POLITECNICO DI TORINO
Repository ISTITUZIONALE

The Art of Creating Code-Based Artworks

Original
The Art of Creating Code-Based Artworks / Verano Merino, Mauricio; Saenz, Juan Pablo. - STAMPA. - (2023), pp. 1-7. (Intervento presentato al convegno ACM CHI Conference on Human Factors in Computing Systems 2023 tenutosi a Hamburg, Germany nel April 23-28) [10.1145/3544549.3585743].

Availability:
This version is available at: 11583/2976495 since: 2023-05-03T15:24:22Z

Publisher:
ACM

Published
DOI:10.1145/3544549.3585743

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
ACM postprint/Author's Accepted Manuscript, con Copyr. autore

(Article begins on next page)

12 October 2024

The Art of Creating Code-Based Artworks

Mauricio Verano Merino*
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
m.verano.merino@vu.nl

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

ABSTRACT

Programming has become an artistic medium for artists; it provides expression possibilities built on top of the computer's interactivity and multimedia features. However, implementing code-based artworks encloses defining characteristics that differ from traditional programming. This paper reports on an in-depth interview with 5 code artists with diverse backgrounds, levels of experience, and working on different code-based artistic expressions. Through their experience, we identified characteristics and commonalities in their development process, the tools they use, their sources of inspiration, and their expectations. Accordingly, we reflect on the particularities of Creative Coding, indicate commonalities with other domains, and suggest opportunities and challenges for HCI researchers and practitioners in proposing tools to support it better.

CCS CONCEPTS

• **Human-centered computing → User studies**; • **Applied computing → Media arts**; • **Software and its engineering**;

KEYWORDS

creative coding, code-based artworks, code artists, interviews

ACM Reference Format:
Mauricio Verano Merino and Juan Pablo Sáenz. 2023. The Art of Creating Code-Based Artworks. In *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems (CHI EA '23), April 23–28, 2023, Hamburg, Germany*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3544549.3585743

1 INTRODUCTION

New technologies enable different forms of communication and expression. Computers were created as tools for fast calculations and automatization, and they have evolved into a medium for human expression [30]. Accordingly, software has become an artistic medium that enable artists to produce dynamic forms, process gestures, simulate natural systems, and integrate various media including sound, image, and text [33]. While computer programming used to be an esoteric skill for engineers and scientists not so long ago, in recent years, people from diverse backgrounds (e.g., designers, artists, poets, and musicians), are also creating software [15]. Nowadays, programming is also an artistic medium [19], and several programming languages and tools for code artists have been developed [15] (*e.g.,* Processing [30], p5.js [21], openFrameworks [28], Cinder [17], TouchDesigner [14], Sonic Pi [29], Max/MSP [20], vvvv [38], Pure Data [12], Nannou [24], and OPENRDR [27]).

Programming with artistic purposes has been portrayed under the umbrella term *Creative Coding*: a discovery-based process consisting of exploration, iteration, and reflection, where code is used as the primary medium to create a wide range of media artifacts designed for an artistic context [22]. With respect to traditional programming, Creative Coding has a different development process and poses distinctive challenges, technically and conceptually. On the one hand, it emphasizes the expressivity of computer programming beyond something pragmatic and functional [18], and its primary goal is to build artwork in line with the artist's expressive intentions. On the other hand, while becoming proficient with coding is already challenging for Science, Technology, Engineering, and Mathematics (STEM) students, programming with expressive purposes requires artists to overcome a steep learning curve. Technically speaking, they must become proficient in several topics, such as image processing, events, network communication, object-oriented programming, control structures, functions, and graphics transformations. Additionally, from the conceptual point of view, code-based artworks commonly embrace topics as diverse as Newtonian physics, cellular growth, and evolution to emulate physical world occurrences [33]. Therefore, code artists must get familiar with vectors, forces, oscillation, particle systems, fractals, genetic algorithms, and neural networks.

In this scenario, a challenge for human-computer interaction researchers and user interface designers is to design and build technologies that support creativity [34] and expanding programming beyond the confines of computer science or software engineering [35]. However, to achieve such goals is compulsory to gather, directly from the code artists' perspective, an accurate understanding of their development process, the tools they use, their inspiration and documentation sources, and their expectations. Previous works provide valuable insights into how visual artists [16], musicians [4], DJs and VJs [36] approach and appropriate software in their artistic practice. Nevertheless, to the best of our knowledge, there is no research focused on identifying fundamental characteristics and commonalities among the **development process of code artists** with diverse backgrounds, levels of experience, and fields of interest, in creating code-based artworks.

This paper reports a user study consisting of an in-depth interview with 5 code artists with diverse backgrounds, years of experience, and working on different code-based artistic expressions. Findings show that all the participants experience Creative Coding as highly iterative, exploratory, and open-ended. Furthermore, code artists commonly integrate their professional backgrounds

*Both authors contributed equally to this research.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
CHI EA '23, April 23–28, 2023, Hamburg, Germany
© 2023 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9422-2/23/04.
https://doi.org/10.1145/3544549.3585743

into their artworks. Additionally, we identified a lack of support for some Creative Coding characteristics in existing tools and indicated possible research directions to support the development process better. Accordingly, more research is required to further validate our results with a larger population.

2 INTERVIEWS

The interviews were structured around six aspects referred to the code artists' experience: (i) how do they design the code-based artwork they are about to program; (ii) which strategies do they adopt when starting to create a new artwork; (iii) where do they tend to take inspiration; (iv) where do they find documentation to get technically or conceptually proficient; (v) what are the most challenging aspects regarding the programming language they commonly use; and (vi) which aspects do the artists like the most and least about the development environment they use. The precise set of questions can be found in Appendix A.

2.1 Participants

The selection criteria concerned code artists that, independently from their formal education background, create artworks that directly result from executing their text-based code. They were recruited through personal contacts and from exhibitions and workshops attended the authors. Four participants self-identified as male, and one self-identified as female, and the average age of participants was 33.4 years (min = 28, max = 40, SD = 4.5). The participants are based in the Netherlands (2), Denmark (1), Germany (1), and the United Kingdom (1). Before conducting the interviews, the participants were given a brief online questionnaire (using LimeSurvey) to gather demographic data, information regarding their background and experience in Creative Coding projects, and the technical resources they use. Table 1 lists the participants, including their names, years of experience (Yrs), background, and fields of interest. When asked about the programming languages that they use, JavaScript and Processing were the most popular (5), followed by Java (3), Python (3), C++ (1), and SuperCollider (1). Each participant could indicate more than one programming language or add one if it was not on the list of suggested ones.

Furthermore, in the pre-interview questionnaire, the participants were asked to rate their confidence level on a Likert scale with the most prominent Creative Coding development environments ranging from *"not confident at all"* to *"completely confident."* Each participant could indicate more than one tool, and a *"not applicable"* option was available for each environment. Additionally, participants could add other development environments not included in the list. The tools in which the participants indicated having a good level of confidence (from *"slightly confident"* to *"completely confident"*) were: Processing (5), p5.js (5), openFrameworks (2), Processing Python (2), Pure Data (2), TouchDesigner (2), OPENRDR (1), and Processing Android (1).

2.2 Procedure

Due to geographic reasons, all the interviews were conducted online via Zoom in the fall and winter of 2022. On average, each session lasted 57 m (min = 49 m, max = 1 h 07 m, SD = 7 m), and all the interviews were in English. They were semi-structured interviews in which, upon the participants' responses, we asked them to deepen their answers. Specifically, they were encouraged to provide qualitative justifications and specific anecdotes. Additionally, we prompted them to reflect on their development process in as much detail as possible, specifying all the tasks they completed.

The interviews were audio recorded (with the prior consent of the participants), transcribed with the Microsoft Word transcript functionality, and subject to qualitative thematic analysis, using an inductive approach [3, 11], for commonly recurring themes. Then, each of the authors reviewed all the transcripts and generated our initial codes (independently and without sharing them to avoid influencing the other author's coding criteria). Finally, we shared this initial subset of codes, discussed the commonalities and the divergences among them, graphically mapped them, identified the most revealing data extracts, and derived a set of four themes.

3 RESULTS

This section presents the four main themes that we derived from the thematic analysis, namely: (i) artists sources of inspiration (You Code as You Live), (ii) Creative Coding defining characteristics (Destination Unknown), (iii) development tools (Development Environment Agnostic Approach), and (iv) artists' mechanisms to technically deal with the Creative Coding versioning (Creative Versioning).

3.1 You Code as You Live

We delved into participants' sources of inspiration to better understand the Creative Coding development process. As a result, we could determine that most of the participants considered that looking for inspiration in other code-based artworks might interfere with the originality of their work (in Tim's words, "I think if you want to develop a very individual style, you should not orient yourself too much on others' code or tutorials."). Besides, understanding someone else's code might be painful and time-consuming. For instance, Sumeet stated, "understanding someone else's code is really painful. I can waste one hour looking at someone else's code. Instead, I would prefer to write my own code." So, instead, participants integrate concepts from diverse disciplines into their coding practices rather than constantly exploring other code-based artworks.

In this sense, Joana's development process: "starts in choreography and then goes into the programming languages." Specifically, she explains:

> *I usually look into choreographic concepts. If there is something within choreography that I am interested in exploring, then I try to see how I can apply that within a web interface. For example, how will that movement take shape? Or how will we experience that same concept in this digital environment? (...) I also name functions after choreographic concepts.*

Sumeet's development process integrates into his code-based artworks concepts from his background in physics and optics. Besides satisfying the pursuit of his artistic purpose, it gives them a sense of uniqueness. In his words, "it is just no fun replicating other people's work. You can take elements but take something

Table 1: Interviewed code artists

Name	Yrs.	Background	Fields of Interest
Felipe Ignacio Noriega [25]	20	Music Composition	Live Coding, Sound Sculpture
Joana Chicau [5]	9	Communication Design, Choreography, and Performance	Live Coding, Experimental Interfaces Design
Michael Kreß [13]	2	Graphic Design, Editorial Design, and Typography	Interactive Font Generation Installation, Generative Typography
Sumeet Rohilla [32]	2	Mathematical Physics and Applied Optical Sciences	Abstract Data Visualizations, Real-Time Light and Audio-Reactive Installations
Tim Rodenbröker [31]	8	Arts and Graphic Design	Graphic Design & Creative Coding Teaching

and make it your own and add a layer on top." In his experience as an artist, he noticed that "people use random numbers a lot, but can I use the randomness in Biology to create art? So I knew that the gene sequences exist in .txt files." Additionally, in his real-time interactive light installations, he is constantly "going back to all my master's courses, exploring what I learned from them, and seeing if I can use that idea creatively." Similarly, Michael often sees that "inspiration comes from mathematics because I find that a lot of times mathematics are closely connected to design or arts." Therefore, he might find inspiration in a "YouTube video where someone is talking about mathematics phenomenon, and I try to research and see how to interpret that idea in my way."

Felipe summarizes his development process as "I use code as my instrument instead of writing for the string quartet or playing piano or stuff like that." He uses the next analogy to illustrate how close his music background can be to Creative Coding:

> *The traditional way of writing music would be organizing sound events in the time by writing them on paper using a music notation, which is a form of code. Then musicians, who can read the scores, interpret them, they are human interpreters. It is just like code or algorithms that special programs or domain-specific languages can interpret to produce sounds."*

3.2 Destination Unknown

While Creative Coding has been widely adopted as the most prominent term to describe the practice of coding with artistic purposes, the adjective *creative* is vague. Joana claimed that "coding is creative regardless of its application, so I find that naming a little bit troublesome." Therefore, at the beginning of the interviews, we asked the participants to share their meaning of Creative Coding trying to identify how its defining characteristics are present in their artistic practice.

In their experience, the artists provided consensus answers around three aspects: firstly, the absence of requirements or a problem to solve, secondly the uncertainty of the directions that the artwork might take; and thirdly, interestingly, the fact that the code assumes an active, visible role in the artwork rather than a purely utilitarian function. Regarding the absence of requirements, Felipe describes it very accurately by stating: "in traditional software writing you have very specific requirements (...); sometimes with creative coding you don't even know what it's going to do." Likewise, Joana enjoys that she is "just trying things out, and it removes the pressure of you arriving at any potential conclusion."

The absence of a predefined arrival point sets the tone for a *trial-error* process with many iterations in which real-time feedback is highly appreciated, and the final artwork is reached by continuously adapting, integrating, or rejecting the code. In Felipe's case, whose artistical expression concerns sound sculptures, he finds that:

> *You sort of start and then see where it's taking you by trial and error and keep a real-time feedback loop. So you're creating an algorithm that's producing certain results. If you like those results, you continue exploring them. If maybe you don't like something, you change the algorithm and do continuous iterations in real time until you produce a sound sculpture.*

In the same vein, Michael states that Creative Coding is about embracing uncertainty, "it's really about learning to handle the techniques, the coding ideas, and then seeing where it takes me". Tim, for his part, reinforces this idea by talking about a *creative dialog*. In his words, "it is a creative dialog like a painter on a canvas but with code (...) and it is, first of all, completely free of a predefined motive."

Concerning the visible role that the code might have in Creative Coding artworks, Joana commented: "In some performances, I print parts of the code because then the audience can sort of read, and I think there's something quite nice about the legibility of it. I think it helps give a different understanding of what's happening." Furthermore, in one of Felipe's performances, a pianist uses CodeKlavier [26] to generate source code by playing the piano. In this scenario, the code is not merely a means to reach a working artwork, but the code itself is the artwork: "Instead of creating software to make music, we're trying to use music to create software, so we're turning things around. So we must make it clear to the audience that the piano is generating the code."

3.3 Development Environment Agnostic Approach

Rather than sophisticated functionalities, Joana, Michael, and Tim appreciate that p5.js and Processing are well-documented. In Tim's words, Processing "is not the software to do everything but to learn about the limitations; it teaches you a lot. It has amazing documentation, and people have been working on it for 20 years." Joana said: "I think p5.js has a really good learning curve. It is super

well documented." In Sumeet's development process, when more sophisticated functionalities are required, he relies on software that might integrate seamlessly with the devices he wants to use in his interactive installations. He explains "if I want to learn an algorithm, I will go back to Processing because I can understand and visualize it. But once I have a sketch, I will mimic it in TouchDesigner because it has a faster, better API in which you can control a lot of stuff. I can integrate audio and bring in sensor data."

Furthermore, contrary to our expectations, participants were not particularly tied to a specific development environment. Indeed, in Joana's case, we found an unconventional use of development tools: she used the browser inspector to create visuals in a live performance by changing the values of the HTML elements on a website. Finally, Tim summarizes his *environment-agnostic* approach by stating:

> *For me, Processing and Creative Coding is more like a school of thought. It's not about teaching people to use high-definition technology but to teach them how to think. Learning how the computer works, how programming languages work, and how to speak to a computer. Later, you can pick any technology and learn it much more quickly when you master these basic concepts.*

As artists become proficient with the basics concepts, they would appreciate additional guidance and support from their tools to understand the programming language better and increase their coding speed. For instance, Joana and Michael mentioned: "I know that if you type in something on Visual Code, it understands you are trying to write a function, it builds everything for you, and then you change it up. I have seen other coding environments with a glossary like not always having to go necessarily online to figure out how to do things." In this sense, Michael finds that "in Processing, in a way, you have to do everything manually. It can be a nice learning process. But I am starting to work faster, and when I know what I am doing, it would be nice to have some functionality to help me out. I am really missing a good runtime debugger or autocomplete function."

Similarly, in his teaching experience, Tim has identified that his students "mostly feel overwhelmed by this blank window. Processing does not tell you anything about how to use it." However, in his opinion, even if "it looks like a flood of things you have to know in the beginning, that is not really true. It is all about practicing."

3.4 Creative Versioning

As depicted in Section 3.2, all participants agreed on the exploratory nature of Creative Coding. From a technical point of view, such nature implies that artists constantly create source code files to implement and experiment with a particular idea or concept. If they like the result, that file can become: (i) the starting point for new artwork, (ii) something that can be integrated into a more extensive artwork, or (iii) a proof of concept they might use in future artworks. As Tim states, it is about having "many different ideas that you can later use as a foundation for a specific project". In any of these three cases, artists have to deal with several versions of their source code files. They have to find mechanisms to categorize and arrange the files, so they iterate, return to a previous state, and reuse them in the future.

Based on the interviews, we could determine, on the one hand, that available tooling is not designed to support the exploratory nature. Consequently, each participant has developed their own strategies. On the other hand, we identified that the Version Control Systems (VCS) (e.g., Git or Subversion) are not employed by the participants due to the steep learning curve to get proficient in using them and the fact that iterations on the files are happening remarkably fast.

Regarding the strategies adopted by the artists to deal with versioning, Tim categorizes his work by topics ("So when there is a specific topic I am dealing with, I create, let us say, a parent folder where I put all the sketches"). Additionally, having a visual representation of the code execution is fundamental for him to locate the sketches quickly. In his words: "If I am writing a sketch and like the result, I save it with another name. Then, I always run a video record function from the video export library, which creates a thumbnail of my sketch exactly in the folder where all my sketches are." Similarly, for Michael, having a visual representation is fundamental. His strategy, however, is to publish his work on Instagram. In this way, when he wants to find a sketch, he compares the post's publication date to the file creation date. "If it's like a small sketch, I have a folder of daily sketches, and I post them on Instagram and when I have a new idea, I probably go back to the sketch and maybe change some variables and then save a new image."

Sumeet, for his part, categorizes its files by creation date. When he is working on a project, he creates a new folder each week and dumps the files from the previous week that he intended to use. Moreover, he uses an external file to document his learning and development progress:

> *I keep a document (OneNote notebook) every week where I put keywords and some learnings from that week. Additionally, I have another notebook for each project, and if I find something more general will just put that in that main document. So there are multiple levels of documentation going on, and if I am aware, I will just update those documents.*

Regarding the reasons that prevent code artists from relying on traditional VCSs, participants expressed a steep learning curve, and using these systems requires some expertise to deal with version conflicts. As Joana stated: "I am not a software developer, and sometimes there are problems I am unsure how to solve. Especially when it comes to merging files, you know? And then, obviously, I can feel a bit anxious about what to do." The second reason artists refrain from using VCS is that their development process is highly iterative. It commonly consists of changing tiny portions of code or experimenting with several parameter values. Therefore, iterations are happening fast, and manually performing commits on every change is not practical or useful. As Michael commented: "I always wanted to use GitHub more appropriately, but I found it does not suit my work style. For the changes that I do on these sketches, it's just not really worth it." Referring to the commits, Sumeet said: "sometimes it is a very small piece of code. It is like: 'why do I care about Git controlling this piece of code in a scheme of big things,' right?". In this context, Felipe finds desirable "Somehow to find a better way (to commit changes), like having some automatic commits every five seconds."

4 DISCUSSION

The interviews enabled us to ascertain how **Creative Coding characteristics materialize in practice**, notwithstanding artists' background, experience, tooling, or fields of interest. In line with the Creative Coding definitions available in the literature, we verified through the participants' descriptions and anecdotes that the goal is open-ended, and they discover the ultimate direction that the artwork will take through the process of programming [2, 23]. More interestingly, we found that despite the steep learning curve, the participants did not rely on copy-pasting others' code; they instead relied on the language's documentation. Furthermore, while their reasoning was not about reaching a predefined final solution, it was about integrating their background knowledge into the code they were writing.

Similarly, we could establish, based on Joana's and Felipe's work, that, contrary to traditional software development, where the code is a means to build a functional application, in the Creative Coding domain, the **source code becomes a piece of art in itself**. In their work, they make explicit the dialogue between the code, the choreography (in Joana's case), or the music composition (in Felipe's case). Consequently, the source code is not behind the scenes: it is always visible to the public as an essential component of their live performances. This observation is consistent with the Creative Coding domain of computer poetry [1, 7], in which code is not simply a practical artifact that produces an artwork but is a critical-aesthetic object in itself [35].

However, among the Creative Coding characteristics, we also found various **commonalities with other domains**, particularly with Data Science, whose tasks are highly iterative and exploratory. Data science activities often require heavy data exploration with different ways to manipulate the data [9]. Indeed, experts in this field struggle to keep track of their exploratory sessions, leading to lost work, reproducibility issues, and difficulties in effectively ideating [6]. Additionally, people working in Data Science (e.g., biologists, physicists, journalists, or financial analysts) may have theories and equations embodied in their source code that could benefit from additional explanations, such as formatted equations or images [41]. Similar to the code artists for each version of code-based artwork, for every useful model feature or insightful visualization data scientists create, there may be many less-successful features, plots, or analyses they have tried [10]. This scenario raises the question of whether Data Science methodologies and tools could be adopted to improve Creative Coding.

More concretely, since the literate programming approach (that interleaves prose with executable code and interactive and static output) has been widely adopted in Data Science and Computer Science education, we wonder if this approach might suit the exploratory nature of Creative Coding. We envision that with a tool inspired by literate programming, we might have in the same development workflow: (i) self-explanations on what the code does and the learned lessons (as Sumeet commented in Section 3.4); (ii) the successive versions of the code can be saved or discarded (as Felipe explained in Section 3.2), and (iii) the code's visual output (as Tim explained in Section 3.2). In this regard, the recent work by Horn *et al.* [8] elucidates how to borrow ideas from computational notebooks to support creative musical expression, including live performances.

Finally, the dialogue concerning **Creative Coding tooling** showed us that, rather than looking to increase their coding productivity or prevent them from writing code, participants' tool choice is influenced by the simplicity of the interface and the intuitiveness of their features. Code artists care more about the usability and user-friendliness of the tools. In fact, for this reason, all the participants were comfortable and positive with Processing.

Among their comments, we recognized two possible improvements in their current tooling. On the one hand, they would like more guidance to author programs and help them with the programming language features within their IDE. Autocompletion and syntax highlighting were commonly mentioned among Processing and p5.js users. However, apart from these features, it would be interesting to explore whether offering alternative notations like hybrid editors (e.g., textual and graphical) could benefit and enrich the experience of Creative Coding users, not by hiding the code but by providing alternative visualizations [39, 40].

On the other hand, we confirmed that managing versions of code-based projects and searching and navigating previous versions are still not supported by existing tools. Consequently, each participant had to find a way to manage versions of their work. Additionally, in Creative Coding, changes in the code are so frequent and small that existing version control systems do not suit the artists' needs and present a steep learning curve for them. Therefore, this observation raises opportunities and challenges for HCI researchers and practitioners in proposing tools to support version management in a more automated way without requiring a complex setup process. Indeed, Sterman *et al.* [37] have recently explored how creative practitioners use version histories in their domains and provide insight into future designs and uses of version control systems to support the creative process in general.

5 CONCLUSIONS AND FUTURE WORK

This paper presents an interview with 5 participants to understand the Creative Coding development process and its characteristics. Our results show that Creative Coding is a highly iterative, exploratory, and open-ended activity. Similarly, we identified that artists often integrate their professional backgrounds into their code-based artworks. Also, based on the participants' answers, we noticed that existing Creative Coding tools are powerful and appreciated by the community. However, they do not support some of the key characteristics of Creative Coding (e.g., exploratory and version management). In future work, we plan to recruit more code artists to determine whether the initial results remain true or they change. Finally, we aim to integrate our findings into some computer-supported creative tools.

ACKNOWLEDGMENTS

We want to thank the code artists—Felipe Ignacio Noriega, Joana Chicau, Michael Kreß, Sumeet Rohilla, and Tim Rodenbröker—who generously shared their time and insights with us.

REFERENCES

[1] Ishac Bertran. 2012. *code {poems}*. Impremta Badia, Barcelona, Spain.

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

[3] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology* 3, 2 (2006), 77–101. https://doi.org/10.1191/1478088706qp063oa

[4] Gregory Burlet and Abram Hindle. 2015. An Empirical Study of End-User Programmers in the Computer Music Community. In *Proceedings of the 12th Working Conference on Mining Software Repositories* (Florence, Italy) *(MSR '15)*. IEEE Press, 292–302.

[5] Joana Chicau. 2022. Choreo—Graphic-Design. https://joanachicau.com/. Accessed: 19-01-2023.

[6] Charles Hill, Rachel Bellamy, Thomas Erickson, and Margaret Burnett. 2016. Trials and tribulations of developers of intelligent systems: A field study. In *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 162–170. https://doi.org/10.1109/VLHCC.2016.7739680

[7] Daniel Holden and Chris Kerr. 2016. *./code –poetry*. CreateSpace Independent Publishing Platform.

[8] Mike Horn, Amartya Banerjee, and Matthew Brucker. 2022. TunePad Playbooks: Designing Computational Notebooks for Creative Music Coding. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 230, 12 pages. https://doi.org/10.1145/3491102.3502021

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

[10] Mary Beth Kery, Marissa Radensky, Mahima Arya, Bonnie E. John, and Brad A. Myers. 2018. The Story in the Notebook: Exploratory Data Science Using a Literate Programming Tool. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/3173574.3173748

[11] Michelle E. Kiger and Lara Varpio. 2020. Thematic analysis of qualitative data: AMEE Guide No. 131. *Medical Teacher* 42, 8 (2020), 846–854. https://doi.org/10.1080/0142159X.2020.1755030

[12] Johannes Kreidler. 2009. *Loadbang: Programming Electronic Music in Pd*. Wolke.

[13] Michael Kreß. 2022. Michael Kreß | Graphic Design and Creative Coding. https://www.kressmichael.de/. Accessed: 19-01-2023.

[14] Patrik Lechner. 2014. *Multimedia Programming Using Max/MSP and TouchDesigner*. Packt Publishing Limited.

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

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

[17] Rui Madeira and Dawid Gorny. 2013. *Cinder Creative Coding Cookbook*. Packt Publishing.

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

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

[20] V.J. Manzo. 2016. *Max/MSP/Jitter for Music: A Practical Guide to Developing Interactive Music Systems for Education and More*. Oxford University Press.

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

[22] Mark C. Mitchell and Oliver Bown. 2013. Towards a Creativity Support Tool in Processing: Understanding the Needs of Creative Coders. In *Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration* (Adelaide, Australia) *(OzCHI '13)*. Association for Computing Machinery, New York, NY, USA, 143–146. https://doi.org/10.1145/2541016.2541096

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

[24] Nannou. 2021. Home | Nannou. https://nannou.cc/. Accessed: 19-01-2023.

[25] Felipe Ignacio Noriega. 2022. Felipe Ignacio Noriega | Portfolio 2011-2019. https://felipeignacio.info/. Accessed: 19-01-2023.

[26] Felipe Ignacio Noriega and Anne Veinberg. 2022. The CodeKlavier: Appropriating the piano as a live coding instrument. *Rethinking the Musical Instrument* (2022).

[27] OPENRDR. 2022. OPENRDR. https://openrndr.org/. Accessed: 19-01-2023.

[28] Denis Perevalov. 2013. *Mastering openFrameworks: Creative Coding Demystified*. Packt Publishing.

[29] Christopher Petrie. 2022. Programming music with Sonic Pi promotes positive attitudes for beginners. *Computers & Education* 179 (2022), 104409. https://doi.org/10.1016/j.compedu.2021.104409

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

[31] Tim Rodenbröker. 2022. Start tim rodenbröker creative coding. https://timrodenbroeker.de/. Accessed: 19-01-2023.

[32] Sumeet Rohilla. 2022. Studio Sumeet Rohilla. https://www.sumeetrohilla.com/. Accessed: 19-01-2023.

[33] Daniel Shiffman. 2012. *The Nature of Code*. Daniel Shiffman.

[34] Ben Shneiderman. 2000. Creating Creativity: User Interfaces for Supporting Innovation. *ACM Trans. Comput.-Hum. Interact.* 7, 1 (mar 2000), 114–138. https://doi.org/10.1145/344949.345077

[35] Winnie Soon and Geoff Cox. 2020. *Aesthetic Programming: A Handbook of Software Studies*. Open Humanities Press.

[36] Anna Spagnolli, Diletta Mora, Matteo Fanchin, Valeria Orso, and Luciano Gamberini. 2020. Automation and Creativity: A Case Study of DJs' and VJs' Ambivalent Positions on Automated Visual Software. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/3313831.3376463

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

[38] vvvv. 2022. vvvv - a multipurpose toolkit | vvvv. https://vvvv.org/. Accessed: 19-01-2023.

[39] David Weintrop and Uri Wilensky. 2017. Between a Block and a Typeface: Designing and Evaluating Hybrid Programming Environments. In *Proceedings of the 2017 Conference on Interaction Design and Children* (Stanford, California, USA) *(IDC '17)*. ACM, 183–192. https://doi.org/10.1145/3078072.3079715

[40] David Weintrop and Uri Wilensky. 2018. How block-based, text-based, and hybrid block/text modalities shape novice programming practices. *International Journal of Child-Computer Interaction* 17 (2018), 83–92. https://doi.org/10.1016/j.ijcci.2018.04.005

[41] Yihui Xie. 2014. *knitr: A Comprehensive Tool for Reproducible Research in R*. Chapman and Hall/CRC, 29.

A INTERVIEW SET OF QUESTIONS

At the beginning of the interview, all the participants were told exactly: "we will ask you some open questions to get a sense of your creative process and the pros and cons of the technical resources you commonly use. Specific anecdotes and examples are very welcome!." Below we report the initial set of questions along with optional follow-up questions that we asked when deemed appropriate.

(1) Once you have a concept, how do you design the code-based artwork?
(2) When you are about to create a new code-based artwork, how do you begin?
(3) Do you usually take inspiration and work on code-based artworks available online?
  (a) What is the most challenging part of appropriating them?
  (b) Which strategies do you adopt?
(4) What opinion do you have about the development environment(s) you use?
  (a) What do you like the most about it/them?
  (b) Why did you choose it/them?
(5) Where do you find technical documentation (documentation concerning the programming language or the development platform)? Where do you find conceptual documentation?
  (a) Where do you search for resources to overcome the learning barriers you find while developing your code-based artworks?
(6) What are the elements that you find more challenging to understand from programming languages?

(a) Which language constructs (variables, functions, loops, objects) do you find more difficult to implement?

(7) Which tasks do you find inefficient, annoying, or complicated to perform in the development environment(s) you use?

(8) Do you use any library in your development environment? Which? What for?

(9) Do you use any version control system? If yes, which and why?

(10) Do you use any external devices to create your code-based artworks? If yes, which?