process-oriented annotations discussed in Section 6.2. While there is a rich body of prior work around understanding code comments to help programmers more easily find information [42, 51, 55], these works focus on professional software engineering contexts. Process-oriented comments like the ones we discuss exceed existing taxonomies. Creative coders' process-oriented comments are therefore closer to programmer note-taking, a comparatively less-explored area [16]. Creative code remixing communities make for productive sites to investigate such annotation systems as authors are consistently confronted with code written by others.

Going beyond previous recommendations to make the remixing graph visible to users, our analysis suggests we might promote collaboration by allowing authors to share text, images, and notes to annotate the remixing subgraph. In a similar vein, *Quickpose* makes the versioning graph visible and editable in a canvas to support version control requirements specific to creative coding [45]. Compared to managing versions in an individual's creative practice, we imagine public remix graphs can promote collaborative production of new sketches. In the context of DIY maker projects, Tseng [59] argues for process-oriented documentation tools which facilitate storytelling over product-oriented write-ups. Such tools can add depth to creative code sketches, making creative process as open and transparent as the code itself.

8.1.4 Smaller Units of Analysis. Our analysis surfaced productive connections between version control and remixing. While remixing inherits a parent-child relationship between antecedents and remixes from software forks, we see remixing often happens at the level of function arguments and chunks of code. This behavior aligns well with prior exploratory programming studies [24]. Our study therefore agrees with related work that similar micro-versioning tools would also be useful for creative coders [62]. Beyond version control for individual practice, we can consider ways that authors might remix smaller chunks of code. Since a remix can have only one parent, the remix graph cannot capture relationships to code snippets referenced from multiple sources. Considering functions, classes, and other code snippets smaller than a full sketch as remixable content can promote collaboration, with implications for both how practitioners remix and the high-level remixing graph characteristics.

8.2 Applicability to Other Remixing and Coding Contexts

In our analysis, we see how creative coders leverage remixing towards a variety of ends. What constitutes a remix is community-dependent. We see this reflected in community guidelines; Scratch does not allow authors to remix their own projects, whereas self-remixes constitute almost half of all remixes on OpenProcessing. This difference is critical for the activity in question: in the context of creative coding, we see self-remixing is a way to version sketches and manage process.

There is an opportunity, then, to investigate how other communities employ remixing in application-specific ways. Our identified strategies might be applied to other open remixing data sets such as Scratch. Scratch is primarily aimed at younger students– as a result, we expect the prevalence and use of remixing strategies to differ. As Scratch is a visual programming language, our qualitative coding methodology cannot map directly. In other visual arts communities, our strategies might fit off-the-shelf. Shadertoy is an online community dedicated to sharing shaders made with WebGL [44]. Shadertoy also permits forking existing projects, and thus a similar analysis can be undertaken. While aligned in its pursuit of creative and visual output, shader programming is quite different than the Javascript and Java based programming on OpenProcessing. A comparative analysis of different remixing communities can help discern productive idiosyncrasies specific to each. We reiterate the benefits that future interview-based studies can offer in this process. Our analysis has helped discern how remixing plays out on OpenProcessing, but understanding why create coders choose to remix would provide useful and complementary insights.

Our findings might also prove useful to understand how remixing is taken up in informal learning contexts. Dasgupta et al. [8] found that users who remix more on Scratch go on to use a larger variety of programming commands, and that exposure to computational concepts via remixing leads to a higher chance of using those concepts in later projects. Creative coding communities like OpenProcessing offer novel empirical settings to investigate related questions with a textual programming language. For example, do authors who tune go on to make more code-intensive extensions, or do annotations lead to larger programming repertoires? The strategies which we present might be operationalized to pursue these questions. Related research builds and uses an experimental editor to teach creative coding in physical classrooms [36]. For online communities, user-driven community resources tend to concentrate around a limited set of mainstream interests. This has the effect

of limiting sources of inspiration, thus unintentionally restricting participation by a broader audience [6]. Beyond participation, supporting a diversity of niche interests is critical to developing long-lasting community [3]. Future research into editors, remixing, and other platform features can expound the relationship between creative code and education.

9 CONCLUSION

Members of OpenProcessing reuse creative code in a diversity of ways. In this paper, we have worked to understand creative code remixing practices by pairing community network data with in-depth qualitative analysis of code changes. We see creative coders use remixes to collect artifacts, annotate process, version chunks of code, explore generative variations, build on the work of others, and manage personal practice. We showcased a range of community work, paying attention to the ways that creative code corroborates and complicates previous studies of remixing and code reuse in other programming contexts. Creative code platforms nurture and sustain active communities of artists, hobbyists, educators, and students. In line with HCI interests to support each of these domains, it is important to understand the successes of existing open source communities. In doing so, we can learn from and work with creative practitioners moving forward.

ACKNOWLEDGMENTS

We would like to thank Sinan Ascioglu for his work on OpenProcessing along with the broader OpenProcessing community for sharing their sketches. We'd like to specifically thank the 20 creative coders whose work is discussed here: *Roni Kaufman*, *caaatisgood*, *Taiki Saito*, *Owaun Scantlebury*, *Richard Bourne*, *Neill Bogie*, *Naoki Tsutae*, *Aaron Reuland (a_soluble_fish)*, *Naha*, *SamuelYAN*, *Trrrrrr*, *Sasha T. (@tequibo)*, *shrike*, *Raven Kwok*, *Jason Labbe*, *Okazz*, *JFrench*, *garabotospr*, *Hans Peter*, and *Sayama* . Many thanks to the p5.js and Processing contributors and community. We'd like to thank Hannah Twigg-Smith for her development on the open-source qualitative coding tool we used, SuperCoder 3000. Thanks also to Mako Hill and Jasper Tran O'Leary for feedback on this work. This research is supported by NSF Award 2007045 and the Gordon and Betty Moore Foundation.

REFERENCES

[1] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. 2009. Gephi: an open source software for exploring and manipulating networks. In *Proceedings of the international AAAI conference on web and social media*, Vol. 3. Association for the Advancement of Artificial Intelligence, USA, 361–362.

[2] Virginia Braun and Victoria Clarke. 2012. Thematic analysis. In *APA handbook of research methods in psychology, Vol 2: Research designs: Quantitative, qualitative, neuropsychological, and biological*. American Psychological Association, Washington, DC, US, 57–71. https://doi.org/10.1037/13620-004

[3] Leah Buechley and Benjamin Mako Hill. 2010. LilyPad in the Wild: How Hardware's Long Tail is Supporting New Engineering and Design Communities. In *Proceedings of the 8th ACM Conference on Designing Interactive Systems* (Aarhus, Denmark) *(DIS '10)*. Association for Computing Machinery, New York, NY, USA, 199–207. https://doi.org/10.1145/1858171.1858206

[4] Giorgos Cheliotis, Nan Hu, Jude Yew, and Jianhui Huang. 2014. The antecedents of remix. In *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*. Association for Computing Machinery, New York, NT, USA, 1011–1022.

[5] Giorgos Cheliotis and Jude Yew. 2009. An Analysis of the Social Structure of Remix Culture. In *Proceedings of the Fourth International Conference on Communities and Technologies* (University Park, PA, USA) *(C&T '09)*. Association for Computing Machinery, New York, NY, USA, 165–174. https://doi.org/10.1145/1556460.1556485

[6] Ruijia Cheng, Sayamindu Dasgupta, and Benjamin Mako Hill. 2022. How Interest-Driven Content Creation Shapes Opportunities for Informal Learning in Scratch: A Case Study on Novices' Use of Data Structures. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 228, 16 pages. https://doi.org/10.1145/3491102.3502124

[7] John Joon Young Chung, Shiqing He, and Eytan Adar. 2021. The Intersection of Users, Roles, Interactions, and Technologies in Creativity Support Tools. In *Designing Interactive Systems Conference 2021* (Virtual Event, USA) *(DIS '21)*. Association for Computing Machinery, New York, NY, USA, 1817–1833. https://doi.org/10.1145/3461778.3462050

[8] Sayamindu Dasgupta, William Hale, Andrés Monroy-Hernández, and Benjamin Mako Hill. 2016. Remixing as a Pathway to Computational Thinking. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work &; Social Computing* (San Francisco, California, USA) *(CSCW '16)*. Association for Computing Machinery, New York, NY, USA, 1438–1449. https://doi.org/10.1145/2818048.2819984

[9] Scott D. Fleming, Chris Scaffidi, David Piorkowski, Margaret Burnett, Rachel Bellamy, Joseph Lawrance, and Irwin Kwan. 2013. An Information Foraging Theory Perspective on Tools for Debugging, Refactoring, and Reuse Tasks. *ACM Trans. Softw. Eng. Methodol.* 22, 2, Article 14 (mar 2013), 41 pages. https://doi.org/10.1145/2430545.2430551

[10] Jonas Frich, Lindsay MacDonald Vermeulen, Christian Remy, Michael Mose Biskjaer, and Peter Dalsgaard. 2019. Mapping the Landscape of Creativity Support Tools in HCI. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–18. https://doi.org/10.1145/3290605.3300619

[11] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. Exploring network structure, dynamics, and function using NetworkX. (Aug. 2008), 11–15.

[12] Eszter Hargittai and Christian Sandvig. 2016. *When Should We Use Real Names in Published Accounts of Internet Research?* The MIT Press, Cambridge, MA, USA, 243–258.

[13] Andrew Head, Caitlin Sadowski, Emerson Murphy-Hill, and Andrea Knight. 2018. When Not to Comment: Questions and Tradeoffs with API Documentation for C++ Projects. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE '18)*. Association for Computing Machinery, New York, NY, USA, 643–653. https://doi.org/10.1145/3180155.3180176

[14] Benjamin Mako Hill and Andrés Monroy-Hernández. 2013. The cost of collaboration for code and art: Evidence from a remixing community. In *Proceedings of the 2013 conference on Computer supported cooperative work*. Association for Computing Machinery, New York, NY, USA, 1035–1046.

[15] Benjamin Mako Hill and Andrés Monroy-Hernández. 2013. The Remixing Dilemma: The Trade-Off Between Generativity and Originality. *American Behavioral Scientist* 57, 5 (2013), 643–663. https://doi.org/10.1177/0002764212469359 arXiv:https://doi.org/10.1177/0002764212469359

[16] Amber Horvath, Brad Myers, Andrew Macvean, and Imtiaz Rahman. 2022. Using Annotations for Sensemaking About Code. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology* (Bend, OR, USA) *(UIST '22)*. Association for Computing Machinery, New York, NY, USA, Article 61, 16 pages. https://doi.org/10.1145/3526113.3545667

[17] Philip N. Howard. 2002. Network Ethnography and the Hypermedia Organization: New Media, New Organizations, New Methods. *New Media & Society* 4, 4 (2002), 550–574. https://doi.org/10.1177/146144402321466813 arXiv:https://doi.org/10.1177/146144402321466813

[18] Steven J. Jackson and Laewoo Kang. 2014. Breakdown, Obsolescence and Reuse: HCI and the Art of Repair. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Toronto, Ontario, Canada) *(CHI '14)*. Association for Computing Machinery, New York, NY, USA, 449–458. https://doi.org/10.1145/2556288.2557332

[19] Jennifer Jacobs, Joel Brandt, Radomír Mech, and Mitchel Resnick. 2018. Extending Manual Drawing Practices with Artist-Centric Programming Tools. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3173574.3174164

[20] Jennifer Jacobs, David Mellis, Amit Zoran, Cesar Torres, Joel Brandt, and Theresa Jean Tanenbaum. 2016. Digital Craftsmanship: HCI Takes on Technology as an Expressive Medium. In *Proceedings of the 2016 ACM Conference Companion Publication on Designing Interactive Systems* (Brisbane, QLD, Australia) *(DIS '16 Companion)*. Association for Computing Machinery, New York, NY, USA, 57–60. https://doi.org/10.1145/2908805.2913018

[21] Laewoo Kang, Steven Jackson, and Trevor Pinch. 2022. The Electronicists: Techno-aesthetic Encounters for Nonlinear and Art-based Inquiry in HCI. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–17.

[22] Laewoo (Leo) Kang, Steven J. Jackson, and Phoebe Sengers. 2018. Intermodulation: Improvisation and Collaborative Art Practice for HCI. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3173574.3173734

[23] Andrew Keen. 2007. *The cult of the amateur: How today's Internet is killing our culture and assaulting our economy.* Broadway Business, New York, NY, USA.

[24] Mary Beth Kery, Amber Horvath, and Brad Myers. 2017. Variolite: Supporting Exploratory Programming by Data Scientists. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. Association for Computing Machinery, New York, NY, USA, 1265–1276. https://doi.org/10.1145/3025453.3025626

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

[26] Kenneth C Knowlton. 1970. Explor-a generator of images from explicit patterns, local operations, and randomness. In *Proceedings of 9th Meeting of UAIDE*. 544–583.

[27] Amy J. Ko, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, Henry Lieberman, Brad Myers, Mary Beth Rosson, Gregg Rothermel, Mary Shaw, and Susan Wiedenbeck. 2011. The State of the Art in End-User Software Engineering. *ACM Comput. Surv.* 43, 3, Article 21 (apr 2011), 44 pages. https://doi.org/10.1145/1922649.1922658

[28] Lawrence Lessig et al. 2008. *Remix: Making art and commerce thrive in the hybrid economy.* Penguin, London, England.

[29] Vladimir I. Levenshtein. 1965. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics. Doklady* 10 (1965), 707–710.

[30] Golan Levin and Tega Brain. 2021. *Code as Creative Medium: A Handbook for Computational Art and Design.* MIT Press, Cambridge, MA, USA.

[31] 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 (CHI '21)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3411764.3445682

[32] Zach Lieberman, Theo Watson, and Arturo Castro. 2021. openFrameworks. https://openframeworks.cc/about

[33] Lingdong Huang. 2022. srcsnap. https://github.com/LingDong-/srcsnap

[34] Lauren McCarthy, Casey Reas, and Ben Fry. 2015. *Getting started with P5. js: Making interactive graphics in JavaScript and processing.* Maker Media, Inc., USA.

[35] Lauren Lee McCarthy, Thomas Hughes, and Golan Levin. 2021. Open Source Software Toolkits for the Arts (OSSTA): a Convening. https://github.com/CreativeInquiry/OSSTA-Report

[36] Andrew M Mcnutt, Anton Outkine, and Ravi Chugh. 2023. A Study of Editor Features in a Creative Coding Classroom. 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 121, 15 pages. https://doi.org/10.1145/3544548.3580683

[37] Andrés Monroy-Hernández, Benjamin Mako Hill, Jazmin Gonzalez-Rivero, and danah boyd. 2011. Computers Can't Give Credit: How Automatic Attribution Falls Short in an Online Remixing Community. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vancouver, BC, Canada) *(CHI '11)*. Association for Computing Machinery, New York, NY, USA, 3421–3430. https://doi.org/10.1145/1978942.1979452

[38] Anders Mørch. 1997. Three levels of end-user tailoring: Customization, integration, and extension. *Computers and design in context* 1997 (1997), 61.

[39] Eduardo Navas. 2014. *Remix theory: The aesthetics of sampling.* Birkhäuser, Switzerland.

[40] Kimberly A Neuendorf. 2017. *The content analysis guidebook.* sage, USA.

[41] Lora Oehlberg, Wesley Willett, and Wendy E Mackay. 2015. Patterns of physical design remixing in online maker communities. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 639–648.

[42] Yoann Padioleau, Lin Tan, and Yuanyuan Zhou. 2009. Listening to Programmers Taxonomies and Characteristics of Comments in Operating System Code. In *Proceedings of the 31st International Conference on Software Engineering (ICSE '09)*. IEEE Computer Society, USA, 331–341. https://doi.org/10.1109/ICSE.2009.5070533

[43] Christiane Paul, Carol Mancusi-Ungaro, Melva Bucksbaum, and Clémence White. 2018. Programmed: Rules, Codes, and Choreographies in Art, 1965–2018. https://whitney.org/exhibitions/programmed

[44] Inigo Quilez and Pol Jeremias. 2013. Shadertoy. https://www.shadertoy.com/

[45] 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*. 1–18.

[46] Casey Reas and Ben Fry. 2014. *Processing: A Programming Handbook for Visual Designers and Artists* (2 ed.). MIT Press, Cambridge, MA, USA.

[47] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. 2009. Scratch: programming for all. *Commun. ACM* 52, 11 (2009), 60–67.

[48] Adam Rule, Aurélien Tabard, and James D. Hollan. 2018. Exploration and Explanation in Computational Notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3173574.3173606

[49] Lillian Schwartz. 1973. Papillon.

[50] Lillian Schwartz. 2013. Oral History of Lillian F. Schwartz.

[51] Yusuke Shinyama, Yoshitaka Arahori, and Katsuhiko Gondow. 2018. Analyzing Code Comments to Boost Program Comprehension. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. 325–334. https://doi.org/10.1109/APSEC.2018.00047

[52] Sinan Ascioglu. 2018. OpenProcessing. https://www.wiredpieces.com/openprocessing/

[53] Katherine W Song and Eric Paulos. 2021. Unmaking: Enabling and Celebrating the Creative Material of Failure, Destruction, Decay, and Deformation. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–12.

[54] Sruti Srinivasa Ragavan, Sandeep Kaur Kuttal, Charles Hill, Anita Sarma, David Piorkowski, and Margaret Burnett. 2016. Foraging Among an Overabundance of Similar Variants. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '16)*. Association for Computing Machinery, New York, NY, USA, 3509–3521. https://doi.org/10.1145/2858036.2858469

[55] Daniela Steidl, Benjamin Hummel, and Elmar Juergens. 2013. Quality analysis of source code comments. In *2013 21st International Conference on Program Comprehension (ICPC)*. 83–92. https://doi.org/10.1109/ICPC.2013.6613836

[56] 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. https://doi.org/10.1145/3555756

[57] the Processing Foundation. 2021. p5.js. https://p5js.org/

[58] Jer Thorp. 2014. Art at the Edge of Tomorrow. https://blprnt.medium.com/art-at-the-edge-of-tomorrow-b78ad9302abe

[59] Tiffany Tseng. 2016. Build in progress: Building process-oriented documentation. In *Makeology*. Routledge, USA, 237–254.

[60] Tiffany Tseng and Mitchel Resnick. 2014. Product versus Process: Representing and Appropriating DIY Projects Online. In *Proceedings of the 2014 Conference on Designing Interactive Systems* (Vancouver, BC, Canada) *(DIS '14)*. Association for Computing Machinery, New York, NY, USA, 425–428. https://doi.org/10.1145/2598510.2598540

[61] E. van Emden and L. Moonen. 2002. Java quality assurance by detecting code smells. In *Ninth Working Conference on Reverse Engineering, 2002. Proceedings.* 97–106. https://doi.org/10.1109/WCRE.2002.1173068

[62] 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. https://doi.org/10.1145/3544549.3585743

[63] April Yi Wang, Will Epperson, Robert A DeLine, and Steven M Drucker. 2022. Diff in the loop: Supporting data comparison in exploratory data analysis. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 1–10.

[64] Stanley Wasserman, Katherine Faust, et al. 1994. *Social network analysis: Methods and applications.* Cambridge university press, Cambridge, England.

[65] Scratch Wiki. 2022. Remix - Scratch Wiki. https://en.scratch-wiki.info/wiki/Remix