

Art in Humanity's Code

Benoit Baudry*, Yogya Gamage*, Nadia Gonzalez Fernandez*, Lena MK*, Roxana Pena Mendieta*, Rafaela Souza Pinter*, Stefano Zacchiroli†

*Université de Montréal, Montréal, Canada

† LTCI, Télécom Paris, Institut Polytechnique de Paris, Palaiseau, France

*{firstname.lastname}@umontreal.ca †stefano.zacchiroli@telecom-paris.fr

Abstract—Artist-led open-source libraries such as `Processing` or `openFrameworks` have had a major impact on artists and designers who use code as a creative medium. In this work, we conduct the first large-scale empirical study of public code repositories that use these libraries. Our study dives into 1,613,571 code repositories collected from the Software Heritage archive. Blending quantitative and qualitative methods, we investigate the diversity of practices and practitioners regarding code hosting, geographical distribution, and the characteristics and purposes of code-based creative works. Key findings include evidence of the worldwide practice of generative art and creative coding, as well as the adoption of these practices in both education and across creative industries. We illustrate these findings with concrete examples of repositories and contributor profiles around the globe, spanning the spectrum of code-based art, from visual to sound art. These results lay the groundwork for educators, researchers, and artists who investigate code and creative practices to sustain diversity in the construction of our digital world.

I. INTRODUCTION

Since the early days of computers and software, artists have experimented with the creative potential of these media [1], [2]. Vera Molnar pioneered the use of software to systematically explore the formal possibilities and variations of an image composition, shape, color, or scale, as opposed to the popular belief that “art simply happens” [3]. Sýkora used software to explore multiple combinations to assemble abstract geometrical shapes [4], while Cornock and Edmonds foresaw the role of software for interactive artworks [5]. These early adopters of code as an artistic medium appreciated the high precision and power of computers, leveraging their ability to methodically and rapidly generate variants of an image for creative purposes. Six decades after those initial experiments, the use of code for artistic ends has evolved from pure formal exploration into a wide range of art practices. Artists from all over the world build software systems that autonomously perform a part of artistic creation at a rate and in a space vastly different from our direct perceptual experience [6].

Open-source software has played an essential role in the growth of these artistic practices. In electronic music communities, the main projects were Puckette’s development of `Pure Data` in the early 1990s [7], as well as the first versions of `SuperCollider` in 1996 [8]. In the design and visual arts communities, Maeda’s *Design by Numbers* [9] led to the development of `Processing` by Reas and Fry in the early 2000s [10]. This supported the emergence of a whole new community of visual artists through an open-source language

```
let numRect = 5;
let marco = 20;
function draw() {
  translate(marco, marco);
  for (let i=0; i<numRect; i++) {
    for (let j=0; j<numRect; j++) {
      let col = Math.floor(random(2));
      switch(col) {
        case 0:
          fill(0,100,60,170); break;
        case 1:
          fill(0,100,100,170); break;
      }
      let move = random(7);
      let inX = move+(width-2*marco)/numRect*i;
      let inY = move+(height-2*marco)/numRect*j;
      rect(inX, inY, width/numRect-10, height/numRect-10);
    }
  }
}
```

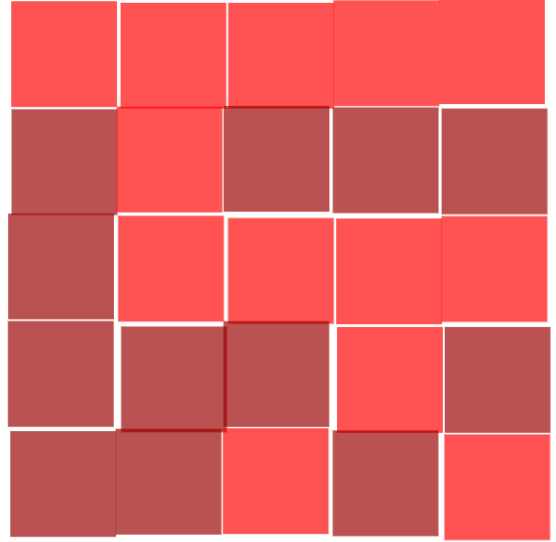


Fig. 1: Example of a program that generates art. Each time the program is executed it draws a unique artwork. It is inspired by 25 carrés (Vera Molnar, 1991). The code was developed in 2016, it is publicly available on GitHub and archived at [swlh:1:rev:2d5ba602e04d55664070b7d6f6297123021fcab5](https://github.com/25carrés).

and editor tailored to their needs for the exploration of visual and interactive arts. `Processing`, `openFrameworks` [11], and `p5.js` [12] are open-source libraries that have played a key role in the development and sustained practices of generative art [13] over the past two decades. These libraries have also been used by software developers to create artistic, expressive, or engaging outputs. Such practices go beyond pure functionality and utility and are commonly referred to as creative coding [14].

Figure 1 illustrates a concrete example of using code to create a generative artwork. At the top of the figure, there is a program that draws a different image every time it runs. This program uses the `Processing` library. The `draw()` function contains two nested loops that divide the canvas into a grid, and `move` is a random variable that defines an offset to draw each square.

Our work analyzes the diversity of practices in public code repositories that host such pieces of generative art and creative coding projects. In order to make our search for art-related code as inclusive as possible and to avoid excluding code that is no longer available online, we mine code repositories from the Software Heritage archive [15] instead of selecting a specific hosting platform such as GitLab or GitHub. Furthermore, to distinguish art-related code repositories from others, we focus our study on repositories that explicitly use an open source library which is dedicated to generative art.

We collect 1,613,571 art-related code repositories from Software Heritage. This remarkable number of repositories is evidence of the vibrant international activity surrounding generative art and creative coding. It is also a challenge for analysis. Accordingly, we define four research questions that explore this dataset on four different scales. First, we analyze the whole dataset to determine which code hosting platforms art contributors use, and we inspect which repositories are still online. Second, we select a subset of 10% of the repositories to study the geographical distribution of their main contributors. Third, we collect 1% of the `p5.js` repositories and use an LLM to automatically characterize the creative practices found in `p5.js` sketches. Finally, we focus on the top 42 accounts that hold the largest number of repositories, and manually analyze their content and purpose with respect to generative art and creative coding.

Previous work at the intersection of software engineering and art has either proposed novel techniques to support generative art practices [16], [17], [18] or relied on interviews and case studies to empirically examine the key features of software engineering systems for the arts [19], [20]. In contrast and as a complement to this work, we present the first large-scale empirical study of open-source repositories for the arts. Our key results are as follows: `p5.js` and `Processing` play an essential role in generative art and creative coding; Software Heritage has already saved 84,666 repositories that disappeared after hosting services shut down or authors removed them; the analysis of 154,832 repositories allowed us to unveil contributors in 124 countries; the automatic analysis of 17,875 `p5.js` sketches reveals that creative coders take advantage of the library’s versatility to develop very diverse works, from randomized still images to interactive audiovisual pieces; and the systematic analysis of the top 42 contributing accounts confirms three main purposes for creative coding in public repositories: computer science education, art education, and art. These new observations provide evidence and case studies for researchers in human aspects of software engineering, the conservation of creative code and software engineering for the arts, as well as for practitioners in the areas of art education, computer science education, and open-source development for the arts. The key contributions of this work are as follows:

- We present the first large-scale empirical study of 1,613,571 public code repositories related to generative art and creative coding.
- Evidence that (i) a portion of this code has already disappeared and archives such as Software Heritage play a key role in its conservation; (ii) practitioners who use code as an artistic and creative medium are found all around the world; (iii) `p5.js` users take advantage of the library’s versatility to develop a wide variety of works; and (iv) art-related code can have three main purposes: education in art, education in computer science, or the development of art installations and new media artworks;
- An open dataset of source code repositories related to generative art and creative coding on [Zenodo](#) with all code and data for this study on [GitHub](#).

II. RELATED WORKS

Empirical studies of Computational Art Practices Several studies have examined how artists use algorithms, programming environments, and computational systems as creative media. Wu et al. [21] review artificial life algorithms that generate visual art and propose a taxonomy for analyzing such artworks; while Verano et al. [22] interview five creative coders in order to understand their experiences across different stages of the creative development process. Other studies focus more on code practices, as is the case for Subbaraman et al. [23], who analyze how creative coding artists reuse and transform existing code in `OpenProcessing`; and in our previous work [24], we construct a dataset of open-source projects used by media artists. Related work has also explored software practices in generative music. Islam et al. [25] analyze the Pure Data (PD) visual language to identify programming patterns that can support users’ development practices, while Bogdan et al. [26] mine GitHub repositories of open-source Virtual Studio Technology (VST) plugins, to extract the engineering characteristics of these projects. Payne et al. [27] study the importance of online forums for live coders, focusing on how artists share code, discuss challenges, and support each other in Tidal Cycles and Sonic Pi communities. These studies highlight the importance of examining how artists use emerging technologies to create new forms of digital arts. We contribute to this body knowledge with a study at a very large scale and a focus on practitioners and their intention when using code.

Empirical Studies of Open-Source Contributors Several works focus on open-source contributors. The growth and availability of public code repositories have facilitated empirical studies of open-source contributors, their roles, practices, and community dynamics. Milewicz et al. [28] combine repository-level metrics with interviews to describe the roles that contributors play in open-source scientific software projects. Trinkenreich et al. [29] survey developers and contributors to examine the circumstances that shape members’ sense of belonging in a team, identifying work appreciation and psychological safety as fundamental factors. Other studies focus on diversity within open-source communities. Hyrynsalmi et al. [30] emphasize the need to make software communities more diverse and inclusive, especially for underrepresented groups, while Vasilescu et al. [31] examine

the positive correlation between gender diversity and team productivity. Our work contributes to this body of knowledge with a special focus on art-related code repositories, and with novel results about the geographical distribution of creative open-source contributors.

Software tools for artists and creative programmers

Tools for artists and programmers are still being created in order to enhance their experience with code for creative purposes. Angert et al. [18] introduce Spellburst, an LLM-powered creative coding environment to generate and explore variations of visual programs through prompt-based interaction, branching, and merging. Fredericks et al. [32] investigates how evolutionary algorithms can amplify the generation of artwork variants. Li et al. [33] propose Demystified Dynamic Brushes (DDB) to facilitate the exploration of how numerical inputs influence the state of visual artworks, and Sacadura [34] propose a solution to explore the space of parameters combinations. Islam et al. [16] propose TriGraph, a graph-based probabilistic model for code completion in Pure Data. Our observations about the purposes and practices of generative art and creative coding can inform future efforts to expand the use of code for artistic creation.

Software engineering challenges in Software-Based Art

At the intersection of software research and generative art practices, several studies have examined the software engineering challenges that artists face while creating their pieces [35]. Trifonova et al. [20] document the specific engineering challenges regarding requirements, architecture and design, testing, validation, and maintenance that occur when developing new media artworks. These challenges are also discussed in community events such as the Open Source Software Toolkits for the Arts (OSSTA): a Convening (2018) [19] and the Open Source Arts Contributor’s Conference (OSACC, 2023) [36]. In these events, the attendants articulate and address the challenges of sustainability, maintenance, accessibility and funding for open source software for and by artists [37].

Creative coding and education Sandberg et al. [38] show that educators have adopted creative coding as a way to teach programming, while at the same time developing creativity and innovation. Woo et al. [39] examine how students use computational thinking and computer science concepts when coding animated narratives. Greenberg et al. [40] use Processing to teach programming concepts through creative activities.

The same environments are used to teach programming to art students. Levin and Brain [41] have collected a series of possible assignments, together with interviews with educators who share their experiences teaching programming to art students. Sykes [42] examines how creative coding is taught in art schools, and proposes new pedagogical approaches to help students better grasp programming concepts for their artistic practice. McNutt et al. [43] analyze how tools and technologies for making art with code affect students’ learning and artistic development. Knochel [44] argues that computational thinking and critical digital making should be integrated into art curricula, where code is an essential pillar of the creative expression. Similarly, Dufva et al. [45] explore how to bridge the gap between artistic expression and computational education in order to have more engaging classes for students.

III. BUILDING A CORPUS OF ART-RELATED PUBLIC CODE REPOSITORIES

We mine source code repositories related to arts from the Software Heritage archive [15]. In this section, we introduce our method and curation process for selecting the repositories for our study.

A. Signals of art in source code repositories

The key challenge to mine art-related code is to distinguish between source code that has a creative intention from source code that has any other motivation. Generative art and creative coding can be developed in any programming language and totally independently of specific libraries or art-related toolkits. Consequently, any C or Rust project in Software Heritage could be related to art or have a totally different purpose. We address this challenge with a conservative strategy: we mine projects that explicitly use one of the libraries that are meant for generative artists. Such libraries can focus on visual arts (e.g., **Processing**, **openFrameworks**, or **nannou**), or on sound synthesis (e.g., **SuperCollider** or **Faust**). Some software libraries focus on live performances (e.g. **Tidal** or **Hydra**). Other libraries provide artists with graphical programming environments (e.g. **TouchDesigner**, **Pure Data** or **Max**). Some artists also develop on top of 3D or game engines such as **Blender**, **Unity** or **Unreal**. Yet, to support the mining process at the scale of the hundreds of millions of repositories in Software Heritage, occupying more than 2 PiB of raw space, we determine the usage of one of these libraries only by the presence of specific files or file extensions. It is not feasible to check file *content* at this scale.

Based on these constraints, we consider a code repository to be art-related if it matches one of the “signals” documented in Table I. Our mining process focuses on seven software libraries and programming environments, which artists use to create and produce software-based artworks. `p5.js` and `openFrameworks`, shown in the top two rows of Table I, are popular JavaScript and C++ libraries, respectively, mainly used for visual arts. For these two ecosystems, the presence of specific files in a source code repository indicates their use. To use `p5.js`, artists and creative coders can store either the `p5.js` bundle or its minified version directly in the repository. In the case of `openFrameworks`, the files named `ofApp.cpp` and `ofApp.h` are conventionally used to structure/build an `openFrameworks` project.¹

We detect the usage of the 5 other libraries and environments based on file extensions. `Processing` is a very popular Java library for artists. While a regular third-party dependency can be used with Maven or Gradle, artists and creative coders can also develop code on top of `Processing` using a dedicated code editor, in which case the files for the artwork are saved with the extension “`.pde`”. `SuperCollider` is an audio programming language built on top of C++ and all `SuperCollider` files have the “`.scd`” extension. The last 3 environments are node-based programming environments. `TouchDesigner` is a proprietary runtime, but the networks,

¹see [How openFrameworks works](#)

TABLE I: Programming frameworks for the arts and their corresponding filename patterns

Art Programming Env.	Language	Filename pattern
p5.js	JavaScript	"p5.js", "p5.min.js"
openFrameworks	C++	"ofApp.cpp", "ofApp.h"
Processing	Java	".pde"
SuperCollider	C++	".scd"
TouchDesigner	node-based + Python	".toe", ".tox"
nodebox	node-based + Python or Clojure	".ndbx"
vvvv	node-based + C#	".v4p"

configurations, and operators for audiovisual installations are saved in files with the ".toe" and ".tox" extensions and can be shared as open source. **nodebox** and **vvvv** are node-based environments for graphical and live programming, saved in specific ".ndb" and ".v4p" files.

B. Finding Art Code Repositories in Software Heritage

Software Heritage (SWH) [15] archives software source code from public sources, including major collaborative development platforms such as GitHub and multiple GitLab instances. The source code in Software Heritage is archived with its full development history, captured by modern distributed version control systems (VCS) such as Git; if an archived Git repository was to disappear, it could be recreated from the archive. The Software Heritage data model is a single, fully de-duplicated Merkle graph structure, which corresponds intuitively to the global VCS of public code. At the time of writing (May 2026), Software Heritage has archived more than 400 million code repositories from more than 5,000 code hosting platforms, totaling almost 30 billion unique source code files and occupying more than 2 PiB of storage.

This is our main motivation to search for art-related code in Software Heritage instead of GitHub or GitLab.com only. First, it includes art-related code repositories that are currently unavailable online, e.g., because the hosting service shut down, but are archived on Software Heritage. Second, it gives access to source code that was shared before the launch of GitHub (2008), e.g., **Processing** art works published in the early 2000s on platforms such as **SourceForge** (launched in 1999). Indeed, the archive contains code that predates its own creation. Such code is obtained either via successive migrations through online repositories, or dedicated digital archaeology action. For example, the archive includes the code of the Apollo space missions Guidance Computer (AGC), a milestone of our software heritage developed in the late 1960s under the leadership of Margaret Hamilton.

We search for art-related repositories within this large volume of code via the filename patterns listed in Table I. We start from the Software Heritage graph dataset [46], version 2025-05-18,² using its compressed representation [47]. This allows us to load into memory (on a beefy server) the entire VCS history of all repositories archived by Software Heritage and to mine them via a Rust API based on WebGraph [48].

In order to identify repositories that contain art-related filenames, we: (1) iterate over all repositories, called "origins"

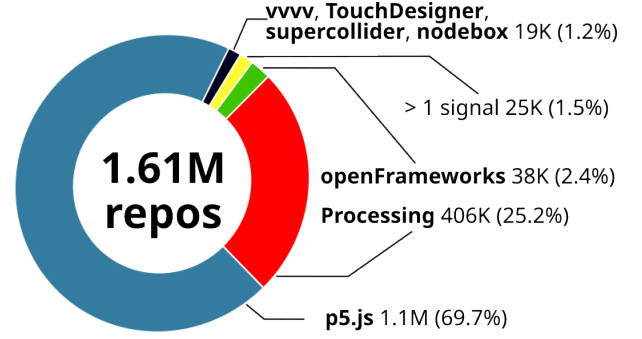


Fig. 2: Distribution of the art software libraries among the 1,613,571 source code repositories mined in Software Heritage.

in Software Heritage terminology; (2) inspect the most recent archived snapshot of each repository; (3) consider only the most recent commit on the main branch of the repository, the equivalent of Git HEAD at archival time; and (4) recursively list the files and directories of that commit starting from the root directory, with a recursion limit of 1,024 directories to avoid Git/tar bombs.

If, at the end of the recursion, we encounter one or more paths matching the patterns of Table I, we flag the repository as potentially containing art-related code and record it along with the mined metadata, including the repository URL, the number of matches, and, for each match, the path name and intrinsic identifier. Each matching repository is saved as an NDJSON entry, as shown below. The example is edited and reformatted for brevity and readability:

```
{
  "ori_url": "https://github.com/imclab/oauthp5",
  "match_count": 13,
  "matches_info": [
    { "swh:1:cnt:a92975aa3a6d9bd0d37d7de0606e8a1b2da76c39",
      "OpenPathsExample.pde" },
    { "swh:1:cnt:32b2e44f256e7b278516e7f30792d45bdc4b5abe",
      "Twitter3LegExample.pde" },
    ...
  ]
}
```

We release the implementation of this mining step as part of the open-source Software Heritage repository-mining toolkit.³

At the end of this mining step, we retrieve 1,645,775 repositories that match one or more art-related code signals in their most recently archived revision in the Software Heritage archive.

C. Consolidated Dataset of Source Code Origins

From the raw output of the mining process described above, we perform the following steps to clean the dataset. First, we normalize and de-duplicate repository URLs to ignore differences in protocol (e.g., http vs. https) and optional .git suffixes, and case sensitivity. Second, we noticed an ambiguity in our file extension signals: .scd files may correspond to either the SuperCollider source code or Unix man pages. To triage these cases, we implement a

²Retrieved from <https://datasets.softwareheritage.org/graphs/compressed/#2025-05-18-compressed>

³<https://gitlab.com/zacchiro/swh-repo-mining/>, binary `find-files`. We use commit `swh:1:rev:d9412b0d5c9ebe4c907b27aa84084382309ce92e` in our experiments.

custom script using two sets of keywords that are certainly present in SuperCollider and in Unix man pages. For example, files containing terms such as NAME, SYNOPSIS, or DESCRIPTION, among others, are classified as man pages, whereas files containing terms such as SynthDef, Synth, or ControlSpec, among others, are classified as SuperCollider source code files.

The resulting dataset, which forms the foundation for our empirical study, includes 1,613,571 repositories. Figure 2 shows the distribution of signals in these repositories. The majority of the repositories match the p5.js signals (69.7%), followed by Processing (25.2%) and openFrameworks (2.4%). The remaining repositories match more than one signal (1.5%) or match SuperCollider, TouchDesigner, nodebox, or vvvv signals (1.4%).

IV. METHODOLOGY

A. Research Questions

We study the 1,613,571 repositories at different scales, as illustrated in Figure 3, and answer the following research questions.

RQ1: Where is art-related code hosted and how much of it disappeared over time?

For this first research question, we analyze the complete dataset. We determine where the artists and creative coders host their public source code. We also investigate to what extent all the repositories that we have mined in the Software Heritage archive are still accessible online. Indeed, some code hosting platforms have disappeared over the years, and they might have hosted art-related code. It is also possible that contributors decide to delete some work. In both cases, our analysis reveals the role that an archive like Software Heritage plays in preserving source code for the arts and creative coding.

RQ2: What is the geographic distribution of contributors to art-related code?

Open-source software is widely accessible, and anyone in the world can access libraries to practice generative art and creative coding. With this research question, we investigate to what extent these practices are indeed found all around the world.

RQ3: Which characteristics stand out in artworks made with p5.js?

Open-source libraries for generative art and creative coding are versatile and can support a wide variety of artistic expressions. For example, it is possible to generate static, abstract graphical compositions, as well as to control interactive audio-visual installations. With RQ3, we aim to unveil these various practices among the users of the most popular generative art library, p5.js.

RQ4: For what purpose do the top 42 contributors in our dataset develop or use art-related public code?

There are many reasons why people contribute source code with creative coding libraries. Understanding these different purposes provides qualitative insights about the motivations of users and can inform further development of these practices. This research question focuses on the 42 most prolific accounts in our dataset and the analysis of their main purpose.

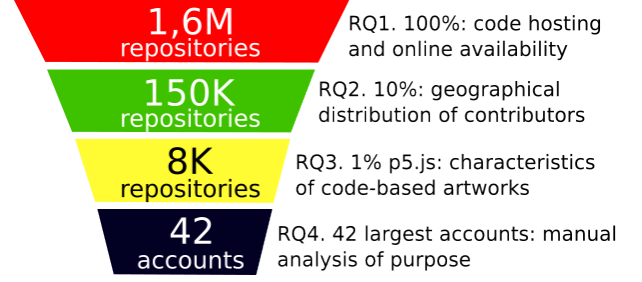


Fig. 3: Four research questions exploring the dataset at different scales

B. Methodology for RQ1

For this question, we collect two pieces of metadata for the complete set of 1,613,571 repositories. First, we determine on which platform each source code repository is hosted. We extract this information from the URL of the source code repository, e.g., `gitlab.com` or `git.sr.ht`. Second, we determine if the repository is still available online. Since we have collected our dataset from the Software Heritage code archive, it is likely that some repositories still present in the archive are no longer accessible online from their original hosting place, e.g., if the authors deleted it or if the hosting service shut down. The status of each URL is determined by sending HTTP requests (to the original hosting place) and interpreting the response codes. Status codes below 400 are considered online, whereas 4xx or 5xx indicate that the resource is no longer available. The first metric is an indicator of where code is distributed from, while the second one reveals challenges related to the conservation of this artistic heritage.

C. Methodology for RQ2

For this question, we randomly sample 10% of the online GitHub repositories in our dataset. We focus on GitHub as it hosts the most repositories and its API allows us to obtain user location information in a consistent way. To capture as many different contributors as possible, we make sure that each repository in the sample comes from a different GitHub account. In addition, we check that the selected repository is still online.

Following this process, we build a sample of 154,832 GitHub repositories. Key statistics about this sample are given in Table II. We fetch the main contributor (i.e., the one who authored the most commits) for each repository, leading to 96,581 unique contributor profiles. Then, we collect the location declared on their GitHub profile. On GitHub, the location field is optional; in our sample, 22,349 contributors provide a non-empty location string. Since location values are free form with no validation, these location entries are not standardized, e.g., “Montréal, QC”, “Montreal, Canada”, and “MTL” are valid, different strings that designate the same location. People may describe their location in different ways, such as a neighborhood, city, country, or even something unrelated to a real place (e.g., “Tatooine”, “my basement”, or “the Internet”). This creates inconsistencies and possible errors for mapping the data.

TABLE II: Consolidating locations of main contributors

Original number of repositories in the dataset	154,832
Number of unique main contributors	96,581
Number of contributors that declare a location	22,349
Number of declared locations resolved	14,427

TABLE III: Curating a dataset of single-file p5.js sketchers

Original number of repositories in the dataset	7,985
Number of p5.js sketch files	19,797
Number of p5.js sketch files under study	17,875

To normalize location values and obtain actual geo-locations, we use the Geoapify Location Platform⁴. For each contributor with a non-empty location field, we invoke the Geoapify API to convert the text into a consistent location format. When the input can be successfully matched to a real-world location, the API returns a valid, standardized location with corresponding GPS coordinates. After this step, we obtain 14,427 valid locations out of the 154,832 entries (9%). When a location is resolved to a place other than a country, such as a city or a state, we invoke the Geoapify API again with the location’s GPS coordinates to determine the corresponding country. With the Bertin.js library⁵, we visualize the different locations on a world map using a polyhedral projection [49].

D. Methodology for RQ3

First, we select a random 1% sample of repositories that use p5.js and are hosted on GitHub. Then, we search the repositories for source code files that implement a piece of art or creative coding. We refer to these files as *sketch files*. In p5.js, a sketch file is either an embedded JavaScript script in an HTML file, or a standalone JavaScript file. We check whether any HTML file in the repository includes the p5.js special `setup()` function. In such cases, we consider the HTML file to be the sketch file. If no such HTML file is found, we look for folders containing an HTML file together with one or more JavaScript files. Then, we check the name of each JavaScript file to determine whether it corresponds to one of the numerous p5.js additional libraries. For this, we use a predefined list of known p5.js libraries per the online documentation.⁶ If a JavaScript file is not identified as a library file, we then check whether it contains a `setup()` function. If it does, we consider that JavaScript file to be the sketch file. In order to focus on sketches that actually perform some creative processes, we ignore the sketches that invoke fewer than three p5.js functions, resulting in 17,875 sketches for analysis. Key metrics regarding the sample size and the collected sketch files are presented in Table III.

Second, we define the characteristics of sketches that we wish to capture, in order to classify p5.js usage practices. For this, we are inspired by the framework for understanding generative art introduced by Dorin and colleagues [50]. The Material and Processes labels capture elements of

the sketch related to the entities and actions it manipulates in order to perform the artwork: `processed_audio`, `processed_image`, `processed_text` capture the fact that a sketch can process preexisting material to display it as part of the work or to extract data from it to include in the generative process; `synthesized_audio`, `synthesized_image`, `synthesized_text` are values related to the fact that a sketch can use code to generate and evolve totally new material; `randomness` captures the fact that the sketch relies on randomness to increase the level of autonomy given to the machine in the generative process. The `Interaction` label can be `interactive` or `non-interactive` and captures whether the sketch has code related to user interactions. The `Sensory Outcome` labels capture the intention of the code regarding the audience’s perception of the work. The outcome can be `visual` if the code displays images or videos, generated or processed from existing sources; `auditory` outcome is when the work streams sound, generated or replayed from previous sources; and the outcome is either `static` or `time_based`. The label “time-based” is borrowed from the art conservators’ terminology to refer to a work that unfolds over time [51].

Third, we use an LLM to automatically characterize our 17,875 sketches, according to the classification of Table IV. In order to design a prompt for this labeling task, we first manually label 74 sketches. Two authors label them, compare their labels, and ask a third author to adjust in case of disagreement. Then, we design a first prompt and query the LLM to label the same 74 sketches. The first version of the prompt includes simple descriptions for each label created by the authors. It contains a one-sentence description for each characteristic, Material and Processes, Interaction and Sensory Outcome, as well as a fixed JSON schema for the expected output format for further processing.⁷ We validate this refined prompt with another sample of 50 manually labeled sketches. This last round led to minor adjustments of the prompt⁸. The prompt is organized as follows. First, it defines the expected output format as a JSON object with three fields: Material and Processes, Interaction, and Sensory Outcome. Second, it specifies the allowed labels for each characteristic and explains how each label should be assigned, including definitions and examples. Finally, it includes decision rules for ambiguous cases, such as the detection of `randomness` and the classification of `time_based` behavior.

To analyze the complete set of 17,875 sketches, we pass the prompt and the source code of a p5.js sketch to the model. The model output is a list of labels for each characteristic of the sketch. In case of failure or ill-formed output, we run the same prompt and file again one more time. We use the Qwen3-

⁷The initial version of the prompt is available as part of our reproduction package: https://github.com/sparkrew/art-in-humanity-code/blob/main/rq3/scripts/initial_prompts.py. An iterative process of prompting the model to generate labels, analyzing the true and predicted label distribution, as well as of comparing the labels inferred by the model with their respective p5.js source code reveal the need to strengthen the descriptions of each characteristic in order to meet expectations. Therefore, we substitute the initial descriptions with much stricter and comprehensive definitions, using concrete p5.js function examples when applicable, explicit decision rules, and edge-case checks in order to create the final version of the prompt.

⁸The final version of the prompt is available here: <https://github.com/sparkrew/art-in-humanity-code/blob/main/rq3/scripts/prompt.py>

⁴<https://www.geoapify.com>

⁵<https://github.com/riatelab/bertin>

⁶<https://p5js.org/libraries/>

TABLE IV: Three key characteristics and possible values for each characteristic to classify `p5.js` sketches

Characteristics	Values
Material and Processes	processed_audio; processed_image; processed_text; synthesized_audio; synthesized_image; synthesized_text; randomness
Interaction	interactive or non-interactive
Sensory Outcome	visual; auditory; static or time_based

Coder-30B-A3B-Instruct-FP8 [52] model for inference. The model was deployed on an NVIDIA H100 GPU with 80 GiB of VRAM, and 8 GiB system RAM, using the Rorqual compute cluster, operated by Calcul Québec,⁹ a regional partner of the Digital Research Alliance of Canada.¹⁰

E. Methodology for RQ4

For this question, we manually analyze the 42 GitHub accounts with the largest number of repositories in our dataset. We focus on these accounts, as their large number of repositories indicates a serious, sustained interest in code for art.

For each account, we collect a set of descriptors, such as the number of repositories, the type of account (individual or organization), the main programming language(s), and the activity status of the account. We classify an account as “active” if it shows activity on GitHub within the last two years, and “inactive” otherwise.

We also manually determine the purpose of each repository. Two authors independently conducted inductive coding of the repositories associated with these 42 GitHub accounts. Rather than applying predefined categories, they developed codes that emerge directly from the data (repository names, descriptions, code comments, README files, and GitHub profile information). This coding process aims to capture the primary purpose of the repositories and to characterize the level of activity of these accounts. After the initial coding phase, the two researchers compared their findings. In cases of discrepancy, they consulted a third author. This process resulted in three main purpose categories: *art*, *art education*, and *computer science education*. Together, these various metrics and assessments provide original, contextual insights about the various purposes for using and sharing code for art.

V. RESULTS

RQ1. Where is art-related code hosted and how much of it disappeared over time?

Figure 4 presents the distribution of public repositories for generative art, across different code hosting platforms. Given the large discrepancies among the number of repositories, we use a $\log_{10}$ scale on the Y axis. In the figure, we also distinguish between repositories that are still available online and those that are no longer available from their original hosting place, but remain accessible through the Software Heritage

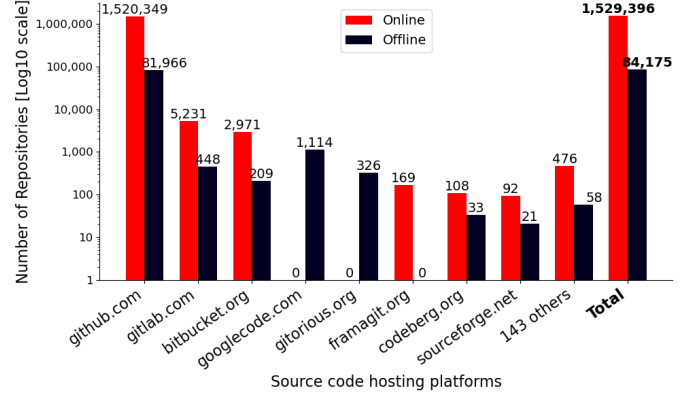


Fig. 4: Distribution of code repositories on different hosting platforms. For each hosting platform, we represent the number of repositories that are accessible as of January 06, 2026 (online), as well as the number of repositories that are not accessible anymore (offline). The 143 platforms that host less than 100 repositories are merged together in the “others” category.

archive. The last pair of bars, on the right of the figure, presents the total numbers of online and offline repositories.

GitHub is by far the most popular hosting platform, with 99.3% of all repositories: 1.5 million online repositories and 81k offline ones. GitLab and Bitbucket follow distantly, with 5,231 and 2,971 online, and 448 and 209 archived repositories, respectively.

Two hosting platforms, `googlecode.com` and `gitorious.org`, only have offline repositories. Gitorious was discontinued in 2015 when it was acquired by GitLab, which offered an opportunity to migrate the code repositories. We found 45 out of 326 URLs that matched organization and repository name in GitLab, suggesting that they were migrated from Gitorious to GitLab(.com). For example, Rhea Myers, a pioneer generative and NFT artist, migrated her project [like_that](#), to GitLab. Yet, the repository has 0 commits on GitLab, evidence that it was migrated from Gitorious for preservation purposes. Google Code was discontinued in 2015. Out of the 1,114 repositories on Google Code, almost all matched the Processing signal, whereas only two repositories included SuperCollider and one included vvvv. Thanks to our mining of Software Heritage, we could still access all of these art-related code repositories, which are now more than 10 years old.

Codeberg is an emerging collaborative software development environment created in 2018. It has grown to host nearly 485,000 repositories as of April 2026, of which 141 are identified in our dataset as generative art projects. An example is [pfzzz/MicroDexed](#), a synthesizer firmware project that combines code and sound design. This repository created in 2018 was included in our dataset because it contains Processing files. Codeberg is often mentioned in forums as an alternative to the monoculture of GitHub. The emergence of 141 art repositories in under a decade in Codeberg illustrates this vitality of open-source creative coding beyond GitHub.

Beyond these major platforms, the remaining repositories are scattered across 143 unique hosts, with 476 online and 58 offline repositories. Out of these 143 hosts, 29 have only of-

⁹<https://www.calculquebec.ca/>

¹⁰<https://www.alliancecan.ca/>

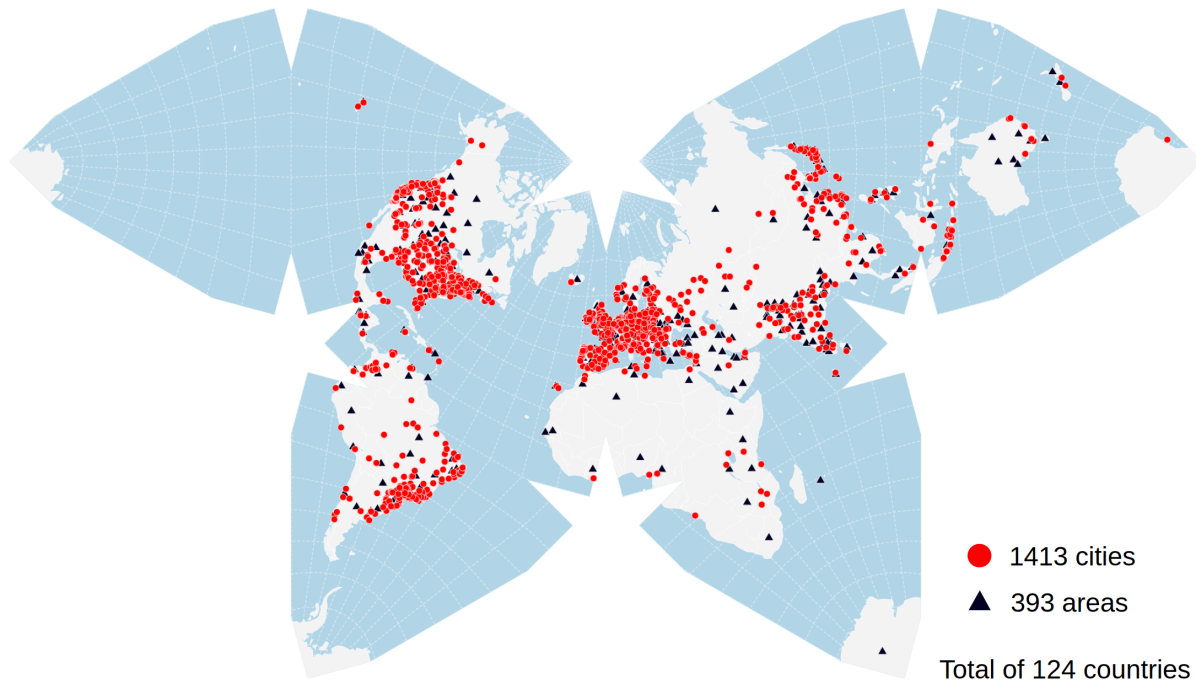


Fig. 5: Map of the 1,806 geographical locations gathered from the 14,427 contributors to generative art and creative coding, who have a valid location as part of their GitHub profile. We consider valid names of cities, or names of areas, such as a county, state, or country. The 1,806 locations are in 124 different countries across all continents.

fine repositories. These hosts are run by public organizations, schools, makers associations or a few individuals. Most of these hosts run a local GitLab, e.g., Stanford’s GitLab instance at code.stanford.edu, which hosts a number of repositories that use `Processing` for designing and documenting art installations. We also found niche self-hosted forges such as the [gitea](#) forge instance of the French robotics association [Poivron robotique](#), or a [forgejo](#) instance for the code of [Simon Molzy](#). Meanwhile, the [Rek](#) and [Devine](#) artists duo hosts their code on [sourcehut](#).

Across all platforms, 84,175 repositories, roughly 5.5% of the total, are no longer accessible online. This highlights the importance of code preservation (not only for software-based art) and validates the choice of mining Software Heritage, rather than a traditional online development platform like GitHub, for the goals of this study.

Answer to RQ1: While a vast majority of art-related public repositories are hosted on GitHub, public organizations, makers, and artists also use a variety of alternative hosts, from GitLab to SourceHut. Software Heritage has archived more than 80K art-related repositories that disappeared after deletion or the interruption of hosting services.

RQ2. What is the geographic distribution of contributors to art-related code?

In this RQ, we focus on 14,427 GitHub users with a geolocatable location in their profile, which we have extracted from a sample of 154,832 repositories. [Figure 5](#) shows the

1,806 unique locations that we found in these user profiles. In the map, we distinguish between different geographic granularities that contributors use when documenting their location: either a city, or a wider area such as a region or a country. We keep this distinction as we can precisely locate a city on a world map (●), whereas an area’s icon is positioned using its centroid (▲). The map clearly illustrates a key result from our study: open-source contributors who use source code as a medium for art and creative practices are present across the entire world. We found contributors across all continents, for a total of 124 distinct countries present in a 10% sample of our global dataset.

[Figure 6](#) shows the countries for which we counted more than 200 contributors. Each bar represents the number of contributors located in a specific country, either because they declare that they are in that country or in a city or region within it. First, we observe that these contributors are present all over the world, from 934 contributors in Brazil to the 586 contributors in Japan. This confirms what we previously observed on the map: there are a significant number of contributors to generative art and creative coding outside of North America and Europe. In particular, India, Brazil, and Japan rank second, third, and sixth, respectively, with 1,055, 934, and 586 contributing. The United States ranks first, with 4,303 contributors located all across the country, from 632 in New York City and 322 in San Francisco, to 8 in Honolulu, Hawaii, and 2 in Anchorage, Alaska. In order to gain deeper insights about these contributors and their diversity, we have manually inspected two groups of contributors: the Montreal community, and the contributors in underrepresented regions.

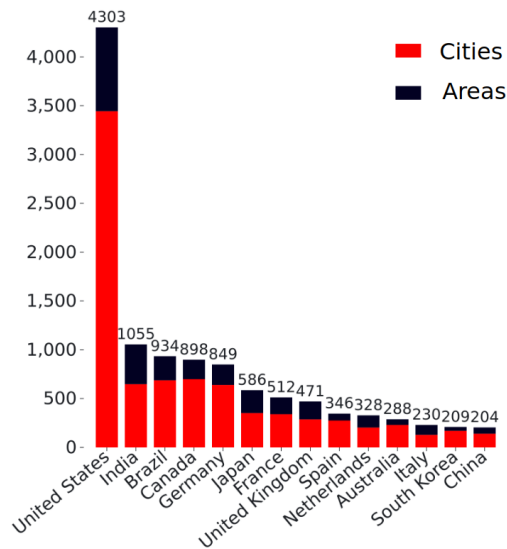


Fig. 6: Countries with more than 200 georeferenced contributors to art-related public source code repositories

a) *The Montreal Community*: Canada has 898 contributors, despite its relatively modest population. Six co-authors of this paper have settled in Montreal in recent years and notice that 116 contributors in our dataset also position themselves in this city. Manual inspection of these 116 contributor profiles reveals that these contributors all work in various sectors of Montreal’s vibrant creative industry : visual artists, light designers, sound artists, game developers, front-end developers, graphic designers, data visualization engineers, as well as students and professors in computation arts, new media arts, AI and robotics. We found a remarkable number of active profiles contributing public source code specifically for the arts. Some of them contribute within major art and creative technology organizations, such as [Stefan Schulz](#) working at Rafael Lozano-Hemmer’s Antimodular studio, [Jean-Michaël Celerier](#) supporting artists at la Société des Arts Technologiques, [Nathanaël Lécaudé](#) building open-source solutions for audiovisual installation at Moment Factory, [Nicolas Bouillot](#) building creative technology at the cooperative Lab148. Some others develop an independent practice, such as [Claire Avery](#), [Thomas Fredericks](#), [David Holcer](#), [Aaron Krajewski](#), [Guillaume Pelletier-Auger](#), [Marion Schneider](#).

b) *The Underrepresented Regions*: We identify the Caribbean, Africa, the Middle East, Central Asia, the Indian Ocean, and Iceland as underrepresented regions. We confirm 334 contributors distributed across 20 African countries, 11 countries in the Middle East, 7 in Central Asia, 5 in the Caribbean, and 3 in the Indian Ocean, as well as Iceland.

Through our manual analysis of these contributor profiles, we encountered several active generative artists in diverse parts of the world. In Central Asia, [Enkh-Amar Gantulga](#) is an active new media artist based in Ulaanbaatar, Mongolia. His practice spans visual arts, immersive installations, and live coding. His GitHub profile is balanced between forks of creative coding projects for live coding or glitch art, and projects that he maintains, mostly for live performances. He

works mostly in C++ and openFrameworks. In the Middle East, [Alekssei Vlasov](#) is based in Dubai, UAE. He develops and maintains open-source projects that interact and integrate with the commercial software Disguise, a leading technology for large-scale projection mappings. In the Indian Ocean, [Vihan](#) is a new media artist based in Sri Lanka who maintains an open-source organization consisting of creative developers, artists, and researchers. On GitHub, he contributes to creative coding repositories in p5.js and generative art projects in Python. In Iceland, all 10 contributors in our dataset are active in sound art or research related to sound. For example, [Jack Armitage](#) is an electronic music composer working in C++ and SuperCollider, as well as in Haskell and Tidal Cycles for live coding. His repositories are reusable libraries for code synthesis in different contexts, e.g., embedded on a Bela board, as well as actual pieces of electronic music, such as this repository for live coding [Charli XCX’s Vroom Vroom](#).

We also found a number of profiles related to graphic design, front-end development, interaction and Arduino development with Processing, as well as creative coding for programming education. Most of the contributors in the Caribbean and Africa are engaged in these activities. For example, [Sarah Marion](#) from Nairobi, Kenya, uses the p5.js library to design web interfaces. In Dar es Salaam, Tanzania, the high school STEM [Loyola](#) teaches programming through p5.js startup challenges. [Abel Wondafrash](#) from Ethiopia is one of the many African engineers who leverage Processing to build interactive and embedded systems with Arduino boards.

c) *The Case of Nonexistent Locations*: Answering this research question, we have discussed the 14,427 valid locations, which we gathered out of the 22,349 profiles that indicate a location. Looking into the variety of nonexistent locations used by creative coders could also reveal novel insights about these practitioners. For example, the audiovisual artist [Devine Lu Linvega](#) indicates they are located in "Neptune 7757", while they develop open-source technology for the arts through a nomadic life aboard a sailing ship. Other artists emphasize their inclusive and open practices through locations such as "earth" for [Serene](#) who “types code and concertos”, or "W/O/R/L/D/W/I/D/E" for [Lee T](#), a generative artist and maker.

Answer to RQ2: Generative art and creative coding rare present all around the world. We found contributors in 124 different countries, including large, active communities in Japan, Brazil, India, Europe and North America. We also confirm the presence of creative coders in Africa, Central Asia or the Indian Ocean. This is evidence of the large-scale adoption of code as a creative medium and an encouragement to develop these practices for sustained diversity in open-source software.

RQ3. Which characteristics stand out in artworks made with p5.js?

With this question, we dive into the diversity of creative practices among p5.js users. We use an LLM to analyze

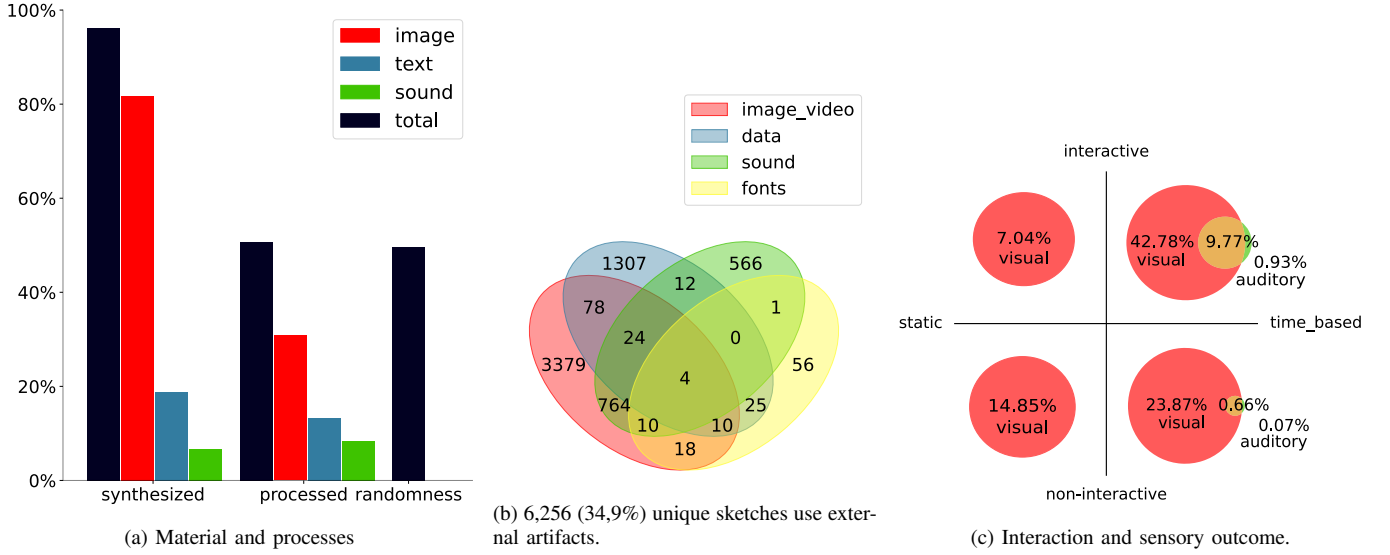


Fig. 7: Diversity of practices and external asset usage across a sample of 17,875 p5.js sketches. (a) and (b) show the distribution of practices based on the LLM output, with respect to their Material and processes (a) and the combination between the Sensory outcome and Interaction (b). In (c) we provide a Venn diagram with an overview of sketches that use at least one external asset, grouped by image/video, data, sound or font files, to feed the piece.

17,875 sketches according to the framework presented in Table IV. This framework characterizes each sketch along three dimensions: *Material and Processes*, *Interaction*, and *Sensory Outcomes*. For instance, the code in Figure 1 draws shapes using random values, without requiring user input, and is classified as follows `material_processes = [synthesized_image, randomness]; interaction = false; sensory_outcome = [visual, static]`.

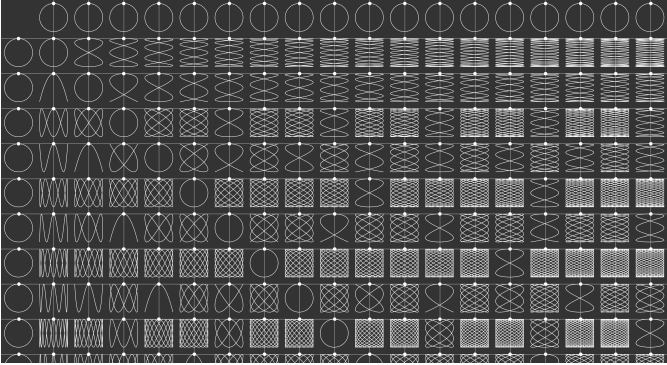


Fig. 8: Example of a static, synthesized image as generated by this sketch.

We first look into the `Material` and `Processes` used in p5.js sketches, as summarized in Figure 7a. The left part of the figure provides the distribution of sketches that synthesize elements. The total bar gives the share of sketches that have at least one synthesis action; the next three bars distinguish between sketches labeled with `synthesized_visual`, `synthesized_text`, or `synthesized_audio` according to different types of elements, noting that one sketch can be labeled with more than one. Almost all sketches (96.21%) include code that relates to synthesizing new material. The vast majority (81.70%) use code to generate visual elements.

This result is expected, given that p5.js initially aimed to support visual arts. From the earliest version of the library, there have been functions to facilitate generating and placing geometric shapes on the canvas, or handling colors.

We also observe a significant share of sketches that synthesize sound (6.61%). Sound synthesis either relies on a third-party library such as `Tone.js`, or on a dedicated p5.js extension for sound synthesis introduced in 2024. Functions for sound synthesis include the creation of oscillators whose frequency, waveform, and amplitude can be precisely defined, and which can be further refined using envelopes, filters, and effects. For example, `sine-rider` defines 6 oscillators with different frequencies and amplitudes. Text synthesis (18.81%) appears when the code prints text on the screen. As an example, some interactive artworks allow the audience to manipulate words that also contain links or typographic effects. It is important to note that the LLM labels a sketch that writes text for a button or a slider as `synthesized_text`.

Randomness is an important component in the generative art practices. As shown in the right part of Figure 7a, 49.63% of the sketches use randomness. This makes it possible to generate a different output in each execution and is a typical practice in generative art. While the concept of yielding a part of the artist's agency to the algorithm and the machine is at the core of generative art, this can take many forms in the code, such as generating a random number or using various sorts of noise functions. In order to embrace this diversity of implementations, we instruct the LLM to account for various expressions of randomness by providing it with a variety of examples that all convey the intention of randomness.

There are also a high number of sketches that do not explicitly use randomness. In these cases, variation still occurs and is triggered by other sources, such as time, user interaction or external data. Therefore, a sketch can still be generative even though its code does not include functions that

generate random numbers directly. In particular, the middle part of Figure 7a reveals that many sketches (50.72%) use external sources to feed the generative algorithm. Most of these sketches process images or video material as part of a work that transforms the artifact, or extracts data from it (size, density of certain pixel colors, etc.), or displays the image as part of a collage or a game.

To understand these sketches that process external artifacts, we analyze the type of artifacts they reuse in Figure 7b. Overall, 34.99% (6,256 sketches) of the dataset uses at least one external artifact. The most common type of artifact is image or video. In many cases, image artifacts are used as part of games, animations, or interactive scenes. For example, a *cycling game* loads image files for characters, obstacles, and backgrounds. Many sketches also process an external artifact to extract data that is used as part of the generative process. For example, this *sketch* processes audio to extract its amplitude and passes it on as the size of a generated ellipse. Data files, such as `.json` and `.csv`, also appear in many sketches. The sketches may extract data to control certain parts of the rendering, such as color, density, or duration; in other cases, data is used as textual source. We find only a few sketches that rely on external font files, which might indicate that the works that use text generally rely on the default font families provided by `p5.js`.

We note that the middle part of Figure 7a shows that 50.72% of sketches process image, text, or audio, while in Figure 7b we find that only 34.99% of sketches load a media artifact from the same folder as the sketch. This difference occurs when the sketch fetches the artifact to be processed dynamically from external sources. The artifact can be loaded through an HTTP request, as in this *sketch*. It can also be passed by a user, for example in a text box or through sensors, as in this *sketch* that fetches a video feed from a webcam.

We now look at the diversity of experiences for the audience, with labels regarding *Interaction* and *Sensory Outcome*, in Figure 7c. The figure shows four quadrants, comparing *interactive*, *non-interactive*, *static*, and *time_based* outcomes. In each quadrant, one Venn diagram represents the distribution of sketches that have *auditory*, *visual*, or both outcomes.

A vast majority of the sketches produce a *time_based* outcome, as observed in the two quadrants on the right of the figure. Among these, 60.52% are also *interactive*, and 39.45% are *non-interactive*. The sketches that produce *visual*, *interactive*, *time_based* outcomes correspond to animations on the web page, games, or interactive visual artworks. Meanwhile, the *visual*, *non-interactive*, and *time_based* sketches are autonomous animations, such as particle systems, simulations, autonomous drawing systems, and cellular automata. For example, this *sketch* creates three walkers that autonomously move at a constant speed, following a random path.

An *auditory* outcome is necessarily *time_based*, as sound has to unfold over time. Therefore, the quadrants on the right show the share of sketches that are *auditory* or *auditory and visual*. Manual inspection reveals that many of the sketches that produce an *auditory and visual*

outcome are web page animations or games, such as the cycling game mentioned previously. An example of non-interactive audiovisual can be found in this *sketch* where spirals and sound unfold over time, autonomously. Purely auditory sketches are few, with only 179 sketches in total. In this group, the interactivity allows the audience to control some parameters of synthesized sounds, such as the pitch, the volume or the shape of the oscillator.

We now look at the left quadrants of Figure 7c. They show that only 21.89% of the sketches produce a *static* outcome, including 14.85% that are *non-interactive* and 7.04% that are *interactive*. The former category implements an algorithm that draws an image on the screen, which does not move, as in Figure 8. Manual inspection of the latter category of sketches reveals that they implement interactive features such as a text box or mouse capture, but these interactions do not affect the generation of the visuals.

Answer to RQ3: `p5.js` is a versatile library used by artists and creative coders across a wide range of practices. A vast majority of the sketches produce a *visual* outcome, yet almost 10% combine *auditory* and *visual* outcomes. Most of the works are *time_based*. Manual inspection of sampled sketches reveals that they range from abstract artworks to web animations and games. These findings are useful for educators to embrace the variety of practices in their curricula, and for researchers to design techniques for fine-grained qualification of generative sketches.

RQ4. For what purpose do the top 42 contributors in our dataset develop or use art-related public code?

In order to address this question, we manually analyzed a subset of repositories from our dataset. We selected the 42 GitHub accounts associated with the largest number of repositories. The subset is therefore made up of 17,156 repositories. The account that has the largest number of art-related repositories belongs to the GitHub organization `League-Level0-Student`, with 2,348 repositories, followed closely by the individual account `junkiyooshi`, with 2,344 repositories. The 42nd account, owned by user `ruchikawhj`, has 152 repositories.

Our analysis reveals three main purposes that characterize how these repositories are created and used, in relation to generative art: *computer science (CS) education*, *art education*, and *art*. The *CS education* category includes accounts created to teach and learn about programming concepts through generative art and creative coding, and is mainly represented by educational organizations and educator accounts. The *art education* category includes accounts developed for teaching and learning contexts related to creative programming, visual design, and artistic experimentation, often by universities, art programs, or educators. The *art* category includes repositories created primarily with an artistic point of view, either by individual artists or by art-oriented organizations such as studios or collectives.

Figure 9 displays how the 42 accounts and their repositories 17,156 are distributed according to their primary purpose. In

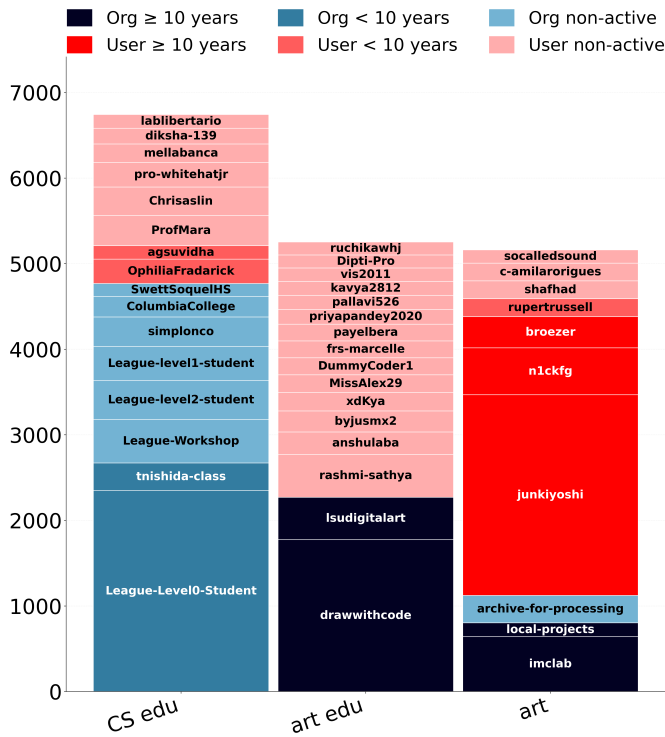


Fig. 9: Distribution of the 17,156 repositories associated with the top 42 contributors in our dataset, grouped by purpose (*art*, *art education*, and *computer science education*) and activity status (*active* vs. *non-active*). Each segment represents one user or organization, with height proportional to the number of repositories.

the figure, each segment represents one GitHub account, with height proportional to the number of repositories associated with that account. We use two colors and three shades within each color to distinguish between accounts belonging to individual users (orange) and organizations (blue), as well as between non-active accounts (lightest shade), accounts active for less than 10 years (medium shade), and accounts active for more than 10 years (darkest shade).

Activity status helps to distinguish between sustained and time-bounded forms of engagement with public code. The figure shows a relatively high number of non-active accounts, particularly in the two educational categories, where participation is often tied to a specific period of study, teaching, or course involvement. Although the *computer science education* category contains the largest number of repositories, many of its contributors were active between 2020 and 2022, a period that coincided with the COVID-19 pandemic and widespread stay-at-home restrictions, when teaching and learning moved to online platforms. This suggests that a substantial part of these repositories was created for specific educational contexts, such as courses, assignments, or teaching materials, rather than as part of a long-term artistic practice. By contrast, the *art* category shows a higher proportion of active contributors, indicating that engagement with public code for artistic purposes is more likely to persist over time. The *art education* category occupies an intermediate position, combining active artistic organizations and pedagogical initiatives with contributions from non-active users. At the same time, the fact that many

educational accounts are no longer active on GitHub does not necessarily mean that these practices disappeared; rather, it may indicate that they moved to other platforms or to less publicly visible forms of code sharing.

CS education: In this category, generative art technologies such as Processing or p5.js are mainly used as an engaging way to introduce programming concepts through visual and interactive output. In many of these cases, each student creates one repository per assignment or module within a GitHub organization, and these repositories accumulate over multiple courses and academic years. An example of an individual account in this category is *OphiliaFradarick*, a data analyst who worked as a coding instructor for *whitehatjr*, an online coding school for children, between 2020 and 2023. In our top 42, we also identify one account related to the same *whitehatjr* organization, *pro-whitehatjr*. The repositories associated with *OphiliaFradarick* account include forks of projects from *pro-whitehatjr* as well as student assignments, illustrating how code in this category forms part of structured teaching activities. Another example of an organizational account is *ColumbiaCollege*, which corresponds to Columbia College, a public community college in Sonoma, California, USA. The repositories in this account mainly use Processing and store student assignments and midterm projects from 2019. This account is associated with 240 repositories in our dataset. In general, individual accounts in this category are often associated with students and freelance educators, many of whom were active between 2020 and 2022. By contrast, organizational accounts are more likely to remain active over time because they are sustained by recurring students, multiple instructors, and the ongoing reuse of GitHub as part of course administration and submission workflows.

Art education: This category reflects the use of creative coding in teaching and learning environments for art and design students. It includes accounts associated with courses, workshops, studio-based learning, and institutional programs, such as university initiatives, in which students experiment with code while developing both technical and creative skills. Compared with *computer science education*, this category places greater emphasis on the exploration of visual forms, interaction, abstract ideas, and digital creativity, and less emphasis on programming concepts and algorithms. An example of a GitHub organization in this category is *lsudigitalart*. The documentation of the repositories in this account reveals that it relates to an online course that teaches students contemporary web design and development workflows, with particular emphasis on interaction and visualization. It stores student homework and project submissions, which helps explain its large number of repositories. The account remains active and has been active since 2016, suggesting a pedagogical use of GitHub in this context. In contrast, individual accounts in this category are often associated with students. For example, *rashmi-sathya* represents a non-active student account in our top list. This account contains one repository per project, many of which use the p5.js library to create interactive pieces such as games. For instance, one repository is described as a *Bouncy Ball* created using the *Matter.js* physics engine.

Such examples illustrate how repositories in this category often function as records of exploratory and project-based learning in creative coding.

Art: By opposition to the GitHub accounts categorized as *computer science education* or *art education*, the accounts whose primary purpose is *art* contain large numbers of repositories for a wider variety of reasons. These repositories often correspond to individual artworks, installation projects, technical experiments, reusable tools, or forks of open-source technologies relevant to artistic production. This diversity reflects the broad range of ways in which code is used as a creative medium in artistic practice. The majority of repositories in the *art* category represent artworks, prototypes, or experiments conceived by artists or art-oriented organizations rather than structured educational tasks. Individual artists and organizations often create one repository per artwork, installation, or technical exploration. In some cases, contributors maintain sustained creative routines, publishing new sketches, experiments, or artworks over several years. As a result, repositories in this category function as records of artistic production, experimentation, and technical evolution over time.

An example of a GitHub organization in this category is *imclab*, an art gallery based in New York City. Its repositories include both original projects and forks, suggesting a practice that combines artistic production with the adaptation and reuse of existing open-source technologies. The range of tools and languages present in this account indicates engagement with multiple technical ecosystems relevant to digital art production, including creative coding and multimedia programming environments. An example of an individual account is *n1ckfg*, whose profile combines artistic practice with technical research and development. This account includes personal projects, forks, and tool-oriented repositories (like AEToolbox UI panel) spanning multiple languages and domains, including animation, 3D drawing, sound-related work, and interfaces for artistic production.

To illustrate the diversity of contributors in our dataset, we examine the three largest accounts. These contributors are informative because they represent different regions, purposes, and technical ecosystems, while also showing a long-term engagement with public code related to generative art.

League-Level0-Student is the largest contributor, with 2,348 repositories. This account belongs to *The League of Amazing Programmers*, an educational organization based in California, USA, that teaches programming to students from 3rd to 12th grade. It is representative of the *computer science education* category, with repositories organized around classes, levels, modules, and assignments. In many cases, each repository corresponds to the work of one student or a group of students on a specific assignment. The account also includes templates, forks, and course materials that support teaching activities. For example, in Module 1 of Level 0, students learn introductory programming concepts such as conditionals (*if/else*), variable initialization, and *for* loops, while creating simple graphical outputs with *Processing*. This structure explains the large number of repositories: GitHub is used here primarily to organize, submit, and store student work

over time. The League is also associated with other accounts in our top 42 contributors, including *League-Workshop* ranked 6th, *League-Level2-Student* ranked 9th, and *League-Level1-Student* ranked 10th. Together, these accounts contribute 3,309 of the 6,742 repositories in the *computer science education* category, representing 55% of that subset.

junkiyoshi represents a different profile. This individual account belongs to NAKAUCHI Kiyoshi, a software developer based in Japan whose public GitHub activity reflects a daily engagement with generative art, particularly with *openFrameworks*. Since 2016, he has created one repository per day, pushing one artwork to that repository and sharing the results on social media platforms such as Instagram or X. In our dataset, this account is associated with 2,344 repositories; as of April 16, 2026, his GitHub profile reported approximately 3.2 thousand repositories. His artworks frequently explore movement, randomness, and geometric forms such as triangles, spheres, and particle-based compositions, often rendered in either vivid color palettes or black and white. *junkiyoshi* provides a clear example of long-term individual artistic engagement with code.

drawwithcode is an organization focused on the education of art and design. It is associated with the elective course *Creative Coding* at Politecnico di Milano, Italy, where code is taught as a generative tool for design students rather than a technical skill. Its repositories are primarily based on JavaScript and *p5.js*, and are linked to student assignments and projects developed within the course. The large number of repositories can be explained by the course structure: initially, each student team’s project was stored in a separate repository for each course edition, while more recently GitHub Classroom has been used to automatically create and manage repositories for student submissions. The account is also connected to *DensityDesign*, a research lab in the Design Department focused on the visual representation of complex social, organizational, and urban phenomena.

Answer to RQ4: The manual analysis of the top 42 contributors and their 17,156 source code repositories reveals that public code in generative art serves for three main purposes: computer science education, where code for the art is used as creative way of learning; art education, where teaching focuses on code as an art and design medium; and art production, where code is produced, reused and remixed to develop novel art installations.

VI. DISCUSSION

In this section, we discuss the implications of our study for the field of software research, and the threats to validity.

A. Implication for Software Engineering Research

a) *Art is a Software Application Domain:* When using code as a creative medium, artists and creative coders have access to the full spectrum of software technology. Consequently, new media installations, interactive artworks, and

blockchain-based generative artworks are full-fledged software systems. While they share some challenges with any other software system, they also have their own, which can be further investigated by the software research community. For example, *Processing* and *p5.js* have grown from libraries into full ecosystems of third-party libraries that address specific concerns in generative art, such as handling sound or gameplay. These specialized libraries support the many practices we have observed in RQ3. Yet, the software supply chain of generative art is vast and much understudied.

In addition to third-party libraries, we have observed that generative artworks depend on many other types of artifacts, such as audiovisual assets, specific protocols to control multiple devices, or third-party APIs to fetch data that feeds the artwork. These various types of dependencies contribute to the fragility of these artworks. The software engineering research community can contribute dedicated techniques to precisely document these dependencies in order to support artists and cultural workers in the preservation of these works.

We have seen that the Software Heritage archive plays a key role in the preservation of art-related source code. The restoration of early code-based artworks will also require specific migration techniques. For example, Adobe Flash and Java applets were popular for *net.art* pieces in the early 2000s, and traces of source code can be found in archives. However, it is hard to perform these pieces on modern computers or even portable devices. The software engineering research community can contribute code transformation and migration techniques that aim at restoring or facilitating access to software-based artworks.

b) Software Literacy with Creative Coding: When answering our four research questions, we have manually inspected hundreds of code repositories from our dataset. This inspection reveals that a significant share of the repositories are authored by students. These student repositories include assignments for various courses related to art and creative coding. In particular, we note the significant influence of Daniel Shiffman’s *coding train* tutorials and learning material that help students learn about the basic concepts of programming, all the way to advanced physics simulations and interactive technology for immersive artworks.

These repositories are evidence of the role that creative coding plays to introduce software concepts to diverse audiences. Our work can inspire software educators to explore art as an entry point to software development, as well as a way to demystify software technology and increase software literacy in diverse communities.

B. Threats to Validity

Here we discuss the threats to the validity of our study and how we have mitigated them.

The first threat relates to our selection process, which reveals a worldwide phenomenon, yet through a limited lens. Our set of 1,613,571 repositories might well just be a drop in the ocean of humanity’s code for the arts. We have selected code repositories based on the presence of certain file-level signals. Consequently, we omit many pieces that do not meet

our selection criteria, including: art-related public repositories that call generative art libraries in the file contents or those that do not use generative art libraries at all, such as the *CryptoPunks* [53]¹¹; art projects that use other libraries like *Blender* or a game engine, such as *Unity*; and all the historical code-based artworks for which source code is not available online, such as pieces from the demoscene or the code that Vera Molnar wrote by hand in her diaries [54]. Also, we mine from *Software Heritage*, who currently does not archive public code that is published on art-specific platforms such as *OpenProcessing*¹² or the code of NFTs that is available on *IPFS* [55]. Last but not least, we can only analyze code repositories that are shared publicly. We do not have access to all the code that is developed within art and design studios, as well as in creative companies that do not share their code. Despite these limitations, our selection criteria, which we motivate in section III, allow us to collect a large, diverse, and historically relevant set of code repositories that have an artistic intention as demonstrated throughout this work.

The second threat relates to the correctness of the code we use to analyze these source code repositories. We identify two threats here: one related to the quality of the code we write; the other related to the quality of the data and third-party tools we use. This latter point is particularly relevant to RQ2, where we analyze locations derived from arbitrary strings entered by users and rely on a geocoding service provided by *Geoapify* to resolve them. We mitigate these threats through several rounds of manual testing and analysis of data samples to clean the data and ensure the quality of the results. This results in a conservative lower bound on the number of locations we analyze and calls for finer-grained geospatial analysis in the future.

VII. CONCLUSION

In this paper, we have explored the public source code that thousands of artists, creative coders, students, and educators have contributed across the world. The study provides the following insights: while most of the code is shared on *GitHub*, various maker and artist communities share on alternative code hosting platforms; active artists and communities are present all around the world; practitioners use *p5.js* to develop a wide variety of practices, from randomized abstract images to interactive auditory and visual artworks or games; art-related code serves various purposes that span programming and art education to professional art and design practices.

Generative art might be as ancient as art itself [56], and as soon as artists have had access to computers, they have explored code as a medium to build generative systems. This first large-scale empirical study of public code repositories related to art and creative coding provides novel insights for artists and software developers who develop new libraries and tools for digital art; for software and art educators who aim at introducing programming concepts through creative technology; for software and art professionals who care about the preservation of our digital cultural heritage. Our results open

¹¹<https://github.com/larvalabs/cryptopunks>

¹²<https://openprocessing.org/>

several threads of future work. First, we wish to investigate to what extent these projects still execute and what the strategies are to restore the broken projects. Second, we aim to model the various types of software, data, and network dependencies that underlie these artworks, in order to support conservation and restoration practices for generative art.

DATA AVAILABILITY

The complete set of art-related code repositories is available on [Zenodo](#), and all the scripts and intermediate datasets to answer the research questions are available as part of our reproduction package on GitHub: [art-in-humanitys-code](#), archived on Software Heritage with SWHID [swh:1:rev:1b4edb9beee3a15687273e70a5500f35a135e8a](#).

ACKNOWLEDGEMENTS

We thank Geoapify for granting free access to their infrastructure for research purposes. We thank [anacat from Braga, Portugal](#) and the thousands of generative artists who have inspired this work. This research was enabled in part by support provided by Calcul Québec ([calculquebec.ca/](#)) and the Digital Research Alliance of Canada ([alliancecan.ca](#)). This research was undertaken thanks to funding from the Canada Research Chairs Program, FRQ, NSERC and from SSHRC. This work was made possible by Software Heritage, the universal source code archive: <https://www.softwareheritage.org>.

REFERENCES

- [1] C. Paul, *Digital art*. Thames & Hudson, 2023.
- [2] M. Cotton, T. Rivers Ryan, and M. Rosen, *Radical Software: Women, Art & Computing, 1960-1991*. Verlag der Buchhandlung Walther und Franz König, 2024.
- [3] V. Molnar, "Toward Aesthetic Guidelines for Paintings with the Aid of a Computer," *Leonardo*, vol. 8, pp. 185–189, 1975.
- [4] Z. Šýkora and J. Blažek, "Computer-aided multi-element geometrical abstract paintings," *Leonardo*, vol. 3, no. 4, pp. 409–413, 1970.
- [5] S. Cornock and E. Edmonds, "The creative process where the artist is amplified or superseded by the computer," *Leonardo*, vol. 6, no. 1, pp. 11–16, 1973.
- [6] J. McCormack, A. Dorin *et al.*, "Art, emergence and the computational sublime," in *Proceedings of Second Iteration: A Conference on Generative Systems in the Electronic Arts*. Melbourne: CEMA, 2001, pp. 67–81.
- [7] M. Puckette *et al.*, "Pure Data: Another Integrated Computer Music Environment," *Proceedings of the second intercollege computer music concerts*, pp. 37–41, 1996.
- [8] J. McCartney, "Rethinking the Computer Music Language: SuperCollider," *Computer Music Journal*, vol. 26, no. 4, pp. 61–68, 2002.
- [9] J. Maeda, *Design by Numbers*. MIT press, 2001.
- [10] C. Reas and B. Fry, *Processing: a Programming Handbook for Visual Designers and Artists*. MIT Press, 2007, vol. 6812.
- [11] J. Noble, *Programming Interactivity: A Designer's Guide to Processing, Arduino, and Openframeworks*. O'Reilly Media, Inc., 2009.
- [12] L. McCarthy, C. Reas, and B. Fry, *Getting Started with p5.js: Making Interactive Graphics in JavaScript and Processing*. Maker Media, Inc., 2015.
- [13] M. A. Boden and E. A. Edmonds, "What is Generative Art?" *Digital Creativity*, vol. 20, no. 1-2, pp. 21–46, 2009.
- [14] K. Pepler and Y. Kafai, "Creative Coding: Programming for Personal Expression," in *Proceedings of the International Conference on Computer Supported Collaborative Learning (CSCL)*, 2009.
- [15] R. Di Cosmo and S. Zacchiroli, "Software heritage: Why and how to preserve software source code," in *Proceedings of the International Conference on Digital Preservation (iPRES)*, 2017.
- [16] A. Islam and A. Hindle, "TriGraph: A Probabilistic Subgraph-Based Model for Visual Code Completion in Pure Data," in *Proceedings of the International Conference on Mining Software Repositories (MSR)*, 2025, pp. 737–749.
- [17] E. M. Fredericks, A. C. Diller, and J. M. Moore, "Generative art via grammatical evolution," in *Proceedings of the International Workshop on Genetic Improvement (GI)*, 2023, pp. 1–8.
- [18] T. Angert, M. Suzara, J. Han, C. Pondoc, and H. Subramonyam, "Spellburst: A node-based interface for exploratory creative coding with natural language prompts," in *Proceedings of the Symposium on User Interface Software and Technology (UIST)*, 2023, pp. 1–22.
- [19] L. L. McCarthy, T. Hughes, and G. Levin, "Open Source Software Toolkits for the Arts (OSSTA): a Convening," <https://github.com/CreativeInquiry/OSSTA-Report>, 2021, accessed: 2026-06-25.
- [20] A. Trifonova, L. Jaccheri, and K. Bergaust, "Software engineering issues in interactive installation art," *International Journal of Arts and Technology*, vol. 1, no. 1, pp. 43–65, 2008.
- [21] Z.-W. Wu, H. Qu, and K. Zhang, "A survey of recent practice of artificial life in visual art," *Artificial Life*, vol. 30, no. 1, pp. 106–135, 2024.
- [22] M. Verano Merino and J. P. Sáenz, "The art of creating code-based artworks," in *Proceedings of the Extended Abstracts of the Conference on Human Factors in Computing Systems (CHI)*, 2023, pp. 1–7.
- [23] B. Subbaraman, S. Shim, and N. Peek, "Forking a sketch: How the Openprocessing Community uses remixing to Collect, Annotate, Tune, and Extend Creative Code," in *Proceedings of the Designing Interactive Systems Conference (DIS)*, 2023, pp. 326–342.
- [24] B. Baudry, E. N. Gustafsson, R. Kaufman, and M. Euler, "Myriad people open source software for new media arts," in *Proceedings of International Conference on Mining Software Repositories (MSR)*, 2025.
- [25] A. Islam, K. Eng, and A. Hindle, "Opening the Valve on Pure-Data: Usage Patterns and Programming Practices of a Data-Flow Based Visual Programming Language," in *Proceedings of the International Conference on Mining Software Repositories (MSR)*, 2024, pp. 492–497.
- [26] A. Bogdan, M. V. Merino, and I. Malavolta, "The Ecosystem of Open-Source Music Production Software - A Mining Study on the Development Practices of VST Plugins on GitHub," in *Proceedings of the International Conference on Mining Software Repositories (MSR)*, 2025, pp. 513–525.
- [27] W. Payne, M. Kaney, I. Rodrigues, and A. Hurst, "Exploring Technical and Creative Posts in Online Live Coding Communities: An Analysis of Tidal Club and in_thread," in *Companion Proceedings of the ACM International Conference on Supporting Group Work (GROUP)*, 2025, pp. 15–21.
- [28] R. Milewicz, G. Pinto, and P. Rodeghero, "Characterizing the roles of contributors in open-source scientific software projects," in *Proceedings of the International Conference on Mining Software Repositories (MSR)*, 2019, pp. 421–432.
- [29] B. Trinkenreich, M. A. Gerosa, and I. Steinmacher, "Unraveling the drivers of sense of belonging in software delivery teams: Insights from a large-scale survey," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2024, pp. 1–12.
- [30] S. M. Hyrynsalmi, S. Baltes, C. Brown, R. Prikladnicki, G. Rodriguez-Perez, A. Serebrenik, J. Simmonds, B. Trinkenreich, Y. Wang, and G. Liebel, "Making software development more diverse and inclusive: Key themes, challenges, and future directions," *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 5, pp. 1–23, 2025.
- [31] B. Vasilescu, D. Posnett, B. Ray, M. G. van den Brand, A. Serebrenik, P. Devanbu, and V. Filkov, "Gender and tenure diversity in github teams," in *Proceedings of the Conference on Human Factors in Computing Systems (CHI)*, 2015, pp. 3789–3798.
- [32] E. M. Fredericks, J. M. Moore, and A. C. Diller, "Generativegi: creating generative art with genetic improvement," *Automated Software Engineering*, vol. 31, no. 1, p. 23, 2024.
- [33] J. Li, J. Brandt, R. Mech, M. Agrawala, and J. Jacobs, "Supporting visual artists in programming through direct inspection and control of program execution," in *Proceedings of the Conference on Human Factors in Computing Systems (CHI)*, 2020, pp. 1–12.
- [34] R. Sacadura, L. Goncalo, T. Martins, and P. Machado, "Addressing the Combinatorial Explosion of Code-Based Visual Artifacts," *IEEE Computer*, vol. 59, no. 09, pp. 69–76, Sep. 2026.
- [35] J. Li, S. Hashim, and J. Jacobs, "What we can learn from visual artists about software development," in *Proceedings of the Conference on Human Factors in Computing Systems (CHI)*, 2021, pp. 1–14.
- [36] "OSACC 2023 Report," <https://opensourceart.cc/osacc-2023-report/>, 2023, accessed: 2026-06-25.
- [37] C. Coleman, "Cultivating the Code: Community, Care, and the Clinic for Open Source Arts," *IEEE Computer*, vol. 59, no. 09, pp. 27–36, Sep. 2026.

- [38] E. Sandberg, “Creative Coding on the web in p5.js: a Library where JavaScript Meets Processing,” Bachelor’s Thesis, Blekinge Institute of Technology, Sweden, 2019.
- [39] K. Woo and G. Falloon, “Problem solved, but how? an exploratory study into students’ problem solving processes in creative coding tasks,” *Thinking Skills and Creativity*, vol. 46, p. 101193, 2022.
- [40] I. Greenberg, D. Kumar, and D. Xu, “Creative coding and visual portfolios for cs1,” in *Proceedings of the Technical Symposium on Computer Science Education*, 2012, pp. 247–252.
- [41] G. Levin and T. Brain, *Code as Creative Medium: a Handbook for Computational Art and Design*. MIT Press, 2021.
- [42] J. Sykes, M. Grierson, R. Fiebrink *et al.*, “Seeing programming seeing: Exploring the pedagogical values of functional errors in creative coding,” in *Proceedings of the Conference on Computation, Communication, Aesthetics & X*, 2023.
- [43] A. M. McNutt, S. Cohen, and R. Chugh, “Slowness, Politics, and Joy: Values That Guide Technology Choices in Creative Coding Classrooms,” in *Proceedings of the Conference on Human Factors in Computing Systems (CHI)*, 2025, pp. 1–16.
- [44] A. D. Knochel and R. M. Patton, “If art education then critical digital making: Computational thinking and creative code,” *Studies in Art Education*, vol. 57, no. 1, pp. 21–38, 2015.
- [45] T. Dufva, “Art education in the post-digital era-experiential construction of knowledge through creative coding,” Ph.D. dissertation, Aalto University, 2018.
- [46] A. Pietri, D. Spinellis, and S. Zacchiroli, “The software heritage graph dataset: Large-scale analysis of public software development history,” in *Proceedings of the International Conference on Mining Software Repositories (MSR)*, 2020, pp. 1–5.
- [47] P. Boldi, A. Pietri, S. Vigna, and S. Zacchiroli, “Ultra-Large-Scale Repository Analysis via Graph Compression,” in *Proceedings of the International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2020, pp. 184–194.
- [48] T. Fontana, S. Vigna, and S. Zacchiroli, “WebGraph: The Next Generation (Is in Rust),” in *Proceedings of the Web Conference (WWW)*, 2024, pp. 686–689.
- [49] U. Brehm and E. Schulte, “Polyhedral maps,” in *Handbook of discrete and computational geometry*. Chapman and Hall/CRC, 2017, pp. 533–548.
- [50] A. Dorin, J. McCabe, J. McCormack, G. Monro, and M. Whitelaw, “A Framework for Understanding Generative Art,” *Digital Creativity*, vol. 23, no. 3-4, pp. 239–259, 2012.
- [51] C. Frohnert and H. Roemich, “Time-based media art conservation education program at nyu: Concept and perspectives,” *The Electronic Media Review*, vol. 5, 2017.
- [52] Q. Team, “Qwen3 technical report,” *arXiv*, 2025.
- [53] M. Hall and J. Watkinson, “CryptoPunks,” <https://larvalabs.com/cryptopunks>, 2017, accessed: 2026-06-25.
- [54] V. Molnar, *Journaux Intimes (1976 -2020)*. Oniris, 2023.
- [55] “fxhash — The Artists,” <https://www.rightclicksave.com/article/fxhash-the-artists>, accessed: 2026-05-21.
- [56] P. Bauman, “Generative Art Timeline,” <https://timeline.lerandom.art/>, 2026, accessed: 2026-06-25.