Mapping Creative Coding Courses: Toward Bespoke Programming Curricula in Graphic Design Education

S. M. Hansen
Department of Digital Design and Information Studies, Aarhus University, Denmark

Abstract

This paper presents a study of 30 syllabi gathered from introductory Creative Coding programming courses. A selection of the results concerning the courses' structure and content is presented and discussed. The majority of the analyzed courses exhibited evidence of being planned to adapt and submit graphic design topics to programming paradigms. Also, topics and algorithms of particular value to graphic design as a spatial practice were absent in many courses. Finally, most courses did not investigate visual output that is achievable only through computation. The present study argues that educators must adapt their Creative Coding syllabi and teaching materials to make programming meet the needs of graphic designers rather than the other way around. The findings in this paper provide a point of departure for a critical discussion among educators who wish to integrate programming in graphic design education.

1. Introduction

In the wake of the convergence of computer programming and graphic design, several scholars and practitioners have argued that there is a need for coding to play a larger role in the future education of graphic designers [Ami11; Pet12; Sau13; Tob12a; You01]. This move toward integrating computation into graphic design practice and education is paramount to engage and nurture a new generation of cross-disciplinary meta-designers who are as visually talented as they are technically proficient [Mad15].

Extending this discussion into classroom practice, design schools across the globe have begun revising their curricula to include courses in *Creative Coding*, which is a vague yet popularized term describing "a discovery-based process consisting of exploration, iteration, and reflection, using code as a primary medium, towards a media artifact designed for an artistic context" [MB13]. However, as an emerging practice, educators and researchers alike still only possess a shallow understanding of how programming should ideally be taught to an audience of visuospatial-inclined graphic designers. The lack of an established epistemological framework [TF17] inadvertently has caused many Creative Coding courses to be haphazardly planned on an uninformed basis. In an effort to mitigate this, design educators have drawn inspiration from programming courses offered within Computer Science, but, without proper adaptation, they unintentionally have made graphic design topics fit the structure and terminology of Computer Science.

Moving toward bespoke programming curricula that is adapted to fit graphic designers calls for investigation into and discussion of how these courses should ideally be planned, developed, and implemented. To facilitate an informed debate on the subject, an overview of the status quo of contemporary Creative Coding courses is needed. Examination of the literature reveals that no study to date has been conducted on this subject. Therefore, to fill this gap, this paper asks the question: "How are introductory Creative Coding courses that are designed to teach programming in a visual context structured, and what topics are covered?" To answer this question, the systematic mapping and content analysis of 30 representative Creative Coding courses were performed.

2. Collecting data

The first phase of this study involved conducting structured Internet search engine queries using combinations of chosen keywords that are essential to the topic of the study (see Table 1). The search was carried out using generic web search engines, Google and Bing, with the browser set to “private mode” to prevent the possible interference of past searches in the results. To prevent a bias toward courses taught in English, queries using translations of the keywords in several languages (i.e., German, Spanish, Portuguese, French, Italian, Danish, Swedish) were also made. Search results from the first five pages of each query were systematically evaluated to construct a gross list of identified courses. In the second phase, search queries using the previously mentioned keywords were made on code sharing websites that are frequently used by educators who teach programming in a visual context: github.com, codepen.io, and openprocessing.org (Main Repository & Class Section). All identified courses were added to the gross list.

Next, the content of each of the courses on the gross list was reviewed and measured against a set of criteria to determine if it was suitable for inclusion in the study:

- Offered by a university, university college, or trade school
- Taught within the past five years (2013-2018)
- Introductory level
- Teaches programming in a visual design context
- Detailed course syllabus is available
- Teaching materials are available (optional)
- Assignments and student submissions are available (optional)

Applying this search strategy yielded 30 courses qualified for in-depth analysis. The syllabus, teaching materials, and assignments from each course were downloaded to form the study's dataset.

DOI: 10.2312/eged.20191024

3. Analysis & Results

A homogenized dataset was developed using a spreadsheet to log 17 constituent parameters from each course (i.e., course duration, class size, scheduled lectures, number of teaching assistants, teaching methods, textbooks, programming environment). To establish a framework for analyzing the courses' structures and contents, an inductive textual analysis of the course syllabi and teaching materials dataset was conducted to identify recurring domain-specific topics relating to both Programming and Graphic Design. Twenty-seven programming topics and 19 graphic design topics were derived directly from the raw dataset through repeated examination without the use of theoretical perspectives or predetermined categories.

The identified domain-specific topics were used to construct a matrix with 30 rows (courses) and 46 columns (topics). The matrix was populated through a detailed examination of each course's syllabus and teaching materials to identify in which class each topic was first introduced and dealt with in-depth. The absolute class number (e.g., 8) and its relative position in the overall course (e.g., the 8th class of 24 total classes = 33.3%) were entered into the matrix. In cases where it could not be definitely decided if or when a particular topic was dealt with in the course, the cell was left blank. Color coding cells, using the relative position mapped to a specter ranging from green (0%) over yellow (50%) to red (100%), revealed the pattern shown in Figure 1. Green refers to "core" topics introduced early in almost all courses, whereas, red refers to advanced or specialized topics introduced late and sparsely across the courses. Next, the average order in which both programming topics and graphic design topics were taught was determined by sorting the relative position value in ascending order. The tabulated results are shown in Table 2. Below, a few of the most noticeable results relevant within the scope of this paper are discussed.

Processing and p5.js were the preferred programming environments. Of the analyzed courses, 37% used p5.js [MFR15], 33% used Processing [FR14], and 20% used both. Furthermore, 10% used lesser known JavaScript frameworks (basil.js, rune.js). Other popular Creative Coding environments (e.g., openFrameworks, Cinder, vvvv, Max, three.js) were used only in one of the analyzed courses, respectively; however, they often replaced or supplemented Processing and p5.js in advanced courses.

Debugging and error analysis techniques were only discussed as separate topics in half of the courses. As a major part of programming is hunting down bugs and fixing problems, failing to equip students with techniques to accomplish this will likely cause frustration among students who have to solve their assignments outside of their scheduled classes and, thus, will not have the opportunity to ask an instructor or teaching assistant for advice.

Recursion was introduced relatively late (64% into the courses), considering its ability to produce visually aesthetic results. Of the courses that introduced recursion, half of them discussed it solely as an abstract concept while the remaining courses explained recursion visually by implementing generic examples (e.g., Koch snowflakes, recursive trees). Only one course exemplified how recursion is used in actual graphic design artifacts.

Collisions, overlapping, and spatial arrangements were given little attention, considering that much of what graphic designers do is arranging elements on a surface. Of the courses, 20% addressed these topics; however, this was mostly done by emphasizing the math involved, thereby, failing to demonstrate how the topics could be practically applied in graphic design projects.

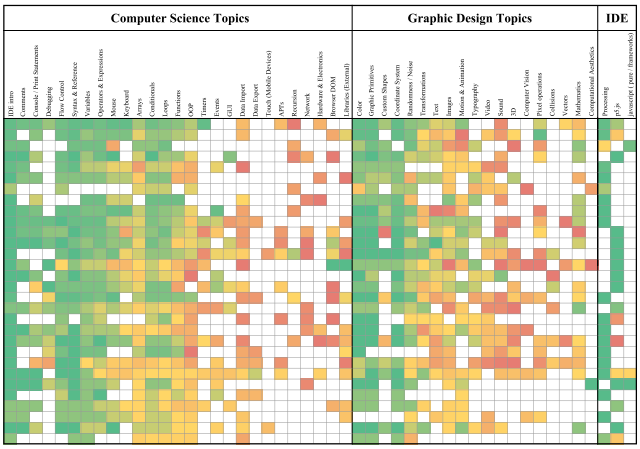

***Figure 1**: The populated course matrix, providing an overview of the analyzed courses (see Section 3 for additional information).*

Domain	Activity	Item
Visual	Programming	Curriculum
Graphic	Coding	Syllabus
Design		Course
Creative		Class

***Table 1**: Search queries were constructed by combining one keyword from each column.*

Programming Topics	
IDE intro	9%
Syntax & Reference	10%
Comments	12%
Flow Control	14%
Variables	17%
Operators & Expressions	18%
Conditionals	21%
Output: Console	21%
Loops	23%
Debugging	24%
Input: Mouse	26%
Input: Keyboard	33%
Functions	36%
Arrays	37%
Events	42%
GUI	47%
Timers	51%
OOP	51%
Input: Touch	57%
API's	61%
Libraries (3rd party)	62%
Browser DOM	62%
Data Import	63%
Data Export	63%
Recursion	64%
Hardware & Electronics	69%
Network	76%

Graphic Design Topics	
Coordinate System	12%
Graphic Primitives	13%
Color	14%
Shapes (Custom)	27%
Randomness & Noise	27%
Transformations	37%
Motion & Animation	38%
Mathematics	39%
Typography	39%
Text	45%
Images	47%
Collisions	53%
Comp. Aesthetics	55%
Video	56%
Pixel operations	57%
Sound	66%
Vectors	70%
3D	72%
Comp. Vision	82%

***Table 2**: The average order in which programming topics and graphic design topics were taught (%-values denote when the topic was taught relative to the entire course duration).*

Graphical User Interface (GUI) was discussed in 27% of the courses. A simple GUI provides students with a familiar way to explore the inherent aesthetic potential of a code without having to continually recompile or resort to arbitrary keyboard/mouse inputs. In courses that used Processing, the built-in "Tweak Mode" provided a rudimentary GUI within the IDE itself, allowing students to experiment with different values and receive immediate visual feedback. However, this option was only mentioned explicitly in two courses. GUI was more frequently discussed in courses that used p5.js, likely because a range of basic interface elements are readily available within the browser DOM.

Math, rarely incorporated in graphic design curricula, is an essential component of Creative Coding. Of the courses, 63% introduced basic algebraic, trigonometric, and geometric principles. Often, math was introduced using an apologetic tone, building on the assumption that graphic designers lack numeracy skills. One course featured a scheduled "Math Day!", with an exclamation mark to indicate caution or danger. This discourse may inadvertently have reinforced the students' pre-conceived notions that programming is hard to learn.

4. Discussion

Several researchers argue that the principles of coding share conceptual aspects with the principles of design [Tob12b; You01]. Despite their commonalities, notable differences were observed in how Creative Coding courses were structured and what content they included depending on the course instructor's disciplinary background. To elaborate on this, the generalized opposing notions of *code-first approach* versus *design-first approach* are introduced.

A code-first approach refers to programming educators who plan a Creative Coding course thinking, "how can I make graphic designers understand what programming is?" This approach forces graphic design topics to adapt and submit to programming paradigms. Typically, students learn how to convert well-known graphic design methods into the medium of code. Assignments are primarily given to test the students' proficiency at programming and refrain from assessing the aesthetic aspects of the students' works. Little attention is given to connecting the activity of programming with the students' field of study. This implies formal rigor and adherence to the established programming practices.

A design-first approach refers to design educators who plan a Creative Coding course thinking, "how can I make graphic designers use programming in their work?" This approach employs programming to explore graphic design topics computationally. Typically, students learn how to expand the boundaries of their discipline through the medium of code. Assignments are primarily given to test the students' ability to arrive at new visual expressions and refrain from assessing the quality of their code. Great attention is given to connect the activity of programming to the students' field of study. This implies exploratory discovery and a casual treatment of code.

4.1 Structure

The majority of the analyzed courses exhibited evidence of having been planned utilizing a code-first approach. Programming terminology was often favored over equivalent graphic design terminology (e.g. "loop" instead of "repetition," "output window" instead of "canvas"). Assignments focused on testing if students had understood a given programming topic and downplayed the assessment of their aesthetic quality. A consequence of structuring the course and teaching materials using a code-first approach is that students fail to utilize their existing, domain-specific, graphic design knowledge as a basis for constructing and acquiring new knowledge in a programming domain that is unfamiliar to them, which is an essential premise in constructionist learning theory [Pap87]. For example, graphic design students can use their existing knowledge of two-dimensional grids to leverage their understanding of the abstract concept of "nested for loops," a strategy specifically employed in five of the analyzed courses.

Another example of how the two approaches affected the syllabi, respectively, can be given by looking at how the topic of color was taught. Most of the analyzed courses used a code-first approach by taking the language reference of the chosen programming environment as an offset to discuss specific functions used to define and manipulate colors, thereby, leaving students to explore colors computationally within the confinements of the programming environment. Had a design-first approach been used, color theory, which has been established over centuries, could be used as a reference to discuss how colors can be defined and used computationally. Aside from replicating certain mathematical principles used to create harmonious color schemes, courses might also discuss new techniques that have become available through computation, e.g., creating palettes by sampling pixel values from an image, pixel-sorting, computing dominant colors, and connecting to APIs like COLOURLovers [Col18].

A few of the analyzed courses had been planned utilizing a design-first approach. One example was the course "Printing Code" [Mad16], taught at ITP by Rune Madsen. In his course, Madsen constructed a syllabus that stayed deeply rooted in graphic design and introduced programming topics only as they were required to illustrate, extend, and explore a particular graphic design principle. Although an advanced course assuming prior programming knowledge and, thus, excluded from the analysis, another noteworthy course was "Computational Form" [Bak18], taught at the Parson School of Design by Justin Bakse. Through highly visual and interactive course materials, adapted to cater to the needs of design students, this course established an exploratory environment where programming was taught with the clear intention of empowering Art and Design students to investigate new modes and forms of expressions as well as where programming topics were chosen for their ability to produce aesthetical output, rather than their canonical value within Computer Science.

Studies [DG06; Guz10] suggest that contextualizing programming into a setting more familiar to the audience positively affects student retention and motivation; thereby, research further prompts educators to use a design-first approach when planning Creative Coding courses intended for graphic designers.

4.2 Content

All courses dealt mainly with foundational graphic design topics, e.g., color, shapes, and typography. This is hardly surprising, as these are considered to be the basic components of the graphic design trade and, as such, would be expected to appear in an introductory level course. Absent in most courses, however, were topics and algorithms of particular value to graphic design as a spatial practice (e.g., object distribution, space filling, space partitioning, and overlap detection). While arguably more complex to implement and understand, it is pivotal to include these in a Creative Coding syllabus, as they can address and provide solutions to well-known issues experienced by graphic design students in their daily work.

Few courses investigated algorithms that produce a visual output that is only achievable through computation (e.g., glitch art, ASCII art, cellular automata, emergence, L-systems, fractals, self-organizing systems, evolutionary design, and drawing using data feeds). An example of one such course was "Computer Graphics con p5.js" [Bel17] at the Brera Academy of Fine Arts, taught by Prof. Antonio Belluscio. This course discussed topics like attractors, fractals, autonomous agents, and flocking behaviors, partially through presenting cases employing the technique and partially by providing simple code examples for students to explore at times. Conversely, courses that neglected to examine the potential of the computational aesthetic and its associated techniques taught students to use code to create works that originated in graphic design principles belonging in the pre-computer design era. This is counterproductive to the aim of educating graphic design students who can expand the boundaries of their discipline through the medium of code.

A final observation worth mentioning is that virtually all of the courses encouraged students to sketch their ideas on paper before performing any coding. Two of the courses even required the first exercises (involving harmonographs, automatons, and tiling patterns) to be solved using only pen, paper, and cardboard, thereby, using a familiar and "safe" medium to help students understand the principles involved in computational thinking [KP16; Win06]; this could potentially help disarm any premature aversion towards programming. However, as truly indigenous computational aesthetics are typically generated through computationally intensive calculations, they are virtually impossible to express manually in an analog sketch. To escape the inherent expressive limitations of physical materials, it is important that educators stress to their students that sketching solely using code is equally as important.

5. Implications & Future Research

Programming allows graphic designers to unlock and explore a new code-driven visual paradigm, but they must be inspired and given the necessary skills to do so in a way that builds upon and extends their pre-existing knowledge. This study indicates plenty of opportunities for educators to rethink and restructure how Creative Coding courses are currently taught in design schools. Considering the results obtained in this study, it is argued that educators must use a design-first approach when planning the structure and content of Creative Coding courses intended for graphic designers. A design-first approach is considered to be essential to effectively promote and embed programming as an established practice in graphic design education.

This study's data and the conclusions derived thereof are currently being used to develop a bespoke Creative Coding syllabus especially for use in design schools. Also underway is a study investigating the relationship between the students' motivations and the aesthetic quality of their assignments. Finally, dedicated research on the pedagogical and didactical strategies employed in the courses can further inform and encourage a dialogue among both programming and graphic design educators.

Acknowledgements

The author would like to thank all instructors whose courses formed the basis of the analysis. They all have put tremendous effort into creating their courses and have been kind enough to share them online.

References

[Ami11] AMIRI, F.: Programming as design: The role of programming in interactive media curriculum in art and design. *International Journal of Art and Design Education 30*, 2 (2011), pp. 200–210.

[Bak18] BAKSE, J.: Hello, Comp Form! *Comp Form* (2018). http://compform.net/.

[Bel17] BELLUSCIO, A.: Computer Graphics con p5.js. *Exframes* (2017). https://www.exframes.net/cg-p5js/.

[Col18] COLOURLovers: COLOURlovers API Documentation. *COLOURLovers* (2018). https://www.colourlovers.com/api.

[DG06] DORN, B., GUZDIAL, M.: Graphic designers who program as informal computer science learners. *Proceedings of the 2006 international workshop on Computing education research* (2006), pp. 127–134. (Proc. ICER '06).

[FR14] FRY, B., REAS, C.: *Processing: a programming handbook for visual designers and artists*. MIT Press, 2014. http://processing.org/

[Guz10] GUZDIAL, M.: Does Contextualized Computing Education Help? *ACM Inroads, 1*, 4 (2010), pp. 4–6.

[KP16] KNOCHEL, A. D., PATTON, R. M.: If Art Education Then Critical Digital Making: Computational Thinking and Creative Code. *Studies in Art Education 57*, 1 (2016), pp. 21–38

[Mad15] MADSEN, R.: On Meta-Design and Algorithmic Design Systems. *Rune Madsen* (2015). https://runemadsen.com/blog/on-meta-design-and-algorithmic-design-systems/.

[Mad16] MADSEN, R.: Programming Design Systems. *Programming Design Systems* (2016). http://printingcode.runemadsen.com/.

[MB13] MITCHELL, M. C., BOWN, O.: Towards a creativity support tool in processing. *Proceedings of the 25th Australian Computer-Human Interaction Conference on Augmentation, Application, Innovation, Collaboration* (2013), pp. 143–146. (Proc. OzCHI '13).

[MFR15] MCCARTHY L., FRY B., REAS C.: *Make: Getting Started with p5.js*. MakerMedia Inc., 2015. http://p5js.org/

[Pap87] PAPERT, S.: Constructionism: A New Opportunity for Elementary Science Education. *National Science Foundation NSF Award Search: Award #8751190* (1987).

[Pet12] PETTIWAY, K.: The New Media Programme: Computational thinking in Graphic Design Practice and Pedagogy. *Journal of the New Media Caucus 8*, 1 (2012).

[Sau13] SAUNDERS, S.: Coding as Craft: Evolving Standards in Graphic Design Teaching and Practice. *AIGA Design Educators Community* (2013).

[Tob12a] TOBER, B.: Making the Case for Code: Integrating Code-Based Technologies into Undergraduate Design Curricula. *Catch22: Eighth Annual UCDA Design Education Summit Abstracts & Proceedings* (2012), pp. 224–229.

[Tob12b] TOBER, B.: Creating with Code: Critical Thinking and Digital Foundations. *Mid-America College Art Association Conference* (2012).

[TF17] TZANKOVA, V., FILIMOWICZ, M.: Introduction: Pedagogies at the Intersection of Disciplines. In FILIMOWICZ, M., TZANKOVA, V. (eds.): *Teaching Computational Creativity*. 1st ed. Cambridge: Cambridge University Press, 2017, pp. 1–17.

[Win06] WING, J. M.: Computational Thinking. *Communications of the ACM 49*, 3 (2006).

[You01] YOUNG, D.: Why designers need to learn programming. In HELLER, S. (ed.): *Education of an e-designer*. New York, NY, USA: Allworth Press, 2001.