Distilling Software-Based Art from Source Code Repositories

Mauricio Verano Merino[1], Benoit Baudry[2]

[1]Vrije Universiteit Amsterdam, Amsterdam, The Netherlands
[2]Université de Montreal, Montreal, Canada

Abstract

Software-based art is highly vulnerable to digital obsolescence, and studying its evolution requires clean datasets. Mining open-source platforms (e.g., GitHub, OpenProcessing) for curating software-based creative code (e.g., p5.js sketches) is hindered because repositories are overwhelmingly dominated by educational boilerplate, homework, and tutorials. Traditional software engineering and MSR metrics (e.g., commit activity, contributor count) cannot be used for curation because they do not capture artistic intent. Therefore, we present a vision for a multimodal agentic pipeline that combines static code analysis, headless visual rendering, and an LLM-as-a-judge architecture to curate a gold-standard software-based artwork dataset. This framework unlocks systematic software evolution, dependency decay analysis, and maintenance research to preserve digital cultural heritage.

Keywords

static analysis, creative coding, software-based artworks, preservation of digital heritage

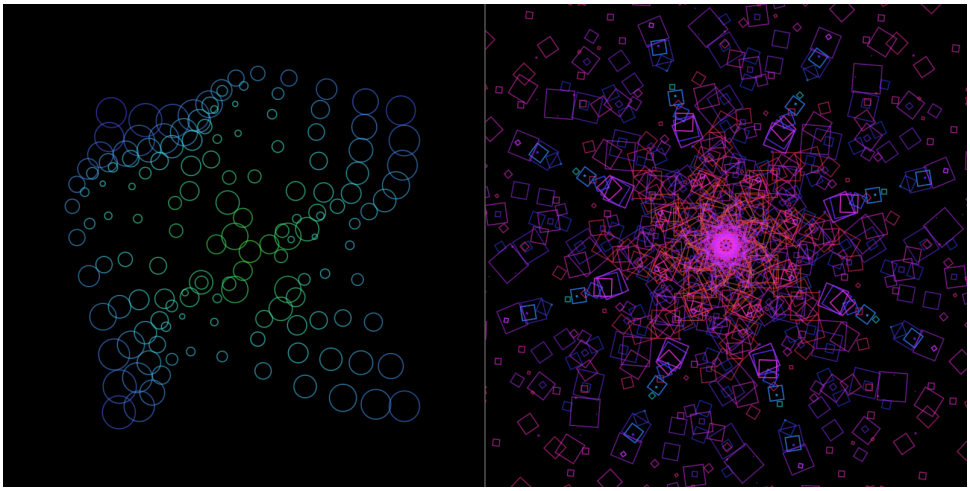


Figure 1: Structural Equivalence vs. Visual Divergence [1]. An example where similar algorithmic logic yield organic fluid fields (left) and complex geometric mandalas (right) based solely on coordinate mapping.

1. Introduction

Software is a vital component of modern digital culture [2], yet it remains extraordinarily fragile. While advances in artificial intelligence (AI) have accelerated digital creation, methods for preserving, maintaining, interpreting, and archiving software-based artwork remain severely underdeveloped. Art conservators, who rarely possess formal computer science backgrounds, face an overwhelming bottleneck: preserving and archiving these dynamic artifacts requires setting up a development environment, understanding and executing source code, and analyzing visual output at a scale that cannot be processed manually.

BENEVOL2026: Belgium-Netherlands Software Evolution Workshop 26–27 November, 2026, Madrid, Spain
m.verano.merino@vu.nl (M. Verano Merino); benoit.baudry@umontreal.ca (B. Baudry)
https://maveme.github.io/ (M. Verano Merino); https://softwarediversity.eu (B. Baudry)
0000-0003-2278-1365 (M. Verano Merino); 0000-0002-4015-4640 (B. Baudry)
© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

Conventional software engineering techniques (e.g., static analysis and Mining Software Repositories) could help us understand how to document, maintain, and preserve this software, but they rely on clean datasets. Creative code on platforms like GitHub or OpenProcessing are overwhelmingly populated by educational projects (e.g., tutorials and homework). Moreover, SE techniques rely on structural signals such as commit frequency, repository metadata, or contributor activity. These metrics are not appropriate for software-based artworks because they are often written by a single contributor with a single commit [3]. What distinguishes a code repository that hosts a genuine software-based artwork from a classroom code repository is creative intent. While centuries of philosophical and art-historical discourse emphasize that the definition of "art" is inherently subjective and contextual, software evolution research requires an operational construct to isolate non-traditional software systems. Rather than proposing an ontological definition of art, our goal is pragmatic: *distinguishing self-directed, open-ended creative applications from structured educational boilerplate, exercises, and tutorials.* This is an intangible implicit construct hidden across algorithms, code structure, rendered visual output, and the author's documentation. For instance, Figure 1 shows an example of a program that has the same structural code, but the visual output is very different.

This vision paper explores whether a multimodal (large language and visual language models) AI system, operating with human-in-the-loop validation, can act as an automated interpretation assistant of dynamic code. This will allow us to curate a dataset of software-based artwork repositories and explore AI reasoning capabilities on non-standard, subjective domains such as creative-intent identification. Traditional supervised learning models are unviable for this challenge because they suffer from a cold-start problem due to the lack of labeled training data and rely on statistical feature matching rather than semantic reasoning. Inferring creative intent requires multimodal reasoning across source code, visual output, and documentation.

2. Background & Related Work

2.1. Domain-Specific Software Evolution

Software engineering maintenance research traditionally focuses on enterprise systems, open-source infrastructure, or mobile applications. However, domain-specific ecosystems such as audio software (e.g., VST plugins) or visual art [4] operate under different development dynamics. Andrei et al. [3] showed that audio plugins suffer from high dependency lock-in (e.g., JUCE) and low long-term repository freshness, leading to rapid software rot. Net art extends this problem further: maintenance is rarely performed by dedicated software engineers, yet these artifacts are acquired by cultural institutions or private collectors who expect multi-decade longevity.

2.2. Web Primitives and the Net Art Dependencies

Unlike desktop creative software that relies on stable host OS environments (e.g., Digital Audio Workstations), net art relies on web-based technologies (e.g., `p5.js`, WebGL, Web Audio API, DOM events). This shifts the maintenance burden to an unpredictable, constantly evolving runtime environment: browser vendor engines, canvas rendering pipelines, dynamic external media hosts, and hardware peripherals (e.g., displays, microphones, webcams). When browser APIs shift or deprecate features (e.g., modern autoplay policies blocking Web Audio API initialization), net art pieces break silently without any changes to their underlying source code.

2.3. Mining Non-Traditional Repositories & Noise Ingestion

Mining software repositories in creative ecosystems introduces severe dataset contamination threats. Baudry et al. [4] highlighted the vast presence of art and creative code in global software archives like Software Heritage, yet isolating high-intent artifacts remains an unsolved problem. Traditional repository filtering heuristics such as commit count, fork count, star count, or issue activity are not

useful in artistic environments because artists frequently build fully realized, static artworks in a single commit or hosted editor session[3]. Without semantic classification that accounts for creative intent, empirical software engineering studies on creative code risk drawing conclusions from classroom tutorials rather than genuine software evolution.

3. Codebook: Noise vs. Artistic Intent

Defining software-based artwork for empirical research requires navigating the inherent subjectivity of artistic evaluation. In software engineering, mining archives like Software Heritage or GitHub relies on functional proxies (e.g., commit messages, issue tracking) that fail in creative ecosystems [4, 5]. To operationalize this domain without making normative claims about aesthetic value, we draw on empirical repository curation methods [5]. We define categories based on *pedagogical and functional context* (e.g., whether a project fulfills a rigid assignment template versus exploring an open-ended algorithmic space) rather than evaluating artistic merit.

Existing platforms such as GitHub allow users to share their coding repositories with other users to collaborate or to serve as a backlog for software development. Similarly, there are other platforms that focus on storage and collaboration in software-based artwork projects such as OpenProcessing. These types of platform are beneficial because users can store and share their work there with the entire world or with a particular group of collaborators. These platforms mix students, hobbyists, and professionals work on one platform. However, when we want to study a particular ecosystem, this becomes challenging. For example, in OpenProcessing, a large portion of projects are tied to educational activities (e.g., course assignments and exercises) [6]. A similar trend is noted by Baudry et al. [4] in the Software Heritage archive. Therefore, any claim about "what creative coders do or how do software-based artists work" is confounded if the sample is dominated by educational submissions.

Addressing software-based artwork as a binary classification of "Art/No Art" is not appropriate in this case because we would lose critical nuances in artistic environments, including technical elements, aesthetic quality, and artistic elements. Therefore, following Kalliamvakou et al. [5] strategy, we define *mutually exclusive primary labels* to force one label per sketch [1], and a secondary *continuous Self-Directed Intent Score (1-5)* that allows us to capture borderline cases without losing information.

Primary Categories

CREATIVE-CODING: Reflects an original creative or technical idea pursued by the author for its own sake. Including personal generative art, an experiment, and a visual/audio piece.
VIDEO-GAME: Interactive system with explicit rules and feedback loops, designed with defined win/loss states, goals, or measurable progression metrics, where user input influences the outcome.
DESIGN: An interactive system engineered to fulfill a specific functional outcome, facilitate user navigation, or communicate branded information through structured layout and aesthetic hierarchy.
HOMEWORK: Satisfying a specific, gradable course requirement.
TUTORIAL: Reproducing a known tutorial (e.g., a Coding Train episode), is explicitly labeled as a walkthrough/demo/example.
EXERCISE: Practicing a technique outside a class structure ("trying out perlin noise," "practicing OOP")
BOILERPLATE: Default `p5.js` web editor template, blank canvas, broken/incomplete sketch, or a test/throwaway file with no discernible creative or pedagogical content.
DERIVATIVE: Forking ("remix") of another sketch with no or trivial code change (parameter tweak only), or an account that exists primarily to re-host/collect others' work. Subbaraman et al. [6]
AMBIGUOUS: Cannot be confidently placed in the previous categories (e.g., code present but instructions blank, sketch unrenderable, or genuinely on the boundary between two categories even after discussion).

[1] In the Processing and `p5.js` ecosystem, a sketch is a program written in Java and JavaScript, respectively.

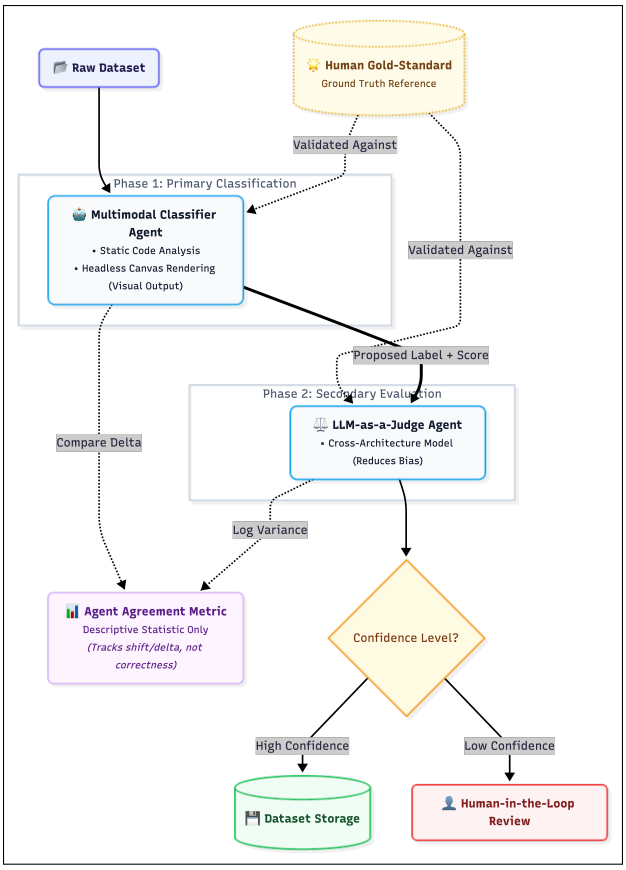


Figure 2: Architecture of the proposed multimodal pipeline.

Creative Intent Score

After assigning a Primary Category to each sketch, evaluators assign a score from 1–5 measuring: "To what extent does this sketch reflect open-ended, self-directed development versus a rigid, template-driven assignment brief?"". The scale is defined as follows:
1 - Fully Template-Driven: Exact copy of a tutorial or assignment brief without variations.
2 - Mostly Assignment-Driven: Minimal structural variation beyond the assignment brief.
3 - Mixed: Meets an assignment brief, but adds a visible personal programmatic or aesthetic extensions.
4 - Likely Self-Directed: Minor ambiguity, but demonstrates agency beyond standardized coursework.
5 - Self-Directed personal expression: No evidence of an assignment driving it.

Authentic Software-Based Artwork

Based on the primary categories and the Self-Directed Intent Score, we defined an authentic creative-coding artwork as a sketch that (i) was not produced primarily to satisfy a graded course requirement, (ii) reflects an idea, aesthetic, or technical choice attributable to the author rather than a copied/templated exercise, and (iii) is not a content-free placeholder, test, or boilerplate project.

4. Multimodal Agentic Dataset Curation

To address the problem of curating software-based artwork at large scale, we propose the following multimodal agentic pipeline as shown in Figure 2.

The architecture is divided into four main components: Raw Dataset, Human Gold-Standard, Curator Agent, and Judge Agent. The process starts with an initial dataset that contains a raw number of repositories that contain software-based artwork. We will start with `p5.js` projects because it is a

popular open-source creative coding ecosystem [4]. For each of the projects, the curator agent will identify a *Primary Category* and *Self-Directed Intent Score* (Section 3). Based on these two metrics, the Judge Agent will validate the outcome. It is important to note that both agents are validated against the Human Gold-Standard, never against each other. Agent-vs-agent agreement is reported only as a secondary descriptive statistic (e.g., to characterize how much the judge actually changes), not as evidence of correctness. Finally, we evaluate the confidence label of the classification and check: if there is a high confidence in the classification, we store it; otherwise, we label it to be manually inspected by an expert.

4.1. Raw Dataset

Repositories collected from a platform that stores software-based projects (e.g., GitHub, OpenProcessing, or Software Heritage). This dataset represents the entry point to the curation pipeline.

4.2. Human Gold-Standard

Assuming an imbalance in the raw dataset towards educational material, we aim to get tighter precision on the rarer categories (e.g., CREATIVE-CODING), we conduct a stratified sampling with oversampling of sketches already flagged by simple heuristics (e.g., know artworks or artists) plus a separate purely-random stratum, so "rare categories" are not underrepresented in our Human Gold-Standard. For the labeling task, two independent coders trained on the codebook (Section 3) will conduct a calibration round (10-20 sketches, discussed jointly) before independently annotating the Human Gold-Standard. After coding the sample dataset, we report the agreement between coders using the Cohen's kappa coefficient on primary label. Finally, this gold-labeled sample is the only ground truth used to validate the pipeline. It is never replaced by agent-judge agreement.

4.3. Curator Agent

The curator agent uses static analysis (e.g., parsing, AST analysis, cyclomatic complexity) combined with headless visual execution (capturing canvas output). This addresses the domain reality. The rationale behind the multimodal approach is that in the domain of creative coding, identical code structures can yield radically different visual behaviors, and simple code can produce intricate generative art as shown in Figure 1. For each sketch, the agent will report a single Primary Category, Self-Directed Intent Score, one or two sentences citing specific evidence for the chosen category, and a confidence value for the classification between 0 and 1.

4.4. Judge Agent

This agent utilizes a distinct model family/architecture evaluated against our manually curated Human gold-standard to reduce systematic LLM bias and flag ambiguous cases that required further attention via human inspection. If the Judge Agent disagrees, it has to propose its own Primary Category and a motivation for choosing that category.

It is essential to clarify the role of the human coders and the LLM-as-a-judge agent. Neither human evaluators nor agents act as aesthetic critics determining aesthetic value or cultural significance. Instead, they perform multimodal pattern recognition across code structure, visual output, and repository metadata to detect indicators of educational scaffolding (e.g., standardized variable names, shared assignment instructions, boilerplates) versus independent creative coding design. The human-in-the-loop validation ensures inter-coder reliability on these operational markers rather than standardizing subjective aesthetic preferences.

4.5. Validation Plan

To validate the proposed pipeline, we will first run the Curator Agent on the same sketches as in the Human Gold-Standard. Based on this, we will report and compare different metrics (e.g., precision, recall, F1 per category). Also, based on the Primary Category assigned to each sketch, we will look into the confusion Matrix.

Judge Agent vs. Human Gold-Standard. We run the Judge Agent over the same sample's Curator outputs (using the human label from the Human Gold-Standard as the reference, not the Curator's). With this, we measure the judge's own precision/recall against Human Gold-Standard, and separately report how often the judge's "agree/disagree" call was itself correct (i.e., did the judge correctly identify the cases where the Curator was wrong?).

Error analysis: For each case, in which there was a disagreement between the Curator and the Human Gold-Standard, we manually inspect those cases to determine possible systematic confusions (e.g., is *EXERCISE* routinely mislabeled as *CREATIVE-CODING*? Is *HOMEWORK* under-detected when the class metadata is missing?). If this is the case, this signals that we must revisit our prompts or our codebook to distinguish between Noise and Art (Section 3).

Threshold before scaling to the full dataset: To proceed to analyze the complete dataset, we will first look to achieve a $macro - F1 \geq threshold$. The exact value will depend on the data.

5. Vision for Software Engineering Maintenance & Art Preservation

After successfully executing the pipeline described in Section 4, we can study in depth how software-based artworks can be preserved and maintained.

Once the curated dataset is established, it unlocks systematic software engineering research in non-traditional domains. This enables researchers to study the ecosystem of software-based artwork, including their dependencies and runtime environments. We outline the research roadmap based on two main pillars: *Empirical Software Engineering for new Media Arts* and *AI & Methodological Frontiers*.

Empirical Software Engineering for new Media Arts

- **Dependency & API Decay:** Tracking how browser evolution, canvas/WebGL deprecations, and audio API updates break legacy interactive artworks over a specific time horizon (e.g., 5-10 years).
- **Supply Chain & Runtime Packaging:** Documenting hardware and software dependencies (e.g., webcams, screen resolution, local storage, third-party libraries) to develop automated tools to derive Software Bills of Materials (SBOMs) that include the different type of dependencies present in software-based artworks.
- **Maintenance & Refactoring Behaviors:** Comparing how creative coders handle technical debt and maintenance compared to traditional open-source software engineers.

AI & Methodological Frontiers

- **Interpretable Decision Models:** Exploring interpretable decision models as lightweight, low-resource alternatives to foundation LLMs for fine-grained software-based artwork classification.
- **Programmatic Intent Verification:** Define behavioral and aesthetic invariants for artistic intent without imposing algorithmic gatekeeping on creative expression.
- **Aesthetic & Visual Bias:** Addressing LLM visual evaluation bias (e.g., over-indexing on complex geometric visualizers while misclassifying conceptual or minimalist art.)
- **Feasibility of Characterising Artistic Intent:** Validating whether AI multimodal Interpretation Agents align with the human gold-standard across varied creative-coding styles.

Declaration on Generative AI

During the preparation of this work, the authors used Gemini 3.6 flash to: improve text flow, grammar and spelling. Further, the authors used the same model to create the mermaid code to render Figure 1. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publications content.

References

[1] M. Verano Merino, sw-artwork-equivalence, https://github.com/analogue-intelligence/sw-artwork-equivalence, 2026.

[2] C. Paul, Digital art, Thames & Hudson, 2023.

[3] B. Andrei, M. Verano Merino, I. Malavolta, The ecosystem of open-source music production software - a mining study on the development practices of VST plugins on github, in: 2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR), 2025, pp. 513–525. doi:10.1109/MSR66628.2025.00085.

[4] B. Baudry, Y. Gamage, N. G. Fernandez, L. MK, R. P. Mendieta, R. S. Pinter, S. Zacchiroli, Art in humanity's code, IEEE Transactions on Software Engineering (2026). URL: https://arxiv.org/abs/2608.05174.

[5] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, D. Damian, An in-depth study of the promises and perils of mining github, Empirical Software Engineering 21 (2016) 2035–2071.

[6] B. Subbaraman, S. Shim, N. Peek, Forking a sketch: How the openprocessing community uses remixing to collect, annotate, tune, and extend creative code, in: Proceedings of the 2023 ACM Designing Interactive Systems Conference, 2023, pp. 326–342.