

PUBLIC CLASS GRAPHIC_DESIGN IMPLEMENTS CODE { // YES, BUT HOW? }

An investigation towards bespoke Creative Coding programming courses in graphic design education

A dissertation by Stig Møller Hansen · Student ID: au285478 · February 2019

Presented to:
Aarhus University
Faculty of Arts
School of Culture and Communication
Department of Digital Design and Information Studies
Supervisor: Martin Brynskov

Character count: 249.752 (104,1 standard pages of 2.400 characters)

ISBN: 978-87-7507-457-0
DOI: 10.7146/aulsps-e.340

"Art and Technology—A New Unity"
Walter Gropius, 1923

SUMMARY

Situated in the intersection of graphic design, computer science, and pedagogy, this dissertation investigates how programming is taught within graphic design education. The research adds to the understanding of the process, practice, and challenges associated with introducing an audience of visually inclined practitioners—who are often guided by instinct—to the formal and unforgiving world of syntax, algorithms, and logic. Motivating the research is a personal desire to contribute towards the development of bespoke contextualized syllabi specifically designed to accommodate how graphic designers learn, understand, and use programming as an integral skill in their vocational practice.

The initial literature review identifies a gap needing to be filled to increase both practical and theoretical knowledge within the interdisciplinary field of computational graphic design. This gap concerns a lack of solid, empirically based epistemological frameworks for teaching programming to non-programmers in a visual context, partly caused by a dichotomy in traditional pedagogical practices associated with teaching programming and graphic design, respectively. Based on this gap, the overarching research question posed in this dissertation is: *“How should programming ideally be taught to graphic designers to account for how they learn and how they intend to integrate programming into their vocational practice?”*

A mixed methods approach using both quantitative and qualitative analyses is taken to answer the research questions. The three papers comprising the dissertation are all built on individual hypotheses that are subsequently used to define three specific research questions.

Paper 1 performs a quantitative mapping of contemporary, introductory programming courses taught in design schools to establish a broader understanding of their structure and content. The paper concludes that most courses are planned to favor programming concepts rather than graphic design concepts. The paper’s finding can serve as a point of departure for a critical discussion among researchers and educators regarding the integration of programming in graphic design education.

Paper 2 quantitatively assesses how the learning style profile of graphic design students compares with that of students in technical disciplines. The paper identifies a number of significant differences that call for a variety of pedagogic and didactic strategies to be employed by educators to effectively teach programming to graphic designers. Based on the results, specific recommendations are given.

Paper 3 proposes a hands-on, experiential pedagogic method specifically designed to introduce graphic design students to programming. The method relies on pre-existing commercial graphic design specimens to contextualize programming into a domain familiar to graphic designers. The method was tested on the target audience and observations on its use are reported. Qualitative evaluation of student feedback suggests the method is effective and well-received.

Additionally, twenty-four heuristics that elaborate and extend the paper’s findings by interweaving other relevant and influential sources encountered during the research project are provided.

Together, the literature review, the three papers, and the heuristics provide comprehensive and valuable theoretical and practical insights to both researchers and educators, regarding key aspects related to introducing programming as a creative practice in graphic design education.

RESUMÉ

Denne afhandling er placeret i krydsfeltet mellem grafisk design, programmering og pædagogik. Den undersøger, hvordan der undervises i programmering på grafiske designuddannelser. Afhandlingen bidrager til forståelsen af de processer, praksisser og udfordringer der er forbundet med at introducere et publikum af visuelt orienterede praktikere – som ofte er styret af instinkt og mavefornemmelser - til en formel verden styret af syntaks, algoritmer og logik. Afhandlingen er motiveret af et personligt ønske om at bidrage til udviklingen af skræddersyede kontekstualiserede programmeringskurser, der er specielt designet til at imødekomme, hvordan grafiske designere lærer, forstår og bruger programmering som en integreret færdighed i deres erhvervmæssige praksis.

Den indledende litteraturoversigt identificerer en mangel på både praktisk og teoretisk viden inden for det tværfaglige område af programmeringsdrevet grafisk design. Særligt mangler der solide empirisk baserede epistemologiske rammer for undervisning i programmering til ikke-programmører i en visuel kontekst. Ydermere mangler der viden om, hvordan dikotomien i pædagogisk praksis forbundet med undervisning i henholdsvis programmering og grafisk design kan håndteres. Baseret på disse mangler er afhandlingens overordnede forskningsspørgsmål: *"Hvordan skal grafiske designere ideelt set undervises i programmering så der tages højde for, hvordan de lærer, og hvordan de har til hensigt at integrere programmeringen i deres faglige praksis?"*

Der anvendes en Mixed Method tilgang til at besvare forskningsspørgsmål gennem kvantitative og kvalitative analyser. Afhandlingens tre artikler er alle bygget på individuelle hypoteser, som efterfølgende bruges til at definere tre separate underforskningsspørgsmål.

Artikel 1 beskriver en kvantitativ kortlægning af nutidige introducerende programmeringskurser fra designskoler, for at skabe en bredere forståelse for deres struktur og indhold. Artiklen konkluderer, at de fleste kurser er planlagt så de favoriserer programmeringskoncepter frem for grafiske designkoncepter. Artiklens resultater kan tjene som udgangspunkt for en kritisk diskussion blandt forskere og lærere om integration af programmering i grafisk designuddannelse.

Artikel 2 vurderer kvantitativt grafiske designstuderendes læringsstilprofil sammenlignet med læringsstilsprofilen for studerende i mere teknisk orienterede discipliner. I artiklen identificeres en række væsentlige forskelle, der kræver fordrer brugen af anderledes pædagogiske og didaktiske strategier for effektivt at kunne undervise grafiske designere i programmering. Baseret på resultaterne gives en række specifikke anbefalinger.

Artikel 3 foreslår en praktisk erfaringsbaseret pædagogisk metode, specielt designet til at introducere grafiske designstuderende til programmering. Metoden anvender allerede eksisterende kommercielle grafiske designprodukter for at kontekstualisere programmering til et domæne, der er kendt for grafiske designere. Metoden er afprøvet på målgruppen og observationer omkring dens anvendelse rapporteres. Kvalitativ evaluering af feedback fra studerende tyder på, at metoden er effektiv og godt modtaget.

Derudover indeholder afhandlingen 24 heuristikker, som uddyber og udvider undersøgelsens resultater ved at inddrage andre relevante og indflydelsesrige kilder fra forskningsprojektet.

Tilsammen giver litteraturoversigten, de tre artikler samt heuristikkerne omfattende og værdifulde teoretiske og praktiske indsigter til både forskere og undervisere om centrale aspekter i forbindelse med introduktion af programmering som en kreativ praksis på grafiske designuddannelser.

ACKNOWLEDGMENTS

Many people provided support and guidance through the development of this dissertation, and in the following, I would like to express my gratitude:

I would like to thank my supervisor, Martin Brynskov, for his cooperativeness, indomitable enthusiasm, and genuine interest in my work.

I am grateful that Maria Hellström Reimer kindly allowed me to participate in the Swedish Faculty for Design Research and Research Education and thank her for her help in arranging my residency at the School of Arts and Communication (K3) at Malmö University.

I would also like to thank many of my colleagues at The Danish School for Media and Journalism. Thank you, Karen-Margrethe Österlin, for encouraging me to submit my application when the call for PhDs was made. I send a collective thanks to my colleagues Eng Agger, Anne Danger Boisen, and Karsten Vestergaard, who readily took my tasks upon themselves to let me focus on my study—I OU big time. I extend thanks to Anne Mette Møller Hartelius for inviting me into her Creative Coding courses; Thomas Rasmussen and Vibeke Borberg for their help in managing the many formalities involved in a cross-institutional cooperation; my PhD peers at DMJX, Maria Eitzinger, Jørn Ullits Olai Nielsen, Troels Østergaard, and Stig Brostrøm, for their support and openness in our meetings—I have taken comfort in knowing that I had you to share my experiences with. I would also like to thank my students at DMJX; without their openness, willingness to participate in my experiments, and continuous feedback, this dissertation would not exist.

Thanks to all external graphic design and computer science educators and researchers for openly sharing their thoughts, comments, ideas, teaching materials, and hard-gained experiences. I would also like to thank my blind-peer reviewers, whoever they are, for their constructive, meticulous, and well-intentioned feedback.

I would like to thank my mom and dad for their constant support and encouragement in letting me pursue my interests in visual design and computers—little did you know what impact that Amiga 500 would have on my life.

Lastly, I owe a heartfelt thanks to my wonderful wife, Anne Gramtorp, for her passionate and invaluable support during my four years as a PhD student: reading drafts, providing insightful comments, letting me vent frustration, sharing my enthusiasm when on a roll, lending her ears when things were tough, and reminding me to take care of myself. To my beloved kids, August and Bertil: Yes, Dad has finally finished writing his dissertation—time to bring out the LEGO, crayons, and paper planes!

A few notes on the text: This dissertation is written in American English. Quotations from British sources have not been changed. Throughout the dissertation, the terms “teacher,” “educator,” and “instructor” are used interchangeably, depending on the context. A glossary is provided for readers unfamiliar with some of the scientific and technical terms used within this dissertation. No gender politics are intended in the text, so if you prefer, please think “she” whenever you read “he.”

CONTENTS

Summary.....	5
Resumé.....	7
Acknowledgments.....	9
Chapter 1: Introduction.....	13
1.1 Overview.....	13
1.2 Background.....	13
1.3 Motivation.....	14
1.4 Subject field and position.....	14
1.5 Research gap.....	15
1.6 Thesis statement.....	16
1.7 Research questions.....	16
1.8 Dissertation structure.....	16
1.9 Contributions.....	17
Chapter 2: Literature review.....	19
2.1 Introduction.....	19
2.2 Graphic design.....	19
2.3 Programming.....	27
2.4 Pedagogy.....	33
Chapter 3: Methodology.....	41
3.1 Introduction.....	41
3.2 Mitigating viewpoint ambiguity.....	41
3.3 Paradigmatic position.....	42
3.4 Type of research.....	44
3.5 Research design.....	50
3.6 Order of execution.....	53
3.7 Validating results.....	53
Chapter 4: Research Contributions.....	55
4.1 Overview.....	55
4.2 Linking the papers.....	55
Chapter 5: Paper 1.....	57
Chapter 6: Paper 2.....	67
Chapter 7: Paper 3.....	75
Chapter 8: Conclusions.....	87
8.1 Introduction.....	87
8.2 Answering research questions.....	87
8.3 Theoretical implications.....	91
8.4 Practical implications.....	91
8.5 Limitations.....	91
8.6 Future Research.....	93
Chapter 9: Heuristics.....	95
Final Remarks.....	107
Glossary.....	109
Figures and tables.....	111
References.....	113

CHAPTER 1: INTRODUCTION

In this chapter, I provide an overview of the study and describe my personal background to explain how I approach this study. I also describe the broader motivation behind the study, state the main thesis, and describe the derived research questions. Lastly, I frame my research field and describe my own position therein.

1.1 Overview

This research project is a study into how programming can be taught to graphic designers with the aim of extending their skill set by the means of computation. The research will focus on pedagogical, technical, and aesthetical issues as seen from the perspective of a graphic design educator trying to establish a context for and an understanding of computation within the framework of creative, visual practice. This is not a historical or cultural study of code, but a consideration of the complexities that arise when teaching formal, word-based logic and numeracy skills to informal, visual, and intuitive graphic design students. The intended audience for the research is graphic design educators who teach, or wish to teach, computationally driven graphic design, a subset within the popularized term *Creative Coding*, defined by Mitchell & Bown (2013, 143) as “*a discovery-based process consisting of exploration, iteration, and reflection, using code as a primary medium, towards a media artefact designed for an artistic context.*”

1.2 Background

My background is in graphic design and interactive design. My research has emerged from a personal and professional interest in the use of code as a way to craft visual expressions. Instigated in the early 1990s by my teenage experiences with programming languages such as AMOS and AmigaBASIC, I became fascinated by the creative potential embedded in computers. During the early 2000s, I built a professional career using programs such as Director and Flash to produce interactive multimedia applications. When offered a position as educator at a design school 2007, I drew on my past experience to argue that coding should be part of the students’ curriculum. By then, technological advances, low computing costs, and a rise in code-based tools aimed specifically at visual designers had given programming a renaissance, making my long-running love affair with code and design *en vogue* and highly sought after by the industry. My approach to this research project, therefore, is that of a formally trained graphic designer and auto-didact programmer who teaches coding in a visual context to graphic design students in a Danish university college setting.

1.3 Motivation

In today's techno-centric, software-driven society, code plays an increasingly important role in our lives (Manovich 2013; Rushkoff 2010). Recently, there has been a massive focus on promoting code literacy, ranging from small-scale private initiatives, cultural grassroots movements, mainstream media coverage, and educational activities by institutions and organizations, to political legislation.

In the decades following the desktop revolution of the 1970s, programming as an artistic practice was mainly exercised by small communities of autodidactic computational artists (Reichardt & Institute of Contemporary Arts 1969), self-proclaimed "hackers" (Florin 1985; Levy 2010), and avantgarde demosceners (Majoros, Iván, & Matusik 2012; Carlsson 2009) who—in the style of true craftsmen—stretched their hardware beyond its limits in extraordinary visual and audible productions. The early 2000s saw programming move from a geeky subculture into mainstream media as an audience of young creatives began to explore the potential of code as an artistic, expressive medium (Manovich 2005). This renaissance of programming sparked a surge of open-source software and user-contributed tutorials dedicated to making expressive output using code. Today, it has never been easier for creatives to harness and utilize programming in their quest for novel expressions. However, I have often wondered why programming has never caught on within the graphic design community. Architecture, a closely related and also highly visuospatial domain, has long adopted programming as a way to explore new ways of constructing buildings through parametric, generative, and procedural computational methods (Cannaerts 2016). I can only speculate about why graphic designers are lagging behind in their adoption of computationally assisted formation, but the fact of the matter is that, while it is on the rise, only a few professional graphic designers use code as an integral part of their workflow and products (Shim 2016a).

A valid question to ask at this point would be: Should designers code at all? This is a highly debated question, and practitioners, educators, and scholars have many different views. Some argue that graphic designers should stick to their primary vocation and leave programming to the programmers (Cooper 2017). Some argue that graphic designers' encounter with code is too conceptual—because of the way they were taught higher order computational thinking skills—to be applied in their existing domain-specific workflow (Panda 2016). Others argue that graphic designers should be able to code their own programs as part of their ideation phase without having to enlist the help of a trained programmer (Stinson 2017; Kolko 2012a).

I subscribe to the final viewpoint. In my Creative Coding courses, it has never been my mission to make graphic designers fully-fledged programmers. That, I know, is never going to happen—nor should it. By teaching graphic designers how to create visual output through the medium of code, I hope to instill in them an understanding of how programming can be a highly versatile and useful addendum to their skill set, not only as a practical tool, but also, in a meta-cognitive way, informing how they think, plan, and execute visual communication. In my view, educating code-literate graphic designers is essential for the continued development of graphic design as a discipline.

1.4 Subject field and position

This study concerns topics that exist in the intersection of graphic design as a vocational discipline, programming as a theoretical and applied skill, and pedagogy as an instructional method to facilitate knowledge transfer. Schematically, this can be visualized as shown in figure 1.1, with this dissertation's subject field located in the middle where the three research fields overlap and are highlighted in black. Also shown in figure 1.1 is the my initial position approaching this study. As described in section 1.2, I have a professional background in graphic design. Graphic design is the

field in which I have accumulated theoretical and applied professional expertise over the span of two decades. Graphic design is also the central axis around which all inquiries in this study are made. Finally, graphic design is the field to which I aim to contribute. This is not to say that pedagogy or programming are less important topics. Together, they form the trinity that constitutes my subject field. As described in chapter 2, some scholars and practitioners have made contributions to my subject field; however, many of them have done so with a personal background in either pedagogy or computer science. My personal profile allows me to evaluate the existing body of knowledge in my subject field through the lens of a pragmatic graphic design practitioner and educator.

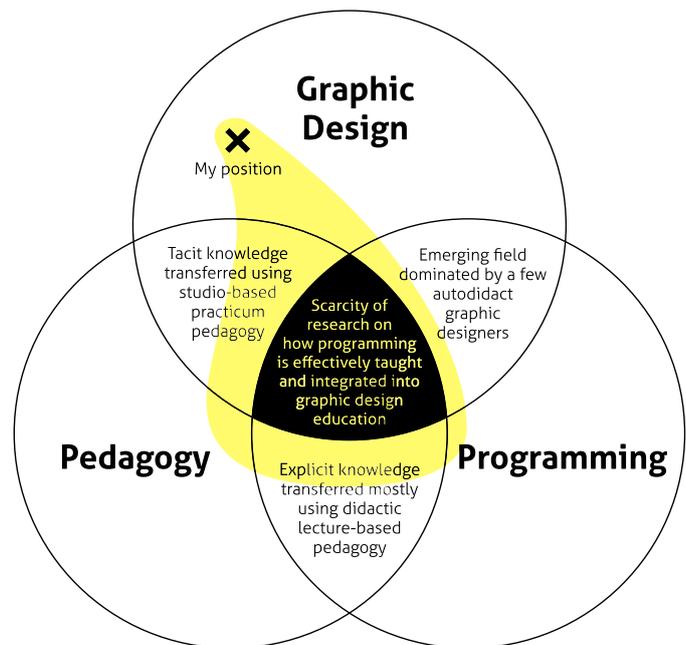


Figure 1.1: The subject field of this dissertation (marked in black), position of the author (marked by X) and the research fields covered in this study (highlighted in yellow).

1.5 Research gap

Technology has created unprecedented possibilities for designers to engage in digital media (Maeda 2004; Manovich 2013). Finding a way to manage the increasing complexity of technology within the design curriculum is key to keeping design education relevant (Fleischmann 2013). While there are other areas of research that look at associations between creative practices and computation (e.g., generative art, software art, digital craftsmanship, software education, digital design education, and media computation) there is a lack of documented investigations into how expressive programming, or Creative Coding, should be adapted and taught specifically to graphic designers to accommodate their needs emerging from the *mode d'emploi* of their vocational practice. While practice-based education and studio teaching have been studied philosophically or ethnographically (Schön 1983; Cross 1982), the tacit and discursive learning essential to Creative Coding has yet to be comprehensively discussed in the light of reflection on current pedagogies (Tzankova & Filimowicz 2017, 1). Finding texts that address pedagogy, curriculum, and educators' professional development in the richly diverse field of Creative Coding is challenging because teaching and learning are considered to be marginal to the prevailing discourses (Tzankova & Filimowicz 2017, 2). Overall, there is a scarcity of works that deal with the pedagogies of computational media and design from practical and interdisciplinary perspectives (Tzankova & Filimowicz 2017, 2).

This research aims to contribute by filling this gap, providing both practical guidelines as well as a starting point for the discussion of how programming can be successfully integrated into graphic design curricula.

1.6 Thesis statement

To frame and guide my research, as well as state my initial position in relation to my investigation, I have formulated the following thesis statement:

Contemporary Creative Coding courses teach programming as an artistic practice informed and driven by technical affordances of the programming environments, leaving graphic designers without a bespoke contextualized syllabus designed specifically to accommodate how graphic designers learn, understand, and use programming as an integral skill in their vocational practice.

1.7 Research questions

Based on my thesis statement, this study asks as its broad and overarching research question (RQ):

RQ: How should programming ideally be taught to graphic designers to account for how they learn and how they intend to integrate programming into their vocational practice?

Subdividing this question into its core constituents allows me to pose three specific research questions (SRQs) which will be investigated in separate studies:

SRQ1: How is Creative Coding currently taught in graphic design education?

SRQ2: How should Creative Coding be taught to accommodate how graphic design students learn?

SRQ3: How can graphic design students be motivated and supported as they are introduced to programming?

The first specific research question aims to provide a snapshot of the current landscape of Creative Coding courses, which establishes a foundation to inform the discussion on issues related to pedagogic strategies and course content.

The second specific research question aims to investigate the ways in which graphic designers learn, specifically focusing on how students use their existing domain-specific knowledge and cognitive models (graphic design) to leverage knowledge and skill acquisition from another domain (programming).

The third specific research question aims to investigate the consequences of contextualizing Creative Coding assignments as a way to heighten students' motivation and improve their attitude towards learning to program.

Each of these are addressed individually in my papers (chapters 5–7), and are answered and discussed together in chapter 8.

1.8 Dissertation structure

This dissertation is divided into nine chapters, each with its own specific objective:

Chapter 1 provides an overview of the study. It describes my personal background to aid in understanding how I approach this study and explains the broader motivation behind the study. It presents the main thesis and describes the derived research questions. Lastly, it frames my research field and describes my own position therein.

Chapter 2 reviews the literature relevant to my subject field and discusses current knowledge, substantive findings, and contributions in areas where graphic design, programming, and pedagogy intersect and overlap.

Chapter 3 describes my paradigmatic position and explains its influence on this study. It then accounts for the research design used to answer the three specific research questions. Lastly, it suggests means for validating my results.

Chapter 4 provides an overview of my research papers and discusses their interrelationship.

Chapters 5–7 each present a research paper along with a submission history and publication state.

Chapter 8 revisits the research questions, answering each of them individually. It discusses both theoretical and practical implications of my research and considers its limitations. Finally, it suggests future research to be conducted.

Chapter 9 aggregates the cumulative knowledge acquired during my study and distills my findings in a list of pragmatic and applicable heuristics.

The paper concludes with a list of figures, tables, and references along with a glossary to explain the scientific and technical terms used within this dissertation.

1.9 Contributions

This dissertation makes five major contributions:

- It provides a comprehensive snapshot of the current structure and content of Creative Coding courses. This knowledge can offer a point of departure for discussion and inform the debate among design educators about how best to incorporate programming in graphic design curriculum.
- It presents two approaches to planning Creative Coding courses: *code-first* versus *design-first*. These perspectives are useful both in discussions among educators and for individual educators as means to reflect on how they plan their courses.
- It gives insight into the learning style profile of graphic design students, specifically related to learning programming. This knowledge can be used by design educators to tailor their teaching material to account for how graphic design students learn.
- It suggests a pedagogic method: *deconstruction/reconstruction*. This method can be used as is by design educators who teach introductory programming to graphic designers.
- It offers a list of heuristics containing pragmatic and applicable guidelines and recommendations for design educators who seek to improve their Creative Coding course.

If any of these contributions manages to find its way into Creative Coding classrooms in design schools, then my work has not been in vain.

CHAPTER 2: LITERATURE REVIEW

In this chapter, I review the literature relevant to my subject field. I discuss current knowledge, substantive findings, and contributions in the areas in which graphic design, programming, and pedagogy intersect and overlap.

2.1 Introduction

To facilitate an understanding of the relevance, importance, and challenges associated with teaching programming in graphic design education, I present a survey of the current literature relevant to my research field. This helps establish the solid foundation that has informed my research design and methodology.

As explained in section 1.4, this study situates itself in the area in which graphic design, programming, and pedagogy overlap and intersect. These three independent and well-established research areas each comprise a vast body of knowledge; consequently, this literature review does not intend to provide an in-depth description of the background, prevailing theories, and methods of each of the three. Instead, it investigates and interweaves findings from each to offer a synergetic and holistic framework for answering the research questions.

Methodologically, this literature review is not systematic in the sense of applying a set of defining keywords to particular databases. While such an approach might theoretically improve my study, I doubt whether it would yield better results in practice due to the varied terminology that is used in discussions across my three research areas. Given the objectives of my study, I conducted the review in an organic fashion, relying on ad-hoc searches, recursive examination of citations in key references, and suggestions from fellow scholars.

In section 2.1, I review literature situated in the field of graphic design that discusses the role, use, and implementation of programming in the graphic design industry and graphic design education.

In section 2.2, I review literature situated in the field of computer science that discusses programming applied in visual creative practices and research into how programming can be taught as an informal skill to non-computer science students.

In section 2.3, I review literature situated in the field of pedagogy that discusses general pedagogic and didactic strategies suitable for supporting students' acquisition of knowledge in creative, technology-driven learning environments.

2.2 Graphic design

2.2.1 What is graphic design?

Graphic design is a relatively young and difficult-to-define discipline in the context of academic research (Hannaford 2012). The term *graphic design* was first coined by Dwiggin in 1922 (Dwiggin 1922; Meggs & Purvis 2006) but did not gain widespread use until the 1960s. Before then, graphic

design was known by the decidedly unacademic term of *commercial art* (Hannaford 2012). Today, graphic design—also known as communication design—is defined by the industry’s leading organization, the American Institute of Graphic Arts (AIGA), as “*the art and practice of planning and projecting ideas and experiences with visual and textual content. The form it takes can be physical or virtual and can include images, words, or graphics.*” (AIGA 2017). It encompasses a large and diverse range of media and forms of communication; for example, visual identities, posters, advertisements, packaging, book design, newspaper design, wayfinding, illustration, information graphics, data visualizations, motion graphics, interactive graphics, web design, and app design (AIGA 2017).

Historically, graphic design was a manual trade learned by apprenticeship and carried out by hand in a physical world using specialized analog and mechanical equipment (Jury 2012; Levit 2016). Throughout history, designers have always implemented systems and logic in their work. Examples include Albrecht Dürer’s *Underweysung der Messung mit dem Zirckel und Richtscheyt* (1525), the *Architype* series by Doesburg and Albers (1920s), *New Alphabet* by Crouwel (1967), Gerstner’s *Programme entwerfen* (1964) and *Kompendium für Alphabeten: Systematik der Schrift* (1972), Müller-Brockmann’s *Raster systeme für die visuelle Gestaltung* (1981), and *Programmiertes Gestalten* (1980) by Kapitzki. Despite their highly systemic nature, these examples were all manually executed. However, in the 1970s, technological advances in production techniques provided graphic designers with access to sophisticated electronic systems capable of automating the tedious and time-consuming tasks of the past. Pioneers began to explore the potential of digital technology in graphic design production, with Knuth’s *TEX and METAFONT* (1979) leading an inquiry regarding the impact of automation in graphic designers’ systems (Shim 2016a).

2.2.2 The advent of computers in graphic design

Early use of computers by graphic designers dates back to the mid-1960s (Faison 1995, 145), but the major breakthrough of computers in the graphic design industry occurred two decades later in the 1980s (Faison 1995; Maeda 2002; Dubberly 1990; King 1988; Meggs & Purvis 2006) and heralded a paradigm shift from analog to digital production. Optimistic graphic designers praised the computer as a major revolution in graphic arts (Heller 2002; Dubberly 1990; Maeda 1999). Less enthusiastic graphic designers simply embraced it as another tool on their workbench (Blauvelt 2011, 23), unaware of how it would soon eliminate the work of production artists, photomechanical technicians, keyliners, paste-up artists, typesetters, color separators, and printers (Blauvelt 2011, 23). Skeptical graphic designers, however, feared, rejected, and decried digital technology during its infancy (Faison 1995) and called designers who did explore it “the new primitives” (Meggs & Purvis 2006, 490). They feared it would detract from creativity and depersonalize the work. As acclaimed graphic designer Paul Rand once famously remarked: “*They [computers] are just like pencils; nothing special.*”

Known as “The Desktop Publishing Revolution” (Tucker 1988), computers quickly rendered previous physical techniques obsolete and indicated a shift toward software as the dominant tool for graphic design production, thereby simultaneously altering both the process and aesthetics of graphic design (Richardson 2010, 46; Heller & Womack 2007, 17). This transition of epistemology in the graphic design process saw tangible materials give way to virtual *metamaterials* (Manovich 2013) with properties and attributes that escaped any physical limitations. As designers began to explore these metamaterials, new creative possibilities emerged. An example thereof is *Beowulf* (1990) by Blokland & Rossum, which used consciously manipulated PostScript code to create a dynamic, ever-changing font.

In the mid-1980s, most graphic designers did not possess the necessary technical skills required to explore the affordances of the digital medium. However, another group of people did. Considered

a sub-culture back then, hackers, nerds, and *demosceners* (Florin 1985; Carlsson 2009) saw computers as creative machines and challenged themselves to push the limited hardware to its boundaries. Using code as their material, they created digital artifacts with a previously unseen and distinct digital aesthetic that would soon make its way into popular culture and play a key role in defining the aesthetics of the decade. Foreseen by Dubberly (1990), and following the rise of the Internet in the mid-1990s, graphic designers began experimenting with HTML, Shockwave, Flash, and Java applets (Pearson 2011; Watz 2010). Maeda saw computers as a new material for expression (Maeda 1999, 101), and in 1999, in an effort to make digital technologies available to design students, he developed *Design By Numbers* (DBN) (Maeda 1999, 2004), a simplified environment to explore programming in a visual context. In 2001, Fry, Reas, & Maeda (2007) evolved DBN into a more versatile and extensible framework, changing its name to *Processing* in the process. DBN and Processing paved the way for a multitude of new programming platforms aimed at designers and artists (Lehni & Puckey 2011). By the turn of the millennium, programming had become trendy (Manovich 2005, 3), and tools developed for technical use were gaining cachet in the digital arts (Maeda 2002). As computers continued to evolve to form the core of graphic design processes, design tools such as code editors and prototyping applications adopted a mode of processing known as “if-then,” or conditional, logic (i.e., if this happens, then do that). For example, Blauvelt coined the term *if-then approach* (Blauvelt 2008) to define this particular design approach, and he argued that this process, mostly used by programmers, could also be beneficial to graphic designers.

Traditionally, graphic design had clear directions and a defined purpose (Davis 1998), but the advent of digital media fostered a range of new tools, skills, and disciplines (e.g., *interactive design*, *media computation*, and *interface design*), which, in turn, forced researchers, educators, and practitioners to engage in a fundamental renegotiation of what comprised graphic design as a discipline. However, amidst the turmoil caused by the paradigmatic changes, Meggs and Purvis (2006) argued that the essence of graphic design, namely that of conveying messages through visual means, remained unchanged.

2.2.3 The role of the post-computer graphic designer

Acknowledging the rapid shift towards digital production, graphic designers were forced to reflect on how they would exercise their practice and how it affected their role as creators (Richardson 2006). Following an investigation of how graphic design practitioners tackled the transition from analog to digital production, Faison (1995) suggested fourteen new possible roles for graphic designers. Interestingly, given the context of this dissertation, Faison did not discuss programming as an activity to be undertaken by graphic designers. Later, representing those skeptical of computers, Kelly (2002) admitted that computers were changing the definition and role of the graphic designer, but he believed the nature of that change was still unclear. Unlike Kelly, Maeda (2002) saw a clear distinction between the pre-computer designer and the post-computer designer. Maeda was confident that designers would come to appreciate the computer’s unique role in the future of arts and design (1999, 13), later arguing that “*any designer that has not adapted to the computer is either lying or out of work*” (Maeda 2002). As time progressed, it became evident that graphic design would increasingly rely on computers, made possible by cheap, accessible, and powerful hardware and software. Hard-learned skills that once took years to master became available to everyone—including non-designers—at the click of a mouse button. This democratization of graphic design practice was criticized by Fleischmann (2013, 7), who believed it would blur the boundaries between amateur and professional practice and promote amateurism. This view was shared by Tober (2017, 109), who maintained that graphic designers should capitalize on the possibilities of new forms of practice;

otherwise, non-designers would do so and create work that society recognized and accepted as design (2017, 109).

2.2.4 Graphic designers as users of software

The prevailing discourse regarding graphic designers' use of technology has tended to stigmatize them as users of software in the shape of applications provided to them by commercial entities. Maeda (1999, 19) strongly objected to pre-packaged software and argued that the computer industry was misleading people to mistake software skills for design skills. Several other scholars and practitioners also criticized the inherent boundaries of the commercial (Mittendorf 2000; Ward 2001; Blokland cited in Hoxsey 2003; Mateas 2005; Terzidis 2009). Watz (2003) expressed the view "*that designers have become dependent on software [...] forcing the designer to adapt her work to the decisions and metaphors chosen by the programmer.*" This was further problematized by Lehni (2008), who accused Adobe's monopoly in the graphic design software market to cause a lack diversity and alternatives. Lehni & Puckey (2011) observed that predominant software applications exerted a strong influence on the aesthetics of the products and that graphic designers rarely questioned their role.

2.2.5 Teaching software packages in graphic design education

Graphic design education has always taught the tools used in the industry. As analog tools became digital tools, design schools consequently began teaching software packages. Early on, critics raised concerns about the potentially negative impact of teaching software packages in graphic design curriculum (Maeda 1999, 2002; Kelly 2001). Continuing to teach software packages, Tober (2012b) warned, would only help to perpetuate the unfortunate belief that these applications were considered design. However, abandoning all commercial software has never been considered an option, as they offer many advantages. Instead, several scholars (Maeda 2002; Pettitway 2012; Tober 2017; Lehni & Puckey 2011; Watz 2003) have seen programming as a solution to relinquish more control over computer systems for graphic design practice. Establishing symbiotic relationships between industry-standard tools and programming through exploration of scripting capabilities would allow graphic designers to forge their own tools and use them in their existing workflow (Lehni & Puckey 2011). Also, by adding programming to the graphic design curriculum, to supplement—not replace—commercial software, design schools would fulfill Maeda's early vision of "*a future in which designers are free to author their own software [...] [making it] possible for designers to define the trends today rather than wait for the industry to define the terms of an evolving expression*" (Maeda 2002, 41). Digital design continues to move from software to programming as a new kind of practice (Richardson 2010). Technical proficiency is now to be measured in terms of a graphic designer's level of fluency in a variety of code-based technologies because mastery of various industry-standard software applications is now presumed of any designer (Tober 2012b).

2.2.6 Should graphic designers learn to code?

A fundamental and much debated topic is whether designers should learn to code. Early on, several scholars (Ursyn et al. 1997; Young 2001; Andersen et al. 2003) addressed the issue, all arriving at the conclusion that art students would benefit from acquiring basic programming skills. Maeda (2002) also argued that coding was an essential and valuable skill for graphic designers to learn and advocated that the best way to do so was by directly engaging with it. Weiman (2001) and Zee (2001), however, believed that a conceptual understanding of code was sufficient. Those who opposed code and digital technology in general were many pre-computer era design educators,

perhaps most famously Rand (cited in Kroeger 2013), who believed it to be the realm of engineers and computer scientists. Today, the debate goes on, with scholars and practitioners making arguments for and against why graphic designers should learn to code.

In favor teaching code to graphic designers is Fallman (2017) who believes that rudimentary coding skills are key for any graphic designer in the digital space. Maeda (2018) sees coding as a fundamental skill to be learned by what he refers to as a new breed of “computational designers.” Kolko (2012a) has suggested the metaphor of “code as material” and argues that in order for graphic designers to master it, they must experience it. Madsen (2015) has identified several reasons why designers should learn to code, including the ability to question assumptions brought about by existing design tools, and maintains that they should be able to build new tools to replace them. A final proponent for code in the graphic design curriculum is Shim (2016a, 2016b); however, he believes that instead of learning the syntax of code, focus should be placed on learning to build logic with code.

Arguing against coding as a skill to be learnt by graphic designers is Cooper (2017), who believes that code/design as a cross-disciplinary exercise serves no purpose. Instead, he argues for a strict distinction between disciplines. Also skeptical, but less dismissive, is Atwood (2012), who has posited that learning to code can be rationalized only if it helps one to perform his job better.

Accepting Atwood’s view and suggesting that coding *is* in fact helping graphic designers perform their job better is Tober (2012b), who claims that coding has become a competency with significant value for design in both professional practice and education. However, the most compelling argument in favor of teaching programming to graphic designers originates from students themselves. Inspired by the growing surge of graphic design products made entirely from using code, more graphic design students request courses that teach them how to incorporate code in their work. When students express an interest in exploring code, it would be unwise of design schools to deny them this. Given the possibilities today, students would simply satiate their appetite for code somewhere else, most likely without graphic design educators to help them contextualize and relate coding to graphic design.

Regardless of one’s position in the debate, the fact is that graphic designers are increasingly engaging in informal end-user programming activities, leading them to reconsider their own role by asking “are we designers or developers?” (Johansson 2007). This blurring of the lines of professional practice, caused by the convergence of graphic design and computer science (Reed & Davies 2006), has prompted many scholars and practitioners to contemplate the possible need for new terms such as “designoper,” “devigner,” “unicorns,” “designicorns,” “hybrid designer,” “computational designers,” or “meta-designers.” The muddled terminology and the ensuing inability to unambiguously describe their professional role illustrates the deficient self-understanding and identity crisis associated with being a designer who also codes.

2.2.7 Integrating programming in graphic design education

Following programming’s surge of popularity in the design community at the turn of the millennium (section 2.2.2), programming gradually gained a foothold in graphic design education as a creative practice. Over time, several scholars and educators have argued strongly in favor of programming to be included in the graphic design curriculum (Dubberly 1990; Maeda 2002; Pettitway 2012; Wasco 2008; Amiri 2011; Lehni & Puckey 2011; Tober 2017). Young (2001, 64) posited the argument that programming would enable designers to “*conceive new categories of solutions and provide the technical ability to realize them.*” Reas (quoted in Hoxsey 2003) argued that teaching designers programming would allow them to develop a deeper understanding of code and software, which, in

turn, would encourage a unique use of the computer medium. Similarly, Watz (2003), discussing computational design, regarded programming as a way to provide designers with a new literacy in digital media, which he believed to be mandatory to fully explore the possibilities of electronic media. Sharing this view was Pettway (2012), who posited that addressing the relationship between graphic design and programming was paramount to encouraging designers to push the boundaries of practice and theory. He has been joined in this view by Tober (2017), who has advocated a comprehensive integration of code both in the foundation of and throughout a design curriculum, and by Amiri (2011), who has warned that excluding programming from the graphic design curriculum will restrict the generation of creative ideas as well as the students' options in translating, expressing, and converting their ideas into artifacts.

Generally, the literature agrees that technology and programming should be introduced in graphic design education and considers it to be long overdue, causing design education to be largely stuck in the past and out of date, with only a few innovative institutions to spearhead initiatives toward integrating programming into the curriculum (Fleischmann 2013). Despite the many advocates, involving students in programming activities as part of their studio-based practice is still a rare occurrence (Knochel & Patton 2015), and explicit focus on the development of technical skills remains taboo in many design programs (Tober 2012b). If design students are fortunate enough to be exposed to programming at all, it is usually in the form of an ancillary elective course that serves only as a basic introduction (Tober 2012b). The reason for this, Pettway (2012) has conjectured, is that graphic design education is at large perplexed, misguided, and constantly challenged as it tries to maneuver programming into the curriculum and balance *"a proportion of technology instruction to problem-solving, visual studies, and theoretical issues."*

Introducing graphic design students to programming comes with its own set of challenges that must be circumvented by understanding and accounting for the domain-specific context. This requires adapting both instructional material and teaching style to fit the graphic designers' actual needs and their learning style preferences. As noted by Young as early as 2001, *"programming is not something that can be tacked on to an existing education"* (Young 2001, 64). Furthermore, it is important to acknowledge that not all graphic designers feel inclined or able to learn programming. Scott & Ursyn (2006) have claimed that students undertaking a design degree tend to be better in either design or IT. This creates a gap between designers who are literate in code and those who are not (Lehni & Puckey 2011), which has led Freyermuth (2016) to suggest that it must be left to students to decide how much coding they need.

The main challenge is defining a role for programming in the curriculum and striking a balance between technical and design skills (Amiri 2011). Amiri commented on this balance, asking that design educators *"find new ways of using and embedding technology in [the] curricula so that it is more in harmony with art and design culture and its traditional creative practices"* (2011, 208). Endorsing this pursuit are Freyermuth (2016) and Madsen (2016), who have simultaneously warned that the long-established fundamentals of the discipline should be maintained while understanding the changes and needs of the discipline as it evolves in a new era.

2.2.8 Keep it a design education, not a computer science education

Kelly (2002) has likened the advent of computers in graphic design to the dilemma faced by the Victorians: a sudden decrease of constraints with a corresponding increase in options because of new technology. Kelly has also cautioned graphic designers not to become seduced by the technology like the Victorians and disregard the *"language of graphic design"* (Poulin 2011) that has been established and refined over centuries. Having observed the ease by which intricate and complex

designs can be effortlessly created using computers, Kelly (2001, 152) argued that *"complexity should not be confused with quality"* and cautioned students to exercise restraint to avoid creating *"visual gibberish and a hodge-podge of elements."* Similarly, specifically addressing the medium of code, Madsen (2016) has asked his graphic design students to honor the legacy of the trade and refrain from *"smudging the canvas with a repetition algorithm"* and *"placing a bunch of stuff randomly across your canvas."* In his programming course, Madsen (2016) has interwoven traditional graphic design virtues with computational principles while maintaining a fundamental design perspective. Another programming course by Bakse (2018) has also stayed true to established design tradition while simultaneously exploring a range of new design approaches made possible only through computation. Heller & Womack (2007, 12) have claimed that technology has so thoroughly altered the way designers now practice that it is as necessary to be a technologist as it is to be an artist. In that respect, both Madsen and Bakse have backgrounds in formal design training and several years of programming experience, making them well-suited to teach programming to design students while maintaining a design perspective. While Heller & Womack have claimed that the increasing focus on technology in design education has tossed the traditional definitions of graphic design and beauty aside (2007, 17), Kelly, on the other hand, has argued that good judgment in making design decisions grows out of visual values or principles, which have not changed, only the technology that gives them form (Kelly 2001).

The literature widely agrees that including computation and programming in the graphic design curriculum is a prudent decision that is long overdue. As more design schools do so, it is important that they keep their programming courses anchored in and related to the graphic design domain (Amiri 2011, 207). Amiri (2011) has noticed a fundamental difference between teaching programming to computer science students and to design students and has likened the difference to that of teaching a foreign language to a linguist and to a tourist. Unlike courses in computing education, learning to program in graphic design education is not a goal in itself; rather, it serves as a means to achieve a higher purpose, namely that of crafting visual communication.

2.2.9 Design educators who can program

Successful integration of programming in the graphic design curriculum requires educators with a profound practical and theoretical experience in the field of graphic design; however, they must also possess an understanding of digital media and programming. Arguments for art and design educators to investigate computers and programming appear in the design education literature around the late 1980s (Ettinger 1988; Dubberly 1990; Hausman 1991), with many contributions to literature made since, most recently and notably by Tzankova & Filomowicz (2017). In 2002, Maeda described how *"design schools today employ an entire generation of disillusioned pre-computer design educators who feel increasingly irrelevant [and] post-computer design educators [who] are scrambling to stay current with tools and systems that are born to evolve on an hourly basis"* (2002, 40), confirming similar previous observations made by McCoy (1998, 11). As true now as it was in 2002, only a few art and design educators have demonstrated expertise in programming (Knochel & Patton 2015, 22). They have many other demands on their time, making it hard for them to muster the commitment, effort, and consistent hands-on practice writing code that is required to keep up with new programming trends, techniques, and languages (Freyermuth 2016). Nevertheless, the assimilation of computers in graphic design education is increasingly making coding literacy mandatory for design educators. Some design schools have sought to circumvent the shortage of code-literate design educators by hiring computer science educators to teach Creative Coding courses. This setup, however, has caused students to complain that their educators did not understand design (Pannafino 2013), effectively

highlighting the quintessential problem of outsourcing programming courses to non-design educators.

2.2.10 Impact of programming on the evolution of the graphic design discipline

The graphic design education literature largely agrees that programming is a natural addendum to the contemporary graphic design curriculum and that students should acquire at least a basic level of computational literacy by engaging in informal, hands-on programming. Hence, it seems appropriate to investigate how programming and computation is believed to affect the evolution of the graphic design discipline.

According to Terzidis (2009), the increased use, misuse, and abuse of computational design has raised concerns about the future direction that design may take. While some have regarded programming and computation as a misappropriation of what design should be, Terzidis considered it a liberation which would hopefully foster a *“new generation of truly code literate creative designers who can take fate into their own hands”* (Terzidis 2009, xx). Likewise, Madsen (2015) has envisioned new generations of what he refers to as *meta-designers*; designers working in the intersection between art, design, and computation, to whom programming is a natural tool. Similarly, Tober (2017) has regarded *meta-designing* as a shift back to a designer’s engagement with production, stating that *“meta-design involves the transformation of the role of the designer from one in which s/he is primarily concerned with the design of individual artifacts to one where s/he also creates or develops new tools, systems, and methods for design”* (2017, 96). Tober has also described the computational graphic design process as a *mega-process* that encompasses both *meta-process* and a *meta-meta-process* to describe the relationship between the designer, user, code, and visual output (2011, 12–14).

A recurring theme in the discussion of how programming will affect the graphic design trade is the making of custom design tools as an alternative and opposition to the industry-standard, commercial software packages (Womack & Lezni 2006; Richardson 2016). This can be seen as a digital re-emergence of the previous analog craftsmanship associated with the discipline (Richardson 2006, 2010).

Learning to program will also enable graphic designers to use algorithms to create flexible systems, a process that is concentrated on iterative formation with parameters instead of a fixed end form. According to Shim (2016a), this resonates with the inherent systemic nature of graphic design (see section 2.2.1) and presents a change in viewpoint from form to formation. However, compared to the “mechanical” and formalistic design systems of the pre-computer era, modern computational graphic design systems can include a wide range of sophisticated and advanced technologies and processes (e.g., artificial intelligence, neural networks, machine learning, autonomous generative systems) whose use raises new questions of authorship, ownership, originality, and creativity (Galanter 2009; McCormack et al. 2014).

Programming has already brought about a new, distinctly “computational” visual style—*New Aesthetic* (Bridle 2012)—which is struggling to mature and establish itself as a solid genre. Though not denying its rightful presence in modern graphic design, Sterling (2012a, 2012b) voiced a critique of *New Aesthetic* for being immature, prompting Watz et al. (2012) to respond by arguing that *New Aesthetic* was already an integrated part of society. In a recent evolution of the *New Aesthetic* genre, graphic designers are now using Web browsers as a tool for creating designs, with the inherent technical affordances directly influencing the visual aesthetic (Benoit 2017).

New breeds of code-savvy graphic designers will emerge too. In his sarcastic piece, “Everyone Hates Creative Coders,” Pearson (2013) pointed to the fact that the established business model has a hard time fitting *creative coders* into its existing practice and workflow. But the graphic design

industry is rapidly adapting, spurred on by the benefit to workflow, creativity, productivity, and products made possible through programming and computation. Indeed, as Maeda (2018) concludes, code-literate designers are in high demand.

2.3 Programming

2.3.1 Learning to program is difficult

Learning to program is notoriously difficult. Teaching programming to novices has been—and continues to be—a big challenge (Bennedson 2008). Repenning (2017), reflecting on twenty years of teaching programming, observed that many students consider programming to be “hard and boring,” a view contributing to frequent high dropout rates in introductory programming courses (Matthíasdóttir & Geirsson 2011). Still, programming is not innate but a learned skill that anyone can acquire and improve with practice (Brown & Wilson 2018). In fact, the belief that some people are born programmers and others are not has been referred to by Guzdial (2015) as “*computing’s most enduring and damaging myth*.” Repenning (2017) has argued that addressing the “hard” part is a cognitive challenge requiring programming to become more accessible, while addressing the “boring” part is an affective challenge requiring programming to become more exciting.

Traditionally, computing education has tended to favor formal learning environments (Dorn & Guzdial 2006). This approach, according to Vihavainen, Paksula, & Luukkainen (2011), assumes most introductory programming courses to be taught using lectures structured according to language constructs, take-home assignments, and perhaps also demo-sessions. Over time, several instructional strategies (e.g., problem-based learning, minimalist instruction, extreme apprenticeship, pair programming) have been employed and tested in attempts to improve computer science students’ learning outcomes and course retention. The pedagogy and instructional strategies of computing education are discussed in more detail in section 2.4. Despite the many approaches taken, the literature suggests that there is still no consensus on how programming is ideally taught. Moreover, because computing as a general topic is no longer exclusive to the domain of computer science (Amiri 2011), instructional strategies for teaching programming must be developed within the individual disciplines to account for the disciplinary context, typical use cases, and the learning style profile of the target audience.

2.3.2 New breeds of programmers require new pedagogical approaches

Despite Andersen et al. (2003) having demonstrated a need to modify the traditional computer science method, computer science educators have largely been hesitant to modify their pedagogical approaches, perhaps too fettered by the mathematical and engineering legacy of the discipline (Greenberg, Kumar, & Xu 2012). However, over the course of the past decade, computer science has increasingly adopted pedagogical models and instructional strategies developed for teaching programming in design schools in an attempt to make introductory programming courses more engaging (Xu, Wolz, & Greenberg 2018). This collaboration between computer science and graphic design educators was considered imperative by Reed & Davis (2006, 186) to ensure that each discipline learned from the other and was prepared for future developments. Amiri (2011) has argued further that teaching programming to graphic designers must abandon the engineering model of software construction in favor of approaches that recognize the unique characteristics of digital design and the malleable nature of interactive artifacts. In this respect, it can be helpful to consider the activity of programming in relation to traditional artistic activities such as writing or painting.

2.3.3 Programming as “sketching,” “sculpting,” “bricolaging,” and “hacking”

Over time, several computer science and design scholars have discussed programming using vocabularies, perspectives, and metaphors originating from artistic practice.

Blum (1996) introduced a “sculpting” metaphor, in which programs take form by departing from initial sketches that are then deviated from until the artifact is considered finished, rather than faithfully adhering to an original plan. Similarly, Andersen et al. (2003) likened artistic programming practice to that of the writer who moves phrases around and the painter who constantly repositions and re-paints, a technique known in linguistics as *commutation*. Also drawing parallels to painting, Graham (2004) maintained that creative programming, which he deliberately called “hacking” to distinguish it from standard programming, should be viewed and practiced using “[...] a language that lets us scribble and smudge and smear” (2004, 22). Graham (2004) chose the term “sketching” for his preferred style of programming to emphasize the process of “figuring out the program” as it is being written, thereby extending Blum’s “sculpting” metaphor. Coincidentally, the popular programming environment *Processing* (Fry, Reas, & Maeda 2007) also refers to projects as “sketches” to purposely incite authors to adopt a whimsical and artistic style of programming.

McLean & Wiggins (2008), extending Turkle & Papert’s (1990, 136) notion of *bricolage programming*, discussed the relationship between artists, their creative process, their program, and their artistic works through the analogy of a painter. In their model, McLean & Wiggins (2008) described bricolage programming as a creative feedback loop in which concepts are encoded as algorithms, which in turn produces output that is observed and evaluated by the artist-programmer, prompting him to adjust the initial concept and begin another cycle. Such a curious and explorative approach to programming closely resembles the natural *modus operandi* of graphic designers and stands in stark contrast to an engineering approach to programming, in which carefully planned, meticulously controlled, and mutually agreed specifications are imperative.

Non-programmers’ *laissez-faire* relationship to programming is further reflected in the way they refer to their activities. Verbs like “hack,” “bodge,” “tinker,” and “dabble” are frequently used to describe their working process, which also involves scavenging, foraging, copying, pasting, welding, and piecing together snippets of code from other sources.

2.3.4 Teaching programming to graphic design students

During the past two decades, it has become widely recognized that a variety of majors have need of computing skills, but a variety of approaches to programming is lacking. In response to this disparity, many computer science educators designed course materials and interventions to encourage non-STEM students to take computer science courses. Andersen et al. (2003, 109) observed that “teaching introductory programming to non-computer-science students and in particular to multimedia students with a liberal arts background is a big challenge [...]” Owing to the nature of their trade and its associated pedagogy, Andersen et al. (2003, 109) characterized liberal arts students as “more inclined to ‘open-ended topics’ in which analysis, discussion and interpretation are core competencies, and are less inclined to take interest in ‘closed, absolute topics’ like math and programming.”

Although programming may not be of primary interest to them, liberal arts students (Andersen et al. 2003) often possess a number of qualifications that can be useful when learning programming; for example, their evolved visual spatial thinking aptitude (Sutton & Williams 2010) positively influences their ability to learn programming (Webb 1985; Jones & Burnett 2008). Dorn & Guzdial (2006, 132) examined a group of graphic designers who engaged in end-user programming and concluded that this group of designers could likely benefit from some aspects of the formal teaching of programming. This view was shared by Pannafino (2013) and Reed & Davies (2006, 183). In this

respect, the 2007 Model Curriculum for a Liberal Arts Degree in Computer Science (Liberal Arts Computer Science Consortium 2007) can provide a comprehensive foundation to aid graphic design educators as they seek to integrate programming in their teaching. Also, Montfort (2016, 279–282) has provided brief outlines of generic syllabi for use with programming courses taught within the arts and humanities.

As discussed in section 2.2.7, the literature widely agrees that programming is a natural and much-needed addendum to the contemporary graphic design curriculum. However, for students across all disciplines, the prospect of having to learn to program can trigger negative emotions and cause fear and anxiety (Byrne & Lyons 2001; Radošević, Orehovački, & Lovrenčić 2009; Connolly, Murphy, & Moore 2009; Nolan & Bergin 2016).

2.3.5 Challenges associated with teaching programming to graphic designers

Educators who set out to teach programming to graphic designers will face several challenges.

First of all, to graphic designers, programming is a threshold concept (Cousin 2006). Meyer & Land (2003) considered threshold concepts “*akin to a portal, opening up a new and previously inaccessible way of thinking about something.*” Grasping a threshold concept becomes a transformative experience involving an ontological as well as a conceptual shift (Smith, Young, & Raeside-Elliot 2015, 1563); when a graphic design student who programs starts to think like a programmer, he will transition from *studying* programming to becoming a *working* programmer able to see the interrelatedness of graphic design and programming that were hitherto hidden from his view (Cousin 2006, 4). Heddy & Pugh (2015) argued that facilitating such big transformative learning experiences is innately difficult; therefore, they alternatively proposed small transformative experiences that are more manageable and achievable.

Next, the mere thought of being taught programming can demotivate many graphic design students (Andersen et al. 2003) and cause their comfort levels to drop drastically (Freyermuth 2016). Pettiway (2012) reported that graphic designers tend to focus on the inadequacy of their programming rather than trying to understand the salient issues that govern how and why programming interfaces with graphic design. As a way to remedy the students’ reluctance, Freyermuth (2016) argued that coding must be incorporated into more coursework throughout the graphic design curriculum to provide students with frequent opportunities to practice their programming skills.

Another major challenge relates to graphic designers’ general aversion toward math. Pearson (2011) recounted his experience of how novice artist-programmers became frustrated when the need for trigonometry was required to create even simple animations. According to Andersen et al. (2003), the majority of graphic designers lack mathematical qualifications, are scared of math, and typically have had very bad school experiences in that subject. Similarly, Tober (2014) has stated that “*creative students perceive themselves at a disadvantage because they feel they often lack mathematical understanding and numeracy skills.*” Ironically, Pepler & Kafai (2005) observed that when programming is integrated into design curricula, programming projects tend to focus precisely on mathematical and science content. This focus on logical structures and mathematical principles, Knochel & Patton (2015, 26) have conjectured, stems from the historical origin of programming in the fields of mathematics and engineering. However, as programming increasingly transgresses disciplinary borders, it becomes necessary to approach the task of teaching introductory programming to non-STEM students in a new and (to computer science) untraditional way (Andersen et al. 2003) that focuses less on math as a distinct topic.

Learning the intricate syntax of conventional text-based programming languages was perceived by Pettiway (2012) to be a major barrier to graphic design students, who feel more comfortable working

in spatial rather than text-heavy environments (Panda 2016) and think in terms of logically connected workflows (Watz in Pearson 2011). This might suggest that a visual programming language (VPL; also referred to as “node-based” or “dataflow” programming), in which programs are constructed by manipulating elements graphically rather than textually, are ideal for teaching programming to graphic designers. Recently, an increasing number of VPLs aimed at visual artists and designers have emerged (e.g., QuartzComposer, TouchDesigner, NodeBox, PraxisLIVE, vvvv, Max, Vuo) as complementary choices to the predominant textual programming languages (TPL) used in Creative Coding (e.g., Processing, p5.js, openFrameworks, Cinder, paper.js, three.js). While the efficiency of VPLs is highly debated (Leitão & Santos 2011; Panda 2016; Hjorth 2017; Iskrenovic-Momcilovic 2017), the main criticisms pertain to their inability to help users build a skillset that can be transferred to other programming environments and paradigms and their inability to be extended through new components without ultimately requiring the user to revert to TPLs. Those in favor of TPLs include Leitão & Santos (2011, 556), who have claimed that “*modern TPLs with user-friendly IDEs can be much easier to program and understand than the older ones, and they can surpass recent VPLs, especially in complex tasks.*” Despite not dispensing with the need for learning syntax, the steeper learning curve when it comes to TPLs provides good return on the investment. Unfortunately, as noted by Leitão & Santos (2011), most TPLs lack domain-specific primitives, which significantly delays the scripting process. Shim (2016a), however, has argued in favor of destressing the emphasis on learning syntax and stressing the focus on teaching graphic designers meta-skills to enable them to build design systems using logic and code. Shim’s idea aligns with the popularized notion of computational thinking, which is discussed in section 2.3.7.

2.3.6 What kind of programming should graphic designers be taught?

With educators, scholars, researchers, and professional practitioners all largely agreeing that graphic designers will benefit from learning programming, it is appropriate to ask what kind of programming they should be taught.

Several studies (Dorn & Guzdial 2006; Dorn, Tew, & Guzdial 2007) found that graphic designers were already taking part in significant programming activities despite having little to no formal training in programming. Nardi (1993) labelled these participants as *end-user programmers*, characterizing them as “*individuals [who are] making use of a class of applications that incorporate features like textual scripting, high-level declarative specification, programming by example, and automation or customization via wizards*” (Nardi 1993). Dorn & Guzdial (2006) argued that end-users would eventually have more and more sophisticated needs and would outgrow the standard affordances of their tools so that learning to program would become a natural progression. However, Maeda (2002) did not believe the solution lay in artists or designers pursuing degrees in computer science. Like Maeda, Denning (2004, 20), advocated the necessity of connecting programming abstractions to domain specific actions, remarking that “*there is little joy in worlds of pure abstractions devoid of action.*” Amiri (2011) has argued that this is particularly true in the case of end-users who are graphic designers, for they see programming as a pragmatic and utilitarian tool allowing them to create their often exploratory, experimental, and artistic artifacts. Graphic designers are not interested in the intricacies of a programming language for its own sake (Amiri 2011; Tober 2013).

The literature has different views on what kind of programming should be taught to students in non-computer science disciplines. In their comprehensive review of introductory programming courses, Pears et al. (2007) identified three different foci of teaching programming: the language and its syntax, problem solving, and the full cycle of system production (i.e., identifying a problem, analyzing the requirements, and solving the problem). Reading the literature through these foci

reveals a general consensus among researchers and educators that the latter two are most important when teaching programming in a design context; herein lies a fundamental difference between teaching programming to computer science students and graphic design students. Using the analogy of teaching a foreign language to linguists or tourists, Amiri (2011, 205) explained: “*The tourist needs the language to be able to communicate with people and explore the new environment and to get by. The linguist needs to understand the syntax, semantics and pragmatics of the language even if he or she never needs to communicate with a native speaker of that language.*” Extending this view, Guzdial (2015a) argued that it suffices students outside computer science to learn a small core of programming skills that will teach them enough computational thinking (see section 2.3.7) to help them design tools in their own domains. Graphic designers do not need to develop the competencies of professional software developers to make something useful.

According to Schneider (1978, 110), choosing a programming language suitable for use in education “*should be based on two critical and apparently opposing criteria: richness and simplicity - rich in those constructs needed for introducing fundamental concepts in computer programming [yet] simple enough to be presented and grasped in a one semester course.*” Interpreting Schneider’s view using the notion of “low threshold, high ceiling, and wide walls” (M. Resnick et al. 2005), it can be argued that effective tools (programming languages) must make it easy for novices to get started (low threshold), make it possible for experts to make sophisticated projects (high ceiling), and support and suggest a wide range of explorations (wide walls). One tool that fits within this description and is frequently mentioned in both graphic design and computer science literature is *Processing* (Fry, Reas, & Maeda 2007), a Java-based textual programming language popular among artists, designers, and architects and used widely throughout professional industries and education. Nevertheless, discussing specific languages to teach is irrelevant: programming languages come and go. However, despite their syntactic variations, they share an overarching set of fundamental computing principles. A study of 12 computing textbooks by Tew & Guzdial (Tew & Guzdial 2010) identified 29 computing constructs that are common across introductory courses. While these constructs can be taught to computer science students using an abstract language-agnostic approach, graphic designers—qua their preference for relatable, concrete, and utilitarian examples (see section 2.3.6)—will rely on a specific programming language to make tangible *software objects* to help them understand the computing constructs in relation to their native domain.

Briefly returning to the aforementioned three foci identified by Pears et al. (2007), it is clear that while a focus on language and syntax is less important, it cannot be disregarded when teaching programming to graphic designers. However, greater emphasis should be placed on the remaining two foci: problem solving and system production. For this purpose, computational thinking (see section 2.3.7) and contextualization (see section 2.3.8) can be helpful.

2.3.7 Computational thinking

Computational thinking (CT) has been a hallmark of computer science since the 1950s (Denning 2017, 33), making frequent appearances in debates on learning sciences and instructional technology. Most notably, scientist and educational theorist Papert discussed the term in his influential book *Mindstorms* (Papert 1980). Mateas (2005) considered CT (referring to it as “procedural literacy”) as a core part of the curriculum for new media practitioners. However, the most recent movement promoting CT began with an essay by Wing (2006), in which she promoted CT as a way to solve problems, design systems, and understand human behavior by drawing on concepts fundamental to computer science. Later, Wing, Cuny, & Snyder (2010) defined CT as an algorithmic problem-solving method “*represented in a form that can be effectively carried out by an information-*

processing agent.“ Despite her many contributions towards solidifying CT, Wing (2006, 2008; 2010; 2014) has been criticized for offering a vague and confusing definition of the term (Denning 2017), leaving teachers and education researchers without a consistent definition of what CT encompasses and how development of CT skills can be assessed (Brennan & Resnick 2012).

Seeking to alleviate the confusion surrounding CT, Denning (2017) offered two perspectives: *traditional CT* and *new CT*. Importantly, the kind of traditional CT taught in computer science is not the same as new CT, which should be taught in graphic design education. Explaining the difference using a paraphrased version of Hemmendinger’s (2010, 6) view on CT, Denning’s intention was not for graphic designers to think like computer scientists, but for them to understand and use computation to solve their problems, to create, and to discover new questions that can be fruitfully explored. To this end, *new CT* has been shown capable of offering syntactic, semantic, and pragmatic support for overcoming the cognitive challenges of learning programming (Repenning 2017).

Spreading to fields beyond computer science, CT has also resonated with several scholars situated in the field of graphic design. Among these are Knochel & Patton (2015), who have advocated for CT to be viewed as an urgent need within art and design education. Sharing this view is Pettitway (2012), who has maintained that CT can provide a gateway for combining practice-oriented and principles-oriented learning in a graphic design context and has described CT as an essential element in an effective curriculum model that intertwines programming as a foundational component of graphic design education. Mishra & Yadav (2013) even claimed that CT (when coupled with a solid understanding of a given domain) can augment human creativity, in particular through automation of problem-solving and algorithmic thinking. Generally, researchers and scholars consider CT, applied as a practice methodology within graphic design education, the key to understanding computation as system and logic for creative processes. However, in practice, CT is rarely found as part of a graphic design curriculum.

Repenning (2017), after evaluating experiences from courses employing CT, concluded that teaching programming does not automatically lead to students developing CT skills; thus, he has continued to suggest that CT must be taught explicitly as a separate topic using an appropriate contextual setting.

2.3.8 Contextualizing programming courses

Contextualization of programming courses is much debated among computer science researchers. Lukkarinen & Sorva (2016, 51) defined a contextualized computing course as *“one in which one or more application domains provide the motivation for learning Computer Science content and inspire the design of learning activities; these domains may be, and often are, external to Computer Science itself.”*

Forte & Guzdial (2005) argued that tailored computing courses aimed at non-computer science students can offer a more motivating and engaging context for the learning of programming, which can lead to an increase in the students’ motivation and engagement and help reduce their anxiety and negative perceptions of programming. Similarly, Andersen et al. (2003) suggested that by *“talking to students in a language they know [and] choose a metaphor already known to them, we could be halfway [to teaching them programming].”* Another study by Dorn & Guzdial (2006), specifically involving graphic designers, has also suggested that computer science educators should consider new contextualizations of programming courses to match the settings in which end-user programming takes place. From the perspective of a design educator, Pettitway (2012) has agreed that such contextualized courses could help decrease the barriers between abstract programming knowledge and practical application by providing a more relational and transformative approach. However, Guzdial (2010) warned that students who learn programming within a context might over-

specialize that knowledge (e.g., see a for loop only as a tool to step through an array of pixels). While this is problematic for computer science students, who must obtain a context-independent understanding of computing concepts, graphic design students contrarily aim to use programming in one highly specific context. Indeed, Guzdial (2010) has argued that contextualized approaches are fine for students who will only use computing within a single domain.

Contextualized approaches to introducing students to computer science have gained momentum and recognition in recent years. Specifically, *Media Computation* has successfully used (audio)visual contexts to teach programming to computer science students (Guzdial 2003, 2009; Greenberg, Kumar, & Xu 2012; Xu et al. 2016; Dodgson & Chalmers 2017; Xu, Wolz, & Greenberg 2018). Maxwell & Taylor (2017) have reported that while these contextualized approaches do not affect the students' learning outcomes, they have a positive influence on the students' engagement. While not denying their effectiveness in making computing fun and relevant, Kay & Road (2011) have encouraged educators to discuss whether contextualization is simply bait used to lure students into computer science studies.

2.4 Pedagogy

2.4.1 Pedagogy in graphic design education

Traditionally, the discipline of graphic design (earlier referred to as *commercial art*) was a trade passed on and learned by apprenticeship (see section 2.2.1). In the early 1800s, studio training emerged as a teaching tradition at Ecole des Beaux-Arts in France (Shaffer 2007). This training involved open-ended projects with structured critiques and led to a public showing and evaluation of work. Later, as dedicated graphic design education emerged, studio training became the preferred instructional setting. Initially, this involved educators passing on their knowledge through didactic lectures and demonstrations, after which students would be given the same assignment to complete while being supervised by their instructor. Later, the Bauhaus teaching structure, developed by Walter Gropius in the early 1920s, introduced an unprecedented type of course that encouraged students to produce creative designs based on their own subjective perceptions (Wick 2000). The unifying pedagogy of the Bauhaus posited *artist-teachers* as those who used their artistic discipline to inform educational issues (Daichendt 2010). Fusing art, technology, and pedagogy, Bauhaus became an influential landmark in the history of modern art and design education (Daichendt 2010). Bauhaus professor Moholo-Nagy later went on to introduce the teaching model in the US and initiated a close relationship with the local graphic design industry, thereby establishing a tradition of strong ties between design schools and design industry, which has been upheld ever since. Today, graphic design education remains heavily influenced by professional practice, and this leads to classroom activities being designed to resemble the industry's nature and situational context (Logan 2006).

A paradigmatic change in graphic design education came about with the advent of computers. As the graphic design trade increasingly relied on computers, obtaining an understanding of digital technology became an absolute necessity for graphic design students. Having traditionally relied on gut-feeling, intuition, and a trained eye, graphic design education suddenly had to adapt its teaching to fit the absolute, logical, and "mechanical" nature of digital media. As mentioned in section 2.2.3, initially, early critics among design educators did not consider computers a pedagogic tool (Faison 1995). Undeterred, however, other educators embraced computers and began to explore their pedagogic potential and concomitant challenges, resulting in the formation of new learning theories.

Currently, design pedagogy at large is in a state of flux as it struggles to respond to constant technological and societal changes (Hardman 2017). Hardman, while embracing the new design

paradigm, has also concluded that it holds the potential to render traditional design pedagogical virtues superfluous unless educators remain vigilant.

2.4.2 Pedagogy in Computer Science education

When computer science began to form as a separate academic discipline in the 1950s, education had to start from scratch, designing curricula, training staff, writing textbooks, even constructing mental models for thinking about computing (Tedre, Malmi, & Malmi 2018). The initial courses were highly theoretical, drawing mainly on mathematics and electrical engineering (Tedre, Malmi, & Malmi 2018). These courses established a tradition of teaching computing through a passive lecture style with limited interaction in practical sessions (Felder & Brent 2005; Van Der Post 2010). Typically, students would attend a few short lectures and practical sessions per week, completing homework in their own time, supported only by their own notes and textbooks.

In the early 1970s, computing saw a shift from mathematically-oriented discipline to a more diverse discipline that began to emphasize hands-on work, programming, and applications. Educators in liberal arts colleges proposed alternative computing curricula better suited for their own purposes. These curricula explored new pedagogical strategies and teaching activities that were anchored in a more pragmatic, experiential, and explorative approach to computing. A major push towards a radical shift in computing pedagogy came in Papert's (1980) book *Mindstorms* (discussed further in section 2.4.5). Recently, computing education has also focused on teaching problem-solving strategies and processes to avoid fixed-problem statements (Tedre, Malmi, & Malmi 2018), with many courses deploying Wing's (2006) idea of computational thinking (see section 2.3.7) using various pedagogic-didactic measures.

Over time, many pedagogic theories, methods, and strategies have been used to teach programming, including cognitive apprenticeship (Collins, Brown, & Newman 1989), extreme apprenticeship (Vihavainen, Paksula, & Luukkainen 2011), problem-based learning (Stevenson 2001; Fee & Holland-Minkley 2010; Nuutila, Törmä, & Malmi 2005), pair programming (Williams et al. 2000; Nagappan et al. 2003), and peer-instruction (Mazur 1997; Simon et al. 2010); however, the literature reveals a lack of consensus on which methods are the most effective for teaching programming—likely because there is no “silver bullet” answer.

Noteworthy, considering the scope of this dissertation, a number of computer science educators have recently attempted to integrate art and graphic design into their existing curricula (e.g., Greenberg, Kumar, & Xu 2012; Xu, Wolz, & Greenberg 2018). In particular, the branch of computer science known as *Media Computation* (Guzdial 2003; Forte & Guzdial 2004), has abandoned programming courses that favor properties such as predictability, robustness, and correctness in pursuit of courses focused on making products that are understandable, entertaining, aesthetically satisfying, educational, and provocative. These courses are increasingly taught using a pedagogical technique known as studio-based learning (SBL) (Carter & Hundhausen 2011), which is discussed further in section 2.4.7.

2.4.3 Educational Paradigms

Mass schooling in the early industrial society relied on *didactic teaching* as its educational paradigm (Kalantzis & Cope 2010, 202). Prominent features were classrooms with desks in rows, a blackboard at the front, textbooks, recitation, memorization, rote learning, and with all students doing the same work and moving ahead at the same pace. Later, the educational paradigm changed in favor of *authentic education* (Kalantzis & Cope 2010, 202). This emphasized experiential learning; group dialogue; students expressing their own opinions and points of view; use of images/diagrams/visual

representations to supplement written text; educators acting as learning facilitators; and individualized, self-paced learning. As Kalantzis & Cope (2010, 202) have argued, we have now entered yet another educational paradigm, namely that of *transformative learning* (Mezirow 1991, 1997). There is no longer any need for classes to be collocated; students collaborate using networked computers, whether at home, at school, or at relevant locations in the field. Also, it entails a balance of knowledge processes: experiential, conceptual, analytical, and applied. The educator has become a designer and manager of learning. Learning has become contextualized, and student differences are catered for through flexible teaching approaches that utilize the complexities and richness of digital media. Willis (2007) has encapsulated the gist of transformative learning in a speculative pedagogic model, centered around algorithms, that suggests viewing courses as databases with multiple points of entry, abandoning formal learning outcomes to allow for unexpected outcomes, moving beyond the acquisition of information to an enhancement of the ability to learn, and the deployment of expertise using a network as metaphor.

According to Kolko (2012b), design studio training (see section 2.4.1) is an exemplary model for how transformative learning theory is used to allow experiential learning to occur. He has argued that in a design studio, knowledge is produced, not disseminated, through projects that involve a constant cycle of constrained making and guided reflection. This recasts the educator from the role of lecturer to facilitator (Kalantzis & Cope 2010, 205), that is, to one who “[...] *has had a certain quality of creative experience who can anticipate, during the knowledge-generation process, where various patterns, methods, approaches, or techniques will be most effective*” (Kolko 2012b, 82). Kolko believed that design studio teaching was an effective way of facilitating transformative learning, by letting “*students have an experience, and [...] [control] the majority of that experience. This means they have approached the learning from within their own frame, a place of comfort. And then, in an emotionally safe environment, they have been nudged outside of their own frame into a place of discomfort*” (Kolko 2012b, 83).

2.4.4 Learning styles and teaching styles

Students have different attitudes about and aptitudes for learning, as well as different ways they respond to specific teaching environments and activities (Felder & Brent 2005). Similarly, educators have different approaches and methods of teaching. These are referred to as *learning styles* and *teaching styles*, respectively. For many years, educators believed that the same instructional method could be used to teach all students (Capretz 2002). Later, researchers (Capretz 2002; Felder & Brent 2005) argued that effective teaching and learning could be achieved if educators adapted their teaching style to match the preferred learning style of their students; however, the notion and usefulness of learning styles is not accepted by all researchers (Felder & Brent 2005; Willingham, Hughes, & Dobolyi 2015). Nevertheless, studies have shown consistent differences in students’ results in accordance with their assessed learning style (Felder & Brent 2005). Regardless of the ongoing debate on the *raison d’etre* of learning styles, there is an acceptance of learning styles as useful mechanisms to facilitate a metacognitive process among students by helping them reflect on and become aware of how they acquire knowledge.

Coffield et al. (2004) identified more than 70 models for analyzing and understanding different learning styles. The most prominent models found within the literature examined in this dissertation are:

- Myers-Briggs Type Indicator (*MBTI*) (Briggs-Myers et al. 1998),
- Kolb’s Experiential Learning Cycle and Learning Style Inventory (*LSI*) (Kolb 1984),

- Felder-Silverman Learning Style Model (*FSLSM*) (Felder & Silverman 1988) and Felder-Soloman Index of Learning Styles (*ILS*) (Felder & Soloman 1997),
- Honey and Mumford's Learning Styles Questionnaire (*LSQ*) (Honey & Mumford 2000),
- VARK Learning Style Inventory and Questionnaire (*VARK*) (Fleming & Mills 1992).

Each learning style model has its own specific theoretical foundations and usually classifies students using a number of scales, based on their responses in a questionnaire. By administering a learning style questionnaire to his class, an educator can obtain a profile of the students and adapt his teaching style accordingly. To this avail, most learning style models provide intrinsically linked educator guidelines. The literature further reveals that both graphic design educators and computer science educators alike make use of prescriptive instructional design models to help plan and structure their teaching activities, for example, ADDIE (Allen 2006), ARCS (Keller 2010), and 4MAT (B. McCarthy 1990).

2.4.5 Constructionism as learning theory

Learning and teaching styles must be applied within the larger context of a learning theory. One particular learning theory, constructionism, was developed in the late 1960s by Seymour Papert, who was pioneering investigations of technology-supported pedagogy using the programming language LOGO to teach children math through programming the movements of a robotic turtle (Papert 1971, 1980). However, thereafter, the use of tools for teaching programming to children remained broadly uninvestigated until the availability of visual programming languages (Lye & Koh 2014). Still, as an attestation of its pedagogical efficiency, LOGO remains in use today (Anderson 2018; Caspersen & Christensen 2000). Papert's early work informed Harel & Papert's influential theory of constructionism (1991), derived from Piaget's constructivist epistemology, which theorized that children learn by assimilating new knowledge into their existing mental schema (Knochel & Patton 2015, 25). Constructionism is about playing with programmable objects as material and assimilating the activities into the students' mental schema while they are also learning how to accommodate the rules of the programming language to make the code work. In particular, constructionist learning theory lets students use their pre-existing knowledge to acquire new knowledge in a different domain through hands-on, playful exploration. Papert, summarizing this approach, explained: "*First, relate what is new and to be learned to something you already know. Second, take what is new and make it your own: Make something with it, play with it, build with it*" (Papert 1980, 120).

Overall, constructionist learning theory appears to be particularly well suited to help bridge the gap between abstract programming concepts and the action-driven, experiential way designers acquire new knowledge. In section 3.3.2, the origins and philosophy of constructionism is described further.

2.4.6 Practicum pedagogy

Traditionally, graphic design has been a trade learned by practice, not by analyzing and studying its theoretical concepts. Accordingly, teaching graphic design has fostered an abundance of pedagogical approaches, most of which are based on experiential learning, hands-on experiences, and experimentation activities. Schön (1987), referring to this as *practicum pedagogy*, concluded that it was a significant factor in the development of design competence; yet, he still critiqued practicum pedagogy as valuing the "tacit theories" of practitioners over formal theories and models, thus keeping knowledge "sealed" (i.e., tacit, implicit, and inaccessible).

Schön (1983, 1987), extending the work of Dewey (1933), argued that practice can be improved through contemplative reflection. To this end, Schön proposed the temporally linked concepts of *reflection-on-action* (looking back on an accomplished task to review the actions, thoughts, and products), and *reflection-in-action* (reflecting while in the act of carrying out a task). Later, Killian and Todnem (1991) added a third concept of *reflection-for-action* (reviewing what has been accomplished to form constructive guidelines for similar future tasks). Schön's thoughts on reflective practice have been hugely influential—almost canonical—in the way they have been applied in graphic design education and continue to inspire the development of new frameworks for reflective practice (Bain et al. 1999; Grushka, McLeod, & Reynolds 2005; Liston & Zeichner 2013). Particularly within the design studio teaching tradition, plenary reflection-on-action frequently occurs in the shape of various forms of critique (crit) sessions facilitated by the educator (Hokanson 2012; Lawson 2004; Stolterman 2008). Such sessions provide a rich opportunity for students to critically reflect on their own work by comparing and contrasting it to the work of their peers.

While Schön's anti-positivist and anti-technical rationality views resonate well with graphic design educators, computer science educators have questioned the relevance and usefulness of Schön's reflective scheme and its ensuing design education as suitable instructional strategy in non-design education (Cáceres 2017, 35–36). A reason for this was pointed out by George (2002), who observed that the act of reflecting differs depending on the underlying nature of knowledge for that particular discipline. A study by Chng (2018) on reflective practice in computing classes in STEM education has revealed the use of several other reflection models (Gibbs 1998; Johns 1995; Fekete et al. 2000; George 2002; Humphrey 2000; Bender & Vredevoogd 2006; Zarestky & Bangerth 2014).

In sum, the literature indicates a dichotomy between models used for reflection in design education versus reflection in programming education. Navigating this space is Montfort (2016), who has ultimately stressed that practicum pedagogy and its associated models for reflection must be employed when teaching programming to students within the arts and humanities.

2.4.7 Convergence: Teaching computing in a design studio environment

Briefly mentioned in sections 2.4.1 and 2.4.3, studio teaching, with its emphasis on practicum pedagogy and reflective practice, has long been the preferred instructional setting in graphic design education. Considering it a central educational device, Shaffer (2007) described the studio as a coherent learning system in which pedagogical processes and theoretical perspectives come together to create an effective learning environment. Each student is encouraged to explore his individual ideas, internalize the fundamental of the design discipline, and develop a unique approach to design practice (Van Der Post 2010, 66).

In his closing address for the CHI 1990 conference, Winograd (1990) advocated the use of the studio teaching model in computer science. Early attempts to restructure software development courses to use a studio teaching model are recounted by Tomayko (1991), who observed that *"the core courses and prerequisites are adequate for teaching the tools of design, reuse, and management, but the more creative aspects are best taught in a studio environment"* (1991, 301). Interestingly, considering the scope of this dissertation, Tomayko (1991) also observed that *"many software engineering and computer science instructors (and their students!) fear the artistic nature of their work."* As computer science increasingly sought to adopt the studio teaching model, differences between the prevalent educational paradigms became evident, leading Reimer & Douglas (2003, 195) to conclude that *"studio teaching is radically different from the usual computer science instruction of lecture/lab/discussion."* Similarly, Van Der Post (2010), comparing studio and computing teaching

methods, concluded that several differences exist pertaining to structure, pedagogy, epistemology, and the educator's role.

Following an adaptation phase, studio-based learning (SBL) has recently become increasingly popular in computer science education (Carter & Hundhausen 2011), in which it is used to reinvigorate a computing curriculum and give students exposure to industry practice. Still, lectures continue to be the prevalent teaching method, whereas SBL is used mainly to reinforce the material presented in the lectures.

Graphic design education, on the other hand, faces a key challenge in finding a natural way to embed programming into the curriculum that is compatible with the existing studio teaching model. Currently, the dominant paradigm for teaching programming in design schools is based on conceptual models of programming lifted from software engineering (Amiri 2011, 204). According to Van Der Post (2010), this engineering bias, coupled with design educators' insufficient technical knowledge and skills, causes programming courses to be planned with a disproportionate focus on technology. In response, Tzankova & Filimowicz (2017) and Tober (2014) have both addressed an urgent need for developing and maturing the pedagogy, curriculum, and educators' professional development in the interdisciplinary field of computation and creative making.

2.4.8 Developing pedagogic strategies to teach programming to graphic designers

Almost two decades ago, Carmen (2000, 71) opposed the increasing focus on teaching software packages in design education (see sections 2.2.4 and 2.2.5) by arguing that *"unless educators take a lead in developing appropriate pedagogies for these new electronic media and forms of communication, corporate experts will be the ones to determine how people will learn, what they learn, and what constitutes literacy."* Taking a lead though, particularly in developing programming courses, proved hard for design educators, who lacked both technical skills and applied know-how (Maeda 2002). Despite many calls from scholars, researchers, and educators to evolve pedagogic strategies to teach programming in graphic design education (see section 2.2.7), very little happened. Three years later, Peppler & Kafai (2005) saw no signs of design education evolving to reflect how programming were used by professional artists and designers. Pettitway (2012), discouraged by the ongoing lack of models for introducing programming in graphic design pedagogy, made a plea for design educators to devise an effective curriculum model that intertwines programming as a foundational component of graphic design education. Recently, however, as more design schools have begun to offer programming courses, the development of tailor-made pedagogic strategies has gained momentum.

The literature describes several attempts at developing custom pedagogic strategies to scaffold liberal art students' journey into programming, including computer-independent activities such as theatre and role-playing (Maeda 2009), paper and cardboard prototypes (Maeda 2009; Artut 2017), workbooks (Maurer et al. 2013), board games (Drake & Sung 2011), and robots (Xu, Blank, & Kumar 2008). While such strategies can successfully capture students' attention and aid their construction of mental models of computing principles, at some point, graphic design students must engage with programming through hands-on exercises to fully understand the properties of code as a creative material. Thus, pedagogical strategies that directly involve code as an element of the teaching must also be considered. According to Tober (2012b, 382–383), such strategies should also be devised to support students outside class, allow for different skill levels through differentiated learning, and make any instructional material available for repeated viewing.

2.4.9 New technology = new teaching methods

When devising pedagogic strategies, educators must consider the new teaching methods made possible by technology.

Blended learning was defined by Friesen (2012, 1) as “[...] *the range of possibilities presented by combining Internet and digital media with established classroom forms that require the physical co-presence of teacher and students.*” Four discrete models of blended learning exist, spanning a continuum from traditional, tech-free classrooms to online-only learning (Philadelphia Education Research Consortium 2014). Studies report positive experiences of using blended learning in design education (Bender & Vredevoogd 2006; Kim 2016; Warburton 2017; Fleischmann 2018).

Flipped classroom (Abeysekera & Dawson 2015) is a type of blended learning that delivers instructional content online outside the classroom and focuses on traditional homework activities in class. Pioneering work by Lage, Platt, & Treglia (2000) investigated what they referred to as “the inverted classroom.” Their work was later extended by other scholars who subsequently renamed the concept as “flipped classroom.” It caught widespread public attention around 2011, but despite its growing popularity, flipped classroom is still generally under-evaluated, under-theorized, and under-researched (Abeysekera & Dawson 2015). Yet, a crucial point can be drawn from the existing, available research: Amresh et al. (2013) have concluded that the flipped classroom is a more effective pedagogy than passive lectures. A testimony to this view are the empirical observations on the effects of flipping an introductory programming class recounted by Shiffman (2018), who found himself able to devote more time to helping, supporting, and preparing students for the next classes.

The topic of digital textbooks (also referred to as *hypertextbooks*, *e-textbooks*, or *e-texts*) in the literature is not new. Early definitions by Brewer (1998) and Ross & Grinder (2002) saw the medium of text as the core enriched by multimedial assets and interactive learning visualizations to support meaning making. Today, nearly two decades later, an abundance of digital textbooks of many different levels of pedagogical, technological, and aesthetical sophistication are available to students. The crux of the matter, however, is their amount of interactivity. Studies (Rockinson-Szapkiw et al. 2013; Nichols 2016; Walsh 2016) have revealed no significant difference to learner comprehension if they read from printed materials or screens. Other studies (Stoop, Kreutzer, & Kircz 2013; Ericson, Guzdial, & Morrison 2015) have implied that digital textbooks containing interactive elements can increase student performance and enhance cognitive and affective learning. Programming lends itself particularly well to being taught using digital media, and, within the scope of this dissertation, Shiffman’s interactive digital textbook, *The Nature of Code* (Shiffman 2012), provides a prime example of how interactive, code-based examples can be integrated in a visual context to aid comprehension. However, as suggested by Gu et al. (2015), many aspects related to designing, developing, and teaching using digital textbooks still require in-depth study.

Lastly, when discussing technology-based teaching, the recent rise of Massive Open Online Courses (MOOCs) must be addressed. Kaplan & Haenlein (2016, 441) defined a MOOC as “*an open-access online course that allows for unlimited (massive) participation.*” MOOCs use traditional course materials, such as short, pre-recorded lectures and selected texts, exercises, and assignments, as well as interactive elements such as quizzes, tests, polls, demonstrations, and editable worked examples. Forums are used to facilitate community interaction and peer-assessment among students. MOOCs represent a plausible, though unresolved, online teaching model that challenges many long-held beliefs of learning and teaching (McNamara 2015). While seemingly allowing for low-cost learning at scale, several shortcomings of using MOOCs in formal computer science education have been identified and described by Ericson, Guzdial, & Morrison (2015, 170). Yet, Falker et al. (2016), accounting for a MOOC-based, introductory programming course specifically designed for a media

computation context, concluded that learning design approaches can be successfully transferred to the massive, online scale. Likewise, McNamara (2015) saw great potential in the use of MOOCs in design education, and he has projected that teaching approaches and efficiencies seen in MOOCs will be incorporated into existing teaching processes as a replacement for certain aspects of face-to-face teaching. Finally, reflecting back on their MOOC *Creative Coding* (FutureLearn 2014), Guglielmetti & McCormack (2017) have asserted that the most promising aspect of MOOCs is through the social learning they enable, despite the low-bandwidth dialogue of online forums.

In sum, graphic design educators will be challenged, perhaps to their dismay, to adapt existing pedagogical strategies to fit a new techno-centric and rapidly changing, transformative educational paradigm, which threatens to erode the prevailing studio-based, face-to-face practicum pedagogy known today.

CHAPTER 3: METHODOLOGY

In this chapter, I describe my paradigmatic position and explain its influence on this study. I then account for the research design used to answer my three research questions. Lastly, I suggest a means of validating my results.

3.1 Introduction

The objective of this study has been to investigate how programming should ideally be taught to graphic designers to account for how they learn and how they intend to integrate programming into their vocational practice. From the thesis statement (section 1.6), three specific research questions were identified (section 1.7). To ensure the research questions were aligned and worked together to address the overarching research question of this study, a good research design had to be developed. Let me begin by discussing how I perceived the relationship between my research activities and my practice-based background.

3.2 Mitigating viewpoint ambiguity

My background in the field of graphic design (section 1.2) inevitably informed and influenced the initial choice of topic for my investigation. Instinctively, I was inclined to approach my research by assuming the familiar role of a designer who applies a designerly, problem-framing and problem-solving mindset to pragmatically improve a process (teaching programming to graphic designers) through the design of a product (instructional design). However, to aid my research, I sought to provide an answer to my research question through a structured scientific investigation based on theoretical and empirical studies of existing practices. This required me to also assume the role of researcher and adopt a scholarly mindset. Bluntly making a distinction between these two roles, Nelson (2013, 4) claimed that “*designers work by synthesizing ideas within real-world situations that involve creating artifacts and managing environments, while scientists think analytically within an abstract, symbolic world.*” I did not, however, intend to choose one role over the other. Instead, I chose to let the inherent methodologies of each role inform how I conducted my research. I saw great strength in qualifying the method best suited for a particular purpose as a result of a dialectic dialogue between the methods used in either design or science. This view is supported by Faste and Faste (2012), who argued that both “*scientific left*” research practice and “*designerly right*” design processes can benefit from a wider perspective (2012, 4). They also maintained that designers tend to use a variety of research methods when pragmatically appropriate, some rigorously scientific and others less so (2012, 4).

Assuming the simultaneous roles of designer *and* researcher inevitably caused ambiguity in my personal viewpoint, ultimately causing me to ponder whether design was a form of research, or

research a form of design—a question that has been asked by other researchers (Nelson 2013; Faste & Faste 2012). Specifically, Nelson (2013, 3) has asked:

Do instructional designers use scientific principles and contribute to the development of theory as they engineer instructional products intended to produce learning by students? Or, do instructional designers employ 'designerly thinking' to solve 'wicked problems' that can't be approached using scientific thinking, yet produce the same kind of products and learning outcomes for students?

The answer to this question, of course, is not a simple binary “either/or”; rather, it creates a fuzzy “both/and” state between each of the statements. I considered this a further validation of my choice of approaching my study with both a designerly and scientific mindset, knowledge set, skillset, and toolset.

I chose to mitigate the ambiguity of my viewpoint by embracing them all. Was I a researcher investigating how programming is introduced in graphic design education? Was I a graphic designer doing research on programming education? Was I an educator designing research-based guidelines for teaching programming? Truth is, I was all of these. Simultaneously.

3.3 Paradigmatic position

Encasing my composite viewpoint, however, was my personal stance. Heavily influenced by my background as a trained practitioner with a long professional career working in the graphic design industry, I had—knowingly and unknowingly—come to adopt a pragmatic approach in my way of life. Designers in the graphic design industry see a constant demand by clients, art directors, and project managers to deliver a steady stream of tangible and billable artifacts. Working under the trammels of strict deadlines, project specifications, and technical and economical limitations inevitably requires any graphic designer to develop a pragmatic approach to his work, constantly having to balance questions like “what is doable within the given time frame?” “what materials are at my disposal?” and “what do I assume will work best?”

Given that I reckoned myself unable to abolish or suppress my inherent pragmatic world view and designerly, problem-framing and problem-solving way of thinking, choosing pragmatism as my paradigmatic position for this study seemed straightforward. However, when I transitioned from practicing to teaching graphic design, my new role as educator required me to supplement my disciplinary knowledge with pedagogical methods in order to transfer my knowledge to my students. I became aware of constructionism, a branch of constructivism, which also shares many traits with pragmatism. I knew from personal experience, that the ideas and methods of constructionist learning theory were useful and efficient for teaching graphic design students, and I therefore chose to supplement my mainly pragmatist position with constructionism. The difference between pragmatism and constructionism is in the epistemology (i.e., the philosophical assumptions as to what constitutes knowledge). Constructionism aims to build understanding and meaning, whereas pragmatism aims to find solutions and solve problems. Below, I will briefly outline the ideas associated with pragmatism and constructivism.

3.3.1 Pragmatism

The philosophical tradition of pragmatism was introduced around the late 1870s by James, Dewey, and Peirce (Biesta & Burbules 2003). As a research paradigm, modern pragmatism advocates the use

of a mixed methods research, "*sidesteps the contentious issues of truth and reality*" (Feilzer 2010, 8), and "*focuses instead on 'what works' as the truth regarding the research questions under investigation*" (Tashakkori & Teddlie 2010, 713). Pragmatists believe the purpose of knowledge is to improve existence through action. This calls for knowledge that points to a better and possible world and useful ways to reach this improved state (Goldkuhl 2011). According to Goldkuhl (2011), essential traits of design research can best be justified within the epistemological foundations of pragmatism:

- a focus on utility, usefulness, and contribution to practice;
- knowledge development through building and intervention;
- problematic situations as a starting and driving point for inquiry and design;
- a search for what is possible and desirable;
- going beyond description, and aiming for prospective, normative, and prescriptive knowledge.

The utility of pragmatism is that it aims to find middle ground between philosophical dogmatism (Johnson & Onwuegbuzie 2004). Pragmatism offers researchers the freedom to choose the best methods to answer research questions at hand and advocates a balance between subjectivity and objectivity throughout the research (Shannon-Baker 2016). Traditionally, there have been strong intellectual and conceptual affiliations between graphic design and pragmatism (Moszkowicz 2013). Furthermore, Dewey is widely recognized as an influential educational reformer who employed the pragmatic approach in his work. His major writing on aesthetics, "*Art as Experience*" (1934), is considered seminal reading by many design school programs. As such, pragmatism has strong ties to both graphic design and education, two of the research areas of this study.

3.3.2 Constructionism

Constructionism is a prevalent learning theory that has its basis in Piaget's and Vygotsky's constructivist learning theories (Papert 1980; Ackermann 2001) and is connected with the modern theory of experiential learning developed by Kolb (1984). Constructionism grew out of a set of innovative educational research projects performed in the 1960s and 1970s at the MIT Media Laboratory by Papert, Resnick, and Ackermann (Kafai & Resnick 1996). The projects were meant to illustrate how computational technologies can transform the concepts of learning, education, and knowledge. Papert subsequently devoted his career to developing and promoting constructionism. In 1980, he published his seminal book *Mindstorms: Children, Computers, and Powerful Ideas* (Papert 1980). Papert has also been a proponent of bringing technology to classrooms, introducing the programming language LOGO (Papert 1980)—along with a robotic turtle—as a way to teach mathematics to children. While constructionism has been used primarily to teach science and math, its origin in media studies also makes it suitable for students who simultaneously engage with media theory and a complementary praxis.

The guiding principle in constructionism is in accordance with the basic constructivist assumptions, namely, that learners must actively construct and reconstruct knowledge based on their own experiences. Constructionism advocates student-centered, discovery learning where students use information they already know to acquire more knowledge. Further, constructionism holds that learning can happen especially felicitously when people are active in making tangible objects in the real world that enable discussion, examination, probing, and admiration (Kafai & Resnick 1996; Papert 1993). Despite it often being described as "learning-by-making," Papert & Harel (1991, 1) maintain that constructionism should be considered "*much richer and more multifaceted, and very much deeper in its implications [...]*"

Constructionism, and its associated learning theories, naturally sits well with graphic design students as it prescribes the making of tangible, shareable objects as a means of acquiring knowledge across domains. Specifically, within the scope of this study, constructionism's cross-domain bridging ability has been particularly useful and influential in my work towards an improved instructional design for Creative Coding courses in graphic design education.

3.4 Type of research

Next, with the intent of qualifying methods for my study, I sought to determine the kind of research I was about to conduct, as each type of research suggests a range of methodological approaches to be used. To do this, I used the typology of research in the creative arts and design proposed by Brown, Gough, & Roddis (2004). The authors list four types of research: *scholarly research*, *pure research*, *developmental research*, and *applied research*. An adapted version of the authors' table listing each type and their mutual differences is shown in table 3.1.

Correlating the nature of my research questions (discussed in detail in section 3.5) with the research types defined by Brown, Gough, & Roddis (2004) suggested that I would be undertaking both developmental and applied research; my research questions described a specific problem (teach programming) to be solved within a specific context (graphic design students), seeking to improve existing processes (instructional design) through a systematical testing of hypotheses. In addition, aligning with my pragmatist, designerly stance, the impetus motivating this study was to test relevant issues (developmental) to solve a specific problem (applied), rather than ask key questions (pure) to create intellectual infrastructure (scholarly). Explicitly stated as mandatory in the call for proposals for this study was that its outcome had to be put to immediate practical use, thus making an additional case for applied research activities. That my research could be understood as both applied and developmental was based on the fact that most research concerning instructional design is either applied or developmental (Nelson 2013, 3).

Noticeably absent from Brown, Gough & Roddis' typology, however, is the type referred to as *design research*. While Akker et al. (2006, 4) roughly equate design research with Brown, Gough, & Roddis' category of developmental research, the momentum and popularity gained by design research as a type in its own right led me to investigate whether the research I was about to conduct also fell in the design research category.

3.4.1 Design research

Design research is a broad term with a long history. Dating back to the 1960s within academia, design research has traditionally referred to the study of design itself, its purpose, and processes. Significant contributions have been made across the decades by researchers such as Schön (1983, 1987), Cross (1982), Frayling (1993), Friedman (2003), Buchanan (2001), and Lawson (1980, 2004). Yet, the concept of design research is still much debated, with the discussion revolving around defining what research is and where it belongs in design education and practice (Frankel & Racine 2010). Recently, non-academic designers have also adopted the term when referring to research that is integral to the design work itself or inquiries that are part of designing, and not directly about design (Faste & Faste 2012). This has resulted in a proliferation of terminology and lack of consensus on definitions. Today, the term design research has become part of the common vernacular in the field of design and is used to describe a wide range of activities, from scholarly investigations to simple methods used by practitioners as part of their design work (Faste & Faste 2012). In this study, however, I have used the term design research to refer to the traditional definition as it is used within academia.

Scholarly Research	Pure Research	Developmental Research	Applied Research
Creates intellectual infrastructure	Asks key questions	Tests relevant issues	Solves specific problems
Records—questions asked, issues explored, solutions proposed—in the field	Sets and explores hypotheses experimentally through logic and/or intuition	Contests and tests existing hypotheses/theories originally	Examines specific cases systematically
Documents the knowledge gained from pure, developmental, and applied research along with the results	Searches for pure knowledge	Tests and applies the outcome of pure research by harnessing existing knowledge to determine new methods or ways of achieving some specific and pre-determined objective	Applies developed knowledge and methods to the examination of a specific context in order to solve a problem in that context
Compiles the resources, methods, tools, and models used in research within the pure, developmental, and applied research fields	Uncovers issues, theories, laws, or metaphors that help to explain why things operate as they do, why they are as they are, or, why they appear to look the way they do	Focuses on how things are done by (a) generating useful metaphors for organizing insight and (b) developing specific theories that can be used to predict the future in specific situations	Creates new or improved artifacts, products, processes, materials, devices, services, or systems of thought and ways of seeing
Maps the field in which issues, problems, or questions are located	Discovers/generates significant new facts, general theories, or reflective models where immediate practical application or long-term benefits are not a direct objective	Tests and reworks knowledge through (a) the generation of alternative visual models, experiences and thought systems and (b) the evolving of special methods, tools, and resources in preparation for solving specific problems in a specific context where immediate practical application is a direct objective	Applies outcomes from pure and developmental research to a specific context where long-term benefits are a direct objective
Disseminates the results of research to the research community and to others who might be interested in them	Yields potentially unexpected results and original theories, discoveries, or models that are unrelated to the disciplines in which the research has been conducted	Produces results that may be usable across many contexts in pure and/or applied research, establishing connections between individual cases and disciplines	Produces results that cannot usually be directly applied to other contexts because of the specificity of the context from which information is gathered

Table 3.1: Types of research in creative arts and design (adapted from Brown, Gough, and Roddis 2004).

Three dominant categories of design research have emerged and are widely accepted in the literature: Research for design, research through design, and research about design (Frayling 1993; Friedman 2003; Downton 2003).

Research for design is the category of research that most practitioners and academics associate with the term “design research,” likely because it has the most potential to contribute to successful design outcomes (Frankel & Racine 2010). In this category, both quantitative and qualitative research methods may be appropriate (Roth 1999, 22–25). Approaches such as Action Research (Archer 1995) or design-oriented research (Fallman 2003) fit in this category.

Research through design seeks to provide an explanation or theory through action-reflection for use within a broader context (Frankel & Racine 2010). Downton (2003, 77) viewed this kind of investigative theory as explaining and becoming “a vehicle for acquiring and shaping knowing” that assists in future design activities. This category is derived from and is valuable for practice, and both practitioners and researchers have contributed significantly to the literature (Frankel & Racine 2010). Jonas’s (2007) action-reflection, Schön’s (1983) reflection in action, and research-through-design by Zimmerman, Forlizzi, & Evenson (2007) fall in this category.

Research about design is characterized by Buchanan (2007) as searching for “an explanation in the experience of designers and those who use products.” According to Cross (2007), this category addresses “the nature of design activity, design behaviour, and design cognition.” Much of the research about design considers design in a more holistic sense, discussing what design should be and methods to achieve this, as well as finding out what designers do and then developing and perhaps refining this (Downton 2003, 37). Design inquiry (Buchanan 2007) is an example of research about design. After comparing the three categories of design research with the aim of this study, I expected to mainly conduct research about design and research through design and only include research for design to a lesser extent (if at all).

To consider design research from a scientific perspective, Faste & Faste (2012) have proposed the idea that design research is not a “kind” of research; rather, research is always a “kind” of design (see figure 3.1) They have argued that design research is creative research and should be regarded as a subset of design practice at large. This approach resonated well with me and allowed me to think of my research as an activity that resided within the “practice” super-set. However, the nature of my research questions implied using rigorous scientific methods associated with the traditional “research” super-set with its known and replicable methods.

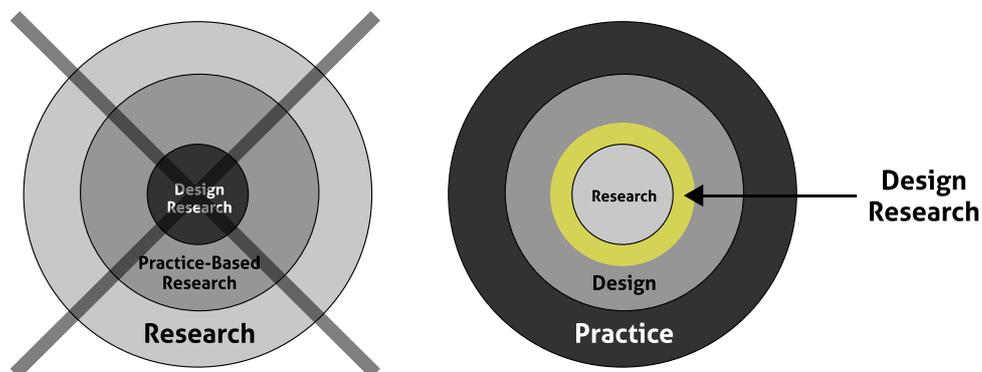


Figure 3.1: Design research as a subset of design practice at large (Faste & Faste 2012).

3.4.2 Understanding design research activities

To help me better understand the complex interrelatedness of my specific research questions and how they would cause me to move between practical, theoretical, and experimental, activities, I positioned my expected activities associated with each of my research questions within Fallman's (2008) Interaction Design Research Triangle. As the name implies, the model was initially developed to describe and guide research within the field of interaction design. However, the model’s general categories closely match the topics and context of this dissertation because “*interaction design, like all design disciplines, [...] resides in people, methods, processes, and artifacts*” (Fallman 2008, 9).

Fallman’s model provides a way of plotting the position of a design research activity within three

extremes: design practice, design studies, and design exploration, with the differences being primarily in tradition and perspective, rather than the methods and tools used.

Design practice denotes the types of activities undertaken outside academia, such as working for a commercial design organization, negotiating with clients, or working under budget constraints. This activity area is about engaging in real-life design practice but doing so with an appropriate research question in mind. The research question is developed and explored in either a reflective or proactive manner.

The category referred to as design studies indicates activities that most closely resemble traditional academic disciplines. This activity area is about building intellectual tradition within the discipline and contributing to the accumulated body of knowledge. It involves analytical work and taking part in ongoing discussions at conferences and workshops about design theory, methodology, and history.

Design exploration is similar to design practice but differs in one key point: it aims to explore "what if?" questions. This activity area is typically driven by the researcher's own research agenda, personal ideals, or theory seeking to transcend accepted paradigms. Design in this area becomes a statement of what is possible and is often intended to provoke or criticize.

The model also introduces three concepts to describe movement between activity areas: trajectories, loops, and dimensions. Trajectories are either intentional moves or unwanted drifting between two or more activity areas in the model. Loops are trajectories with no starting or ending points that move between different activity areas and are mainly used to describe a shift in the researcher's perspective rather than a change in actual practice. Dimensions describe a continuum between activity areas. Each activity area represents a perspective that indicates an infinite number of views. By choosing views from each activity area and placing them on a bipolar scale, a researcher can establish a frame in which a specific issue can be discussed.

Positioning my specific research questions (SRQs) in Fallman's Design Research Triangle (see figure 3.2) clarified that my research activities would mainly reside in design practice and design studies.

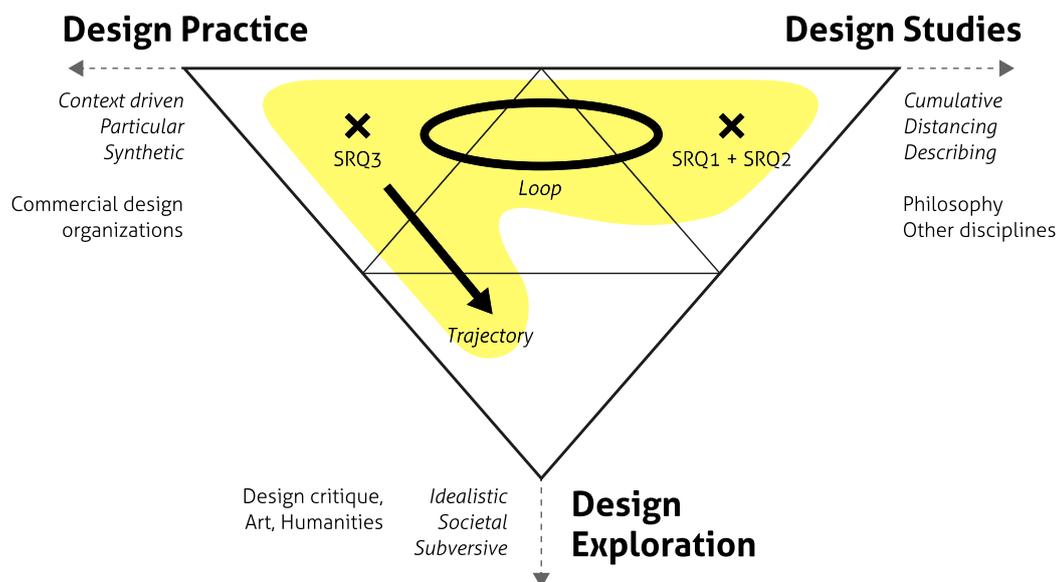


Figure 3.2: The study's specific research questions (marked by Xs) positioned in Fallman's (2008) Interaction Design Research Triangle. The yellow area shows the activity areas of this study. Also shown is the loop between the areas of design practice and design studies, as well as the trajectory pointing from design practice toward design exploration.

Design practice was appropriate because I was attempting to introduce programming as a design activity to be undertaken by graphic design students in their future professional vocation in the design industry. It would assume my active, first-hand engagement with students, which meant I would be part of the students' experiences as well as taking part in discussions, rather than being just an observer. My job as educator teaching Creative Coding naturally bestowed this upon me, but I would need to approach my teaching from another perspective—that of a researcher. Design studies was appropriate because I would have to use my personal experience, empirical observations, and data I gathered as a foundation for analytical work that would add to the body of knowledge of the graphic design discipline.

Given my background as practitioner-become-researcher, this study took its departure from design practice, but answering the research questions required establishing a loop between design practice and design studies. I expected a trajectory pointing from design practice toward design exploration to manifest itself through the discussion of the theoretical and practical implications of teaching programming to graphic designers.

3.4.3 Educational design research

Design research has also gained momentum in educational studies (Cobb et al. 2003; Akker et al. 2006). Specifically, design research within the field of educational studies aims at developing empirically grounded theories through combined study of both the process of learning and the means that support that process (DiSessa & Cobb 2004; Prediger, Gravemeijer, & Confrey 2015). Barab and Squire (2004, 2) define educational design research as *"a series of approaches, with the intent of producing new theories, artifacts, and practices that account for and potentially impact learning and teaching in naturalistic settings."*

While this study has only briefly touched upon the purpose of programming in graphic design education, it is, nonetheless, deeply concerned with the processes and practices related to how programming is taught in design schools, which ultimately will affect how students think about, conceive, and execute graphic design products. Taking a pragmatist's approach to this study, I have focused my attention on producing something that changes the world. My contribution will not be a self-contained Creative Coding course but rather a series of individual investigations, which, when viewed together, can guide and inform educators in their work to design, develop, and deliver Creative Coding courses. This process is generally referred to as instructional design.

3.4.4 Instructional design

Reiser and Dempsey (2007, 11) define instructional design as *"a systematic approach that is employed to develop education and training programs in a consistent and reliable fashion."* Practically, this entails determining the state and needs of the learner, defining the end goal of instruction, and creating some intervention to assist in the transition. But do investigations into instructional design constitute design research? Nelson (2013) has answered this question affirmatively, arguing that it may be useful to view instructional design as a design field in which applied and developmental research activities are carried out to develop grounded theories based in design practice. Also, reinforcing the relationship between instructional design and design research, Akker et al. (2006, 45) have argued that instructional design can be perceived as informal predecessors of design research.

Järvelä and Renninger (2014) have asserted that instructional designers must generate interest in learners, motivate them to learn, and create environments that can be engaging to the learners. These key functions can be greatly aided by the skilled application of visual design principles. Instructional design is also about stimulating the senses of the learner to create a positive emotional response to

the subject matter. Well-crafted instructional design can entice learners to complete the course, as well as inspire them to ask their own questions and explore the subject independently. Hokanson, Miller, and Hooper (2008) referred to the role of instructional designer as an instructional craftsman (i.e., someone who has honed his knowledge and skills through experience, and who is concerned with the technical and aesthetic aspects of a project). Furthermore, Christensen & West (2013) argued that an instructional craftsman refers to so much more than the ability to create an instructional product that simply looks good; the role becomes the manifestation of interdisciplinary design research. Accordingly, in this study, I took on the role of instructional craftsman, conducting educational design research in an interdisciplinary field with the pragmatic intent to improve how programming should be taught in graphic design education.

3.4.5 Mixed methods

My choice of the pragmatist paradigm negated a focus on any specific perspective or way of doing research; instead, it emphasized the specific problem or questions asked and used whatever data-collection methods were needed to understand the issue. Pragmatism advocates the use of any combination of methods—independent of their paradigmatic affiliation—that are qualified solely on their utilitarian value in investigating “what works” in order to arrive at an actionable result. This approach of bringing together several seemingly unrelated methods to form a coherent research strategy is commonly referred to as mixed methods research (MMR).

MMR has been practiced since the 1950s but formally emerged in the US in the late 1980s (McKim 2017) after the “paradigm wars” had ended (Gage 1989; Melles 2008, 5) and more dialogue between different methods emerged. MMR began occupying the middle ground between post-positivist and constructionist research paradigms. Recently, MMR has seen an explosion in theory development and it has taken off in several different directions (Creswell 2009, 2014; Creswell & Plano Clark 2007; Mertens 2007; Teddlie & Tashakkori 2009).

As a research method, MMR focuses on collecting, analyzing, and mixing both quantitative and qualitative data in a single study or series of studies (Hall 2013). Its central premise is that the combination of quantitative and qualitative approaches provides a better understanding and corroboration of research problems than either approach alone (Johnson, Onwuegbuzie, & Turner 2007, 123). Doyle, Brady, and Byrne (2016) discuss the main rationales or benefits proposed for undertaking an MMR study. An edited excerpt of these can be seen in Table 3.2.

Rationale	Explanation
<i>Triangulation</i>	Uses quantitative and qualitative methods so that findings may be mutually corroborated.
<i>Expansion</i>	Requires the findings of earlier phases to be explained qualitatively, if needed.
<i>Exploration</i>	Requires an initial phase to develop an instrument, intervention, identify variables to study or develop a hypothesis that requires testing.
<i>Completeness</i>	Provides a more comprehensive account of phenomena under study.
<i>Offset weaknesses</i>	Ensures that weaknesses of each method are minimized.
<i>Different research questions</i>	Allows for both quantitative and qualitative questions to be posed at the beginning of the study.
<i>Illustration</i>	Uses qualitative data to illuminate quantitative findings.

Table 3.2: Main rationales and benefits of Mixed Methods Research (adapted from Doyle, Brady, and Byrne 2016).

I arrived at my decision to adopt MMR through my choice of pragmatism as my paradigmatic position. Other researchers, however, have taken the opposite direction. Hall (2013) investigated the use of mixed methods while looking for an appropriate paradigm to justify its use. Hall (2013) identified three approaches to paradigm choice: the a-paradigmatic approach, the multiple paradigm approach, and the single paradigm approach. He concluded that a single paradigm can indeed provide a justification for mixed methods. Despite acknowledging pragmatism as a contender, he ultimately pointed to realism as a better choice of paradigm for mixed methods, criticizing pragmatism's lack of clear definition of "what works." Hall opposed Goldkuhl (2011), who claimed that design research had found an appropriate home in the pragmatist paradigm. His view aligned with that of Melles (2008), who posited that acknowledging pragmatism as useful and relevant within design research, would give purpose to the mixed methods approach that design currently employs on an ad-hoc, eclectic basis (Melles 2008, 10). In light of my findings, discussed in this chapter, which all indicated a strong natural relationship between pragmatism and the educational and design parts of my research topic, I rejected Hall's recommendation in favor of the views of Melles and Goldkuhl and maintained my position as pragmatist.

Collectively, the topics discussed in the previous sections were all indicative of and supported mixed methods research as the most appropriate and best suited to provide an exhaustive, broad, and nuanced answer to my overarching research question.

3.5 Research design

In section 1.7, the study's overarching research question, "*how should programming ideally be taught to graphic designers to account for how they learn and how they intend to integrate programming into their vocational practice?*" was subdivided into three specific research questions to be addressed in individual papers. Given the methodological flexibility of MMR, coupled with my paradigmatic pragmatist position, I chose to let the nature of each individual research question decide which particular qualitative or quantitative method was appropriate. Below, I account for the research design of my three papers individually.

3.5.1 Paper 1

This paper seeks to answer SRQ1: "*How is Creative Coding currently taught in graphic design schools?*"

The objective of the study on which the paper is based was to survey the structure and content of contemporary Creative Coding courses to establish a snapshot of current teaching praxis. Its underlying hypothesis, derived from my personal experience and observations, was that contemporary Creative Coding courses are mainly taught using a computer science approach that is not beneficial to graphic design students.

The paper describes research that analyzed syllabi collected from contemporary Creative Coding courses taught at design schools across the globe. The reason for this choice of material was that it would be practically impossible to do in-depth observation studies of several courses occurring over extended periods of time in many different places. I could have opted to do in-class observation studies, but at the expense of the breadth of the study. However, there are benefits of looking at finished courses: 1) they are typically well-documented, 2) teaching materials are available, 3) course assignments and exercises, plus the students' submissions to these, are available, and 4) it allows the study to document "what actually happened" as opposed to the course educators' intentions of "what will happen."

To provide a broad and nuanced overview of the structure and content in contemporary Creative Coding classes, I chose breadth over depth.

Given the nature of the data collected, I performed a quantitative study of constituent course parameters and discussed the results in juxtaposition with traditional graphic design education to evaluate the paper's hypothesis.

3.5.2 Paper 2

The intent of this paper is to answer SRQ2: *"How should Creative Coding be taught to accommodate how graphic design students learn?"*

The objective of the study presented in this paper was to assess if the instructional design of how Creative Coding is currently taught is optimal with respect to how graphic designers learn. Its underlying hypothesis, based on the observations gained from the study described in paper 1, was that contemporary Creative Coding courses are not taught in the most optimal way and do not account for how graphic designers learn.

To evaluate the hypothesis, the study set out to investigate how graphic design students learn. Specifically, the study sought to understand how graphic design students compare with students in other disciplines to determine characteristics that educators must account for. The study used a standardized test to allow for direct comparison of the results with findings obtained in similar tests of different audiences performed by other researchers. As my literature review revealed, the Felder-Soloman Learning Style Index (ILS[®]) online questionnaire (Felder & Soloman 1997) appeared to be widely used and frequently referred to in publications across many fields. Accordingly, in this study, I administered the Felder-Soloman ILS[®] questionnaire to groups of graphic design students to establish their learning style profile. I compared my results with groups of students from technical disciplines to determine if graphic design students exhibit different characteristics. Finally, I compared the learning style profile with my findings from paper 1 to identify any measures that should be addressed to adapt the instructional design of Creative Coding courses to suit the learning styles of graphic design students.

To ensure a level of homogeneity and consistency in the data used in the analysis, I opted to test students from just one school (The Danish School of Media and Journalism). A similar choice had been made by the researchers whose results I used for comparison.

3.5.3 Paper 3

This paper addresses SRQ3: *"How can graphic design students be motivated and supported as they are introduced to programming?"*

The objective of the study behind the paper was to describe and test a new pedagogical method for teaching programming to graphic designers. The pedagogical method was developed using constructionist learning theory as its underpinning theoretical framework. Following Papert's (1980) idea of using familiar knowledge in one domain to leverage knowledge acquisition from an unfamiliar domain, the pedagogical method relied on pre-existing graphic design specimens to contextualize computational concepts.

The underlying hypothesis of the study was that contextualizing programming to fit the students' predominant learning style and pre-existing domain-specific skills would improve the motivation and desire to learn programming of graphic design students, which would be essential to pave the way for their self-initiated exploration of code in their vocational practice.

The study took an empirical action research approach. I employed and tested the pedagogical method in real Creative Coding courses to 1) make first-hand empirical observations, and 2) receive qualitative feedback from the students as they described their experience of working with the method. This strategy was chosen as a direct consequence of the first two papers having used quantitative methods and not engaged directly with the intended audience. A key rationale in MMR, explained in section 3.4.5, is that combining methods is believed to lead to a better understanding and a more comprehensive account of the studied phenomenon than could be arrived at by either approach alone.

The paper describes the pedagogical method, explains its intended use in Creative Coding courses, and assesses the potential, successes, and shortcomings of the pedagogical methods based on empirical observations and plenary student interviews.

3.5.4 Summary

The research design chosen for each of the papers is summarized in table 3.3.

Specific Research Question	Objective	Hypothesis	Research Type	Research Method	Material
SRQ1: How is Creative Coding currently taught in graphic design schools?	Survey the structure and content of contemporary Creative Coding courses to establish a snapshot of current teaching praxis	Contemporary Creative Coding courses are mainly taught using a computer science approach which is not beneficial to graphic design students	Research about design Developmental research (Brown, Gough, Roddis) Design studies (Fallman)	Scientific approach Quantitative study of structure and content in contemporary Creative Coding syllabi	Syllabi from contemporary introductory Creative Coding courses taught at design schools
SRQ2: How should Creative Coding be taught to accommodate how graphic designers learn?	Assess if the instructional design of how Creative Coding is currently taught is optimal with respect to how graphic designers learn	Contemporary Creative Coding courses are not taught in the most optimal way and do not account for how graphic designers learn	Research about design Developmental research (Brown, Gough, Roddis) Design studies (Fallman)	Scientific approach Quantitative Learning Style test to graphic design students	Felder-Soloman Learning Style Index (ILS®) tests taken by graphic design students
SRQ3: How can graphic design students be motivated and supported as they are introduced to programming?	Suggest and test one possible pedagogic method to assess its potential, successes and shortcomings	Contextualizing programming to fit the students' predominant learning style and pre-existing domain-specific skills, will improve the motivation of graphic design students and their desire to learn programming	Research through design Applied research (Brown, Gough, Roddis) Design practice (Fallman)	Design approach In-situ testing of the pedagogic method on its intended audience Qualitative assessment of its efficiency	Iteratively improved proprietary pedagogic method Graphic design specimens collected using Pinterest Instructional material

Table 3.3: Summary of research design chosen to answer each SRQ.

3.6 Order of execution

Gathering data for paper 1 was initiated immediately. Compared to papers 2 and 3, gathering data could be carried out in-between other activities. Once sufficient data was gathered, analysis could commence immediately as this study was not dependent on input from papers 2 and 3.

To obtain a sufficient sample size, gathering data for paper 2 also had to be initiated immediately. My choice of surveying students from only one school (section 3.5.2) had the drawback that data could only be collected across a larger timespan because each class was fairly small (approx. 24 students).

Paper 3 describes a pedagogic model and observations on its employment in a Creative Coding course. As the course is taught two times a year, across the span of my PhD, I would teach the course approximately six times. This provided an opportunity to iteratively improve the pedagogic method using feedback and knowledge obtained from the preceding courses. Consequently, I initiated development on the pedagogic method immediately, giving me the chance put the method to the test and report my observations before having to turn my attention to papers 1 and 2. This, however, means that only findings from the first iterations of the method have been reported in paper 3.

Taking all of the above-mentioned elements into account, the order in which to execute the activities described in each paper emerged. Figure 3.1 provides a schematic overview:

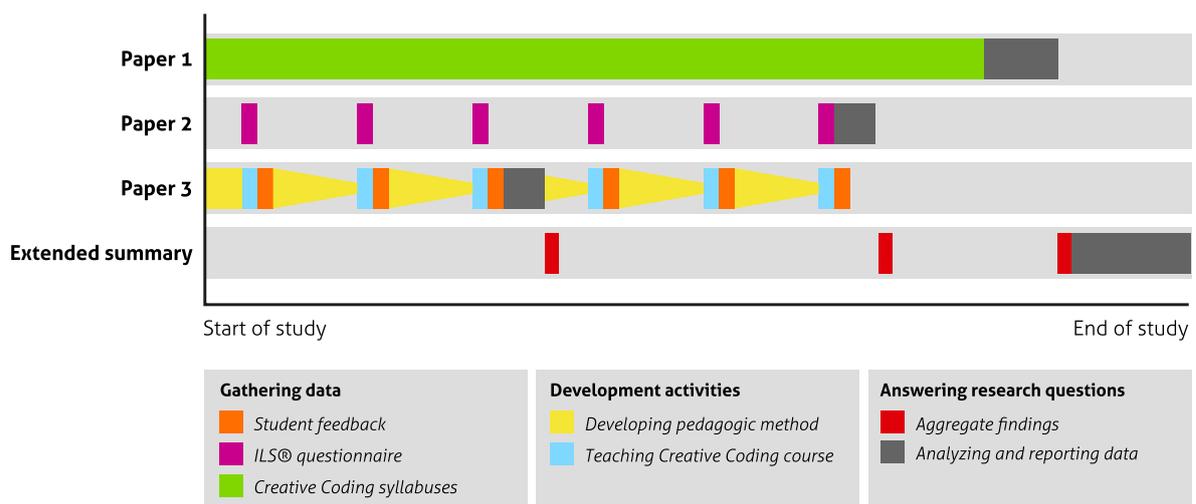


Figure 3.3: Schematic overview of research activities in this study.

3.7 Validating results

As this is a paper-based dissertation, results have been continuously evaluated in multiple peer reviews, and revisions have been made according to the feedback (see the submission and publication history at the beginning of each paper). However, acknowledging that some readers wish to critically scrutinize my results, I will recommend some possible ways to validate the results of my research.

Results obtained in paper 1 (chapter 5) can be validated by replicating the method and analysis described in the paper. A comprehensive list of the courses that make up the dataset is described in the paper. Teaching materials, syllabi, and student assignments for each course can be located and downloaded using one of the Internet search engines mentioned in the paper. As new Creative Coding courses are constantly added and indexed by Internet search engines, following the search

strategy described in the paper will always yield a new set of courses to be analyzed. However, given my decision to opt for breadth over depth, I feel assured that any attempts to replicate my study in the near future using a set of different courses will likely see a re-emergence of the reported pattern, thus confirming my results.

Results in paper 2 (chapter 6) are easily validated due to my choice of using a standardized and widely used profiling tool (Felder & Soloman 1997). The paper lists the exact data obtained through the questionnaire, thereby making it easy to subject my results to various validation checks. I speculate that attempts to validate my results on a similar audience will arrive at the same observations.

Validating the usefulness of the method described in paper 3 (chapter 7) requires practically implementing, observing, and assessing the method by following the outlines described in the paper. Student feedback was gathered using a qualitative approach, thus not permitting a direct quantitative comparison of the performance of the suggested method to other similar methods. As of now, the verification of the method's efficiency is based on my past experience with other methods, students' feedback of their perceived experience working with the method, and an assessment of the effect of the iterative improvements made to the method. As described in section 8.6, future studies are planned to formally qualify the method as a valid and effective approach. Additionally, I hope that other educators will incorporate the method in their teaching and report their observations regarding its usefulness.

The overall research question posed in this dissertation is answered by merging results found in papers 1–3. Given the open-ended nature of the research question, it is important to stress that the answer provided in section 8.2.4 should be seen as just one of many possible conclusions. Following my personal ambition to also provide a utilitarian output, the answer is interpreted and supported by a set of heuristics. I suggest that the usefulness of the heuristics be evaluated through the eyes of a design educator. In the text accompanying each heuristic, I provide a detailed explanation anchored in supporting studies, findings from my own papers, and my own empirical observations. This allows the reader to make a personal assessment of the validity of the suggested heuristics.

CHAPTER 4: RESEARCH CONTRIBUTIONS

This chapter provides an overview of the research papers and discusses their interrelationship. Each paper is presented in the following chapters along with a submission history and publication state.

4.1 Overview

“Mapping Creative Coding Courses: Towards Bespoke Programming Curricula in Graphic Design Education” (paper 1; chapter 5) reports a study of thirty 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 paper 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.

“Assessing Graphic Designers’ Learning Style Profile to Improve Creative Coding Course” (paper 2; chapter 6) assesses the learning style of graphic design students to help design school educators teaching Creative Coding programming courses adapt their teaching style to account for the way their students learn. The Felder-Soloman Index of Learning Styles (ILS[®]) was administered to 77 bachelor-level graphic design students. Compared to students in technical fields, the graphic design students differed by being considerably more intuitive, with an increased preference for active and visual learning. Based on these findings, specific recommendations and issues for educators to consider are presented.

“Deconstruction/Reconstruction: A Pedagogic Method for Teaching Programming to Graphic Designers” (paper 3; chapter 7) proposes, describes, and exemplifies a hands-on, experiential pedagogic method, *deconstruction/reconstruction*, specifically designed to introduce graphic design students to programming in a visual context. The method uses pre-existing, commercially applied graphic design specimens as its main material to contextualize programming into a domain familiar to the audience. Observations of the method used in teaching are discussed, and its potential is evaluated based on feedback provided by the students.

4.2 Linking the papers

The papers share a common origin in the dissertation’s overarching research question (RQ; section 1.7). Each paper takes a specific research question (SRQs; section 1.7) as its point of departure.

Individually, the papers are self-contained pieces that contribute valuable knowledge by posing and answering highly specific questions. Seen together, the papers provide a coherent and comprehensive contribution to research on instructional design in the interdisciplinary field of graphic design and programming.

Further linking the papers are their mutual temporalities, deliberately designed to give the dissertation a broad chronological span by covering both past, present, and future (see table 4.1).

Paper	Purpose	Temporality	Chronology
Paper 1	Analyze syllabi from recently finished programming courses in design schools	Past > Examine what has been done	↓
Paper 2	Profile how graphic designers learn to inform how programming courses should be designed	Present > Understand current potential	
Paper 3	Suggest and test a bespoke contextualized pedagogic method as one possible way to teach programming	Future > Suggest what can be done	

Table 4.1: Progression of temporalities in papers.

The linear temporal succession of the papers suggests that they should be read in the sequence they appear in this dissertation. However, due to issues accounted for in section 3.6, the papers were originally written in a different order.

In the final conclusion of the dissertation (section 8.2) each paper’s contribution to answering the dissertation’s overarching question is discussed in more detail.

CHAPTER 5: PAPER 1

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

Submission history	Accepted	Publication State
ITiCSE 2018 (Poster) SIGCSE 2019 (Paper) EuroGraphics 2019 (Paper)	ITiCSE 2018 (Poster) EuroGraphics 2019 (Paper)	Published (Poster) Published (Paper)

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.

Keywords

Curriculum, Syllabus, Graphic Design, Creative Coding, Processing

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.

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.

Next, the content of each of the courses on the gross list was reviewed and held 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.

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.

Computer Science Topics																	Graphic Design Topics																			IDE												
IDE intro	Comments	Console / Print Statements	Debugging	Flow Control	Syntax & Reference	Variables	Operators & Expressions	Mouse	Keyboard	Arrays	Conditionals	Loops	Functions	OOP	Timers	Events	GUI	Data Import	Data Export	Touch (Mobile Devices)	APTs	Recursion	Network	Hardware & Electronics	Browser DOM	Libraries (External)	Color	Graphic Primitives	Custom Shapes	Coordinate System	Randomness / Noise	Transformations	Text	Images	Motion & Animation	Typography	Video	Sound	3D	Computer Vision	Pixel operations	Collisions	Vectors	Mathematics	Computational Aesthetics	Processing	p5.js	javascript (pure / frameworks)
2	3	3	6	3	2	2	3	2	2	7	3	4	5	8	2			17			17	25		18			2	2	7	2	5	6	11	10	5		23					10	14	17		2	7	
2		5		3	2	6	4	4	4	12	6	6	8	10				18								20	12	2	2	4	14	18	16	26		16	22	14					2	14		2	16	
2	2			2	2	2	10			3	3	2								10								4	2	3	2	6	5	11	12	5						13	11			3	7	2
1	1	12		4	1	10	4	1		16	1	10	10	19			7				40	31		42			4	1	6	4	16	13	16	22	25				31			22			12		1	
2	3			3	2	8	8	11	11	19	9	9	15	17		11		23									5	5	5	5			13	13	13	21	21									2		
1	2	2		2	2	2	2	3	3	5	2	2	6	6										7		8		2	3	2	6	3	3	5			3	5		3					6		1	
4			4							6	5	5		6							10						8	4		4	4			8			9	8	8			12			10	4		
1		6			2	2	6	6		2	2	2						8									3	3	2	10			3	9	4			13						5		1		
1	2	3		2	2	3	2	2	7	3	4	5	9				15				16						1	1	6	2	3	13	19	18	13							18			6		1	
1	2	2	2	1	2	2	2	6	7	8	3	4	6	11	6	7	15	14	14							15	2	2	4	2	6	12	5	5	5	5		16	17		13		18	6		1		
1	2	3	3	1	2	2	3	10	5	3	2	5	6	12	8		11			11		10				8	1	1	13	1	2	5	7	3	5					7			13			1		
1	1	1	2	1	2	3	2	3	6	2	3	7	6	5		5	10			11		12				4	10	1	1	1	5	3	1	5	5		5	5						5			1	
1			2	1	2	3	2	6	5	3	3	4	5	11	6	6	7			10	7	4	12			6	12	1	1	1	2	2	7	5	3		8	8	11		11	5				1		
1	2		1	4	2	3	5	5	4	3	4	5	6	5		7		7								1	1	2	2	3	4	3	7	5	2		7	6	7	7			6	4	7		1	
1	1	2		2	1	2	3	4	5	3	4	5	8			3											1	3	1	3	3	4	4		4	4	2						8			1		
1		6	1	2	2	2	3	4	5	2	2	8		7	3	5	6									10	2	2	2	4	7	6	4	3	4			7				4		6		1		
2			13	2	2	3	4	2		8	4	4	5	6				23	23			14				27	13	1	1	16	1	3	16	21	18		21	19	25	16	20	18			14		1	
3	3	6	4	3	3	5	7	5	7	9	7	8	10	11	12	12	7									13	4	4		4	6	4		13	5		13	13	14					12		3		3
1				2	2	3				3	2	4	8					10			10						1	1			4	4	4		4	4	5									1	9	
1	1	2	2	1	2	2	2	3	4	3	4	6	6							9	5						1	1	1	2	8	4	3	2	4	5	5	9	9								1	
1	2	3	1	1	2	1	2	2	5	2	2	3	8	4	8		11			8		12	11	7			1	1	7	1	2	4	9	3		3	6	6	11		6	6	11	4			1	12
1			5	2	1	2	2		4	2	2	3	12					9	10								6	1						9	8			8	10	7		8			7		1	
1		15	20	1	1	4	3	8	8	12	9	5	10	19			8			17	17					31	3	2	3	2	6	11	21	17		21	28	31	32		26	8	16	14		1		
1	1	1		1	1	2	4	9	10	6	4	4	5	5	8	10	10	6	3		6					10	3	1	2	1	4	2	5	10	8	5		12	11	13	10			4		1	5	5
1	1	1	3	2	1	2	2	3		4	2	2	4	11		8											1	1	1	2			5	3											1		1	1
1			3	1	2	1				4	3	4	3	6		3											2	2	2																		1	
2	2		2	2	2	3	3	4	5	3	5	11	11	7		8										7	12	2	2	2	6		6	5												2		
2	2		2	2	2	3	3	3	5	3	3	4	7				9										10			4		9	5	4												4		1
1	1	3		1	1	3	3	1		3	3	3	4	4		6		6		6								1	1	2	1	3	2		5	2	5	7	5						2		1	
2			2	2						7	5	6	6	7				19										2	2		2	15														2		9

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

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

Computer Science 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 (3 rd 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%
Computational Aesthetics	55%
Video	56%
Pixel Operations	57%
Sound	66%
Vectors	70%
3D	72%
Computer Vision	82%

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

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 educator 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.

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 educator'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, automata, 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 educators 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

Appendix: List of analyzed courses

Course Name	Institution	Country	Semester & Year
DM-UY-1133-A Creative Coding	Integrated Digital Media, NYU Tandon School of Engineering	US	Spring 2017
DM-UY-1133-C Creative Coding	Integrated Digital Media, NYU Tandon School of Engineering	US	Fall 2017
Programming Design Systems (previously "Printing Code")	NYU Tisch School of the Arts	US	2016
15-104 • Computation for Creative Practices	Carnegie Mellon University	US	2016
MCC-UE 1585 Creative Coding	NYU Steinhardt	US	2014
CCT 126 Programming for Artists and Designers	Maine College of Art	US	Winter 2013
VA345 Creative Coding	Sabanci University	TR	2017
VCD 293 Design By Code	Istanbul Bilgi Üniversitesi	TR	2016-2017
CS 110 (Sect. 2): Introduction to Computing	Bryn Mawr College	US	Spring 2016
EDPX 2100 Coding	University of Denver	US	Winter 2016
PUCD 2035-E Creative Computing	Parsons The New School for Design	US	Spring 2016
DAT405 Creative Coding	University of Plymouth	GB	Fall 2017
ICM-2017 ITP Foundation Course to Computational Media	NYU Tisch School of the Arts	US	2017
Corso di Computer Graphics	Accademia di Belle Arti di Brera	IT	2018
HCDE 598 MS Creative Computing	University of Washington	US	Winter 2017
Creative Coding 1701ICT	Griffith University	AU	2017
AET 319 Foundations of Creative Coding	University of Texas at Austin	US	Fall 2016
MART 120 Creative Coding 1	University of Montana	US	2017
Kickstart Algorithmic Thinking & Programming	Lucerne University of Applied Sciences and Arts	CH	Spring 2015
Programming for Visual Artists	Aalto University	FI	Spring 2018
Programming for Visual Artists	Purchase College State University of New York	US	Fall 2017
Programming for Artists	University of Florida	US	Spring 2016
Computer Science 1050: Introduction to Computer Science: Multimedia	Saint Louis University	US	Spring 2016
ARTS 249-01 Creative Coding	Queens College	US	Spring 2017
PUCD 2035-E Creative Computing	Parsons The New School for Design	US	Fall 2015
ASIM 1310 Art + Code	Southern Methodist University	US	Fall 2013
CIM 540 Intro to Creative Coding	University of Miami School of Communication	US	Spring 2017
CAT 117 Process & Interaction: An Introduction to Creative Coding	Bloomfield College	US	Spring 2015
Creative Coding	Politecnico di Milano	IT	Fall 2018
Creative Coding	Integrated Digital Media, NYU Tandon School of Engineering	US	Fall 2018

CHAPTER 6: PAPER 2

Assessing Graphic Designers' Learning Style Profile to Improve Creative Coding Courses

Submission history	Accepted	Publication State
EuroGraphics 2019 (Paper)	EuroGraphics 2019 (Paper)	Published (Paper)

Abstract

This study aimed at assessing the learning style of graphic design students to help design school educators teaching Creative Coding programming courses adapt their teaching style to account for the way their students learn. The Felder-Soloman Index of Learning Styles (ILS©) was administered to 77 bachelor-level graphic design students. Compared to students in technical fields, the graphic design students differed by being considerably more intuitive, with an increased preference for active and visual learning. Based on these findings, specific recommendations and issues for educators to consider are presented.

1. Introduction

In today's software-driven, techno-centric world, the popular prevailing discourse is that everyone must learn to program. Many national and international initiatives have helped put coding and computational thinking into schools' curriculums, aided by a rapidly increasing undergrowth of dedicated programming environments tailored to suit the needs and learning situations of specific audiences. Programming has also made its entrance into Graphic Design education. Design schools across the globe are now offering programming courses, typically branded using the popularized term Creative Coding. These courses teach informal programming practices that enable graphic design students to create expressive visual output for use in commercial contexts.

However, being a recent addition to graphic design education, programming has not yet been taught extensively. Hence, empirically gained knowledge to help design educators navigate and operate within the intersection among graphic design, programming, and teaching is largely missing. As a contribution to close this gap, this study will assess the learning style profile of graphic designers. Learning styles are different and unique ways used by students as they prepare to learn and recall information. Incorporating learning styles into teaching plans can make learning easier and lead to better achievement. Conversely, failing to match the students' preferred learning styles risks impeding their learning. While several studies have been undertaken to assess the learning style preferences of students in many diverse disciplines, no known studies have explicitly sought to profile graphic design students. Understanding the preferred learning style of graphic design students will help design educators plan and execute enjoyable, enriching, and effective learning

experiences that teach programming in a way which accounts for how graphic design students prefer to acquire new knowledge.

2. Method

Although some researchers consider the idea of learning styles a contested notion [FB05, p.58], more than 70 learning style models are described [CMHE04] with Kolb [Kol84], Honey and Mumford [HM92], Myers-Briggs [BMQH98], and Felder and Silverman [FS88] being the most commonly used. Each proposes different classifications and descriptions of learning styles.

This paper uses the Felder-Silverman learning style model (FSLSM) developed in 1988 [FS88] and updated in 2002. FSLSM is widely used in scholarly literature within science, technology, engineering, and mathematics (STEM) fields; thus, there is a large pool of studies to compare the findings against. The learning styles defined in FSLSM can be identified using the Index of Learning Styles (ILS©) questionnaire [FS00]. ILS© is an often used and well-investigated instrument generally considered reliable across disciplines [FG07, ZWA00, Zyw03]. ILS© is available as a free online test, which makes it easy for educators to deploy and interpret. This also allows researchers to verify and extend the results reported in this paper. While other studies on how graphic designers learn focus on qualitative, holistic, and procedural aspects [Cro82, Law05, Sch83], the combination of FSLSM used in conjunction with ILS© provides a quantitative, utilitarian lens through which to consider the ways students prefer to acquire knowledge.

Compared to other learning style models, which tend to classify learners into a few groups, FSLSM allows for a more nuanced profile by placing the learner on a scale between two contrasting poles across four dimensions. Each dimension can be summarized as follows [FS05]:

- *"sensing (concrete, practical, oriented toward facts and procedures) or intuitive (conceptual, innovative, oriented toward theories and underlying meanings);*
- *visual (prefer visual representations of presented material, such as pictures, diagrams, and flow charts) or verbal (prefer written and spoken explanations);*
- *active (learn by trying things out, enjoy working in groups) or reflective (learn by thinking things through, prefer working alone or with one or two familiar partners);*
- *sequential (linear thinking process, learn in incremental steps) or global (holistic thinking process)."*

Moreover, FSLSM is based on tendencies, indicating that learners, despite exhibiting a preference for a certain behaviour, can sometimes act differently.

3. Study

The study was conducted between May 2015 and May 2018. A total of 77 bachelor-level graphic design students participated: 41 males and 36 females, with ages varying between 19 and 35 years (median 25 years). All students were enrolled in introductory Creative Coding classes at The Danish School of Media and Journalism (DMJX). The study and the purpose were explained to the students, who were then asked to complete the ILS© online questionnaire and submit the results. Upon completion, students were briefed about the ILS© learning modalities to allow them to make use of their test scores.

The data collected was entered into Microsoft Excel and analyzed according to instructions given in [FS05, p.105]. Statistical analysis against the chosen comparison studies was not possible due to lack of exact data provided.

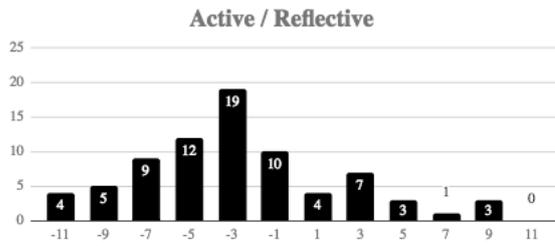


Figure 1: Active-Reflective distribution of the respondents.

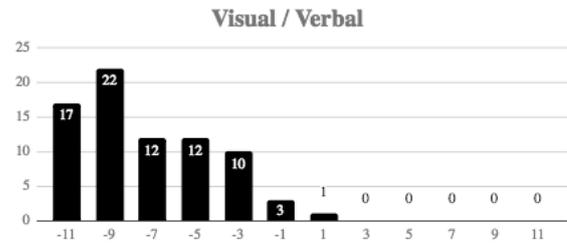


Figure 2: Visual-Verbal distribution of the respondents.

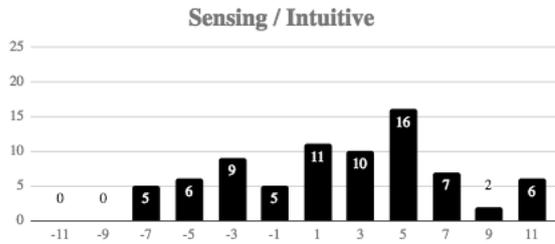


Figure 3: Sensing-Intuitive distribution of the respondents.

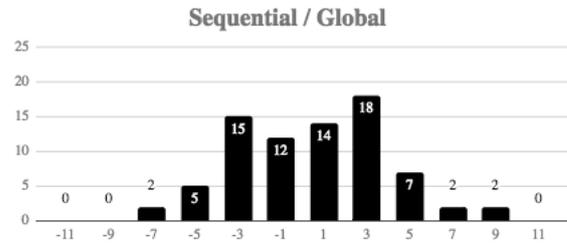


Figure 4: Sequential-Global distribution of the respondents.

4. Results

Figures 1–4 depict the accumulated results of all four dimensions. The left side of each dimension pair is presented as a negative value, and the right side is positive, the encoding being from -11 to +11 in odd numbers. Following the encoding procedure described in [FS05], Table 1 lists students' cumulative results arranged according to the strength of their preference as either strong (± 11 , ± 9), moderate (± 7 , ± 5) or mild (± 3 , ± 1). Furthermore, results are also expressed percentage-wise in Table 2, row A, to make them comparable with results reported in other studies.

The 77 graphic design students who participated in the study were characterized by a majority of highly visual, active, intuitive types with a fairly balanced number of sequential and global learners. Remarkably, looking at the VIS/VRB dimension, 82% of the students were moderate or strong visual learners. A majority of visual learners was anticipated as most people prefer to learn this way, but for students enrolled in a design school, the ratio was hypothesized to be more pronounced, and indeed, only one student showed a mild preference toward verbal instruction. In dimension SEQ/GLO, students were more diverse. Median value resided at 1, indicating that most students had a mild preference for both. A quarter of the students (24%) showed a moderate to strong preference towards either sequential (9%) or global (15%) learning. Similarly, in dimension ACT/REF, students exhibited a wide spread in

Active / Reflective	Moderate-Strong Active	39%
	Mild	52%
	Moderate-Strong Reflective	9%
Sensing / Intuitive	Moderate-Strong Sensing	14%
	Mild	46%
	Moderate-Strong Intuitive	40%
Visual / Verbal	Moderate-Strong Visual	82%
	Mild	18%
	Moderate-Strong Verbal	0%
Sequential / Global	Moderate-Strong Sequential	9%
	Mild	76%
	Moderate-Strong Global	15%

Table 1: Strengths of preferences.

their preferred learning style. The median value was -3, suggesting that the majority of students (52%) had a mild preference toward active learning. Of students showing a moderate to strong preference in this dimension, more were active (39%) than reflective (9%). On the SEN/INT dimension, nearly half the students (46%) had a mild preference for either type, with an almost equal number of students (40%) having a moderate to strong preference toward intuitive learning.

4.1 Results compared to other disciplines

To put the results into perspective, a number of studies reporting ILS© scores across different disciplines (Table 2, rows B-H) have been used as a comparison. The studies have been selected with a desire to eliminate any skewing in the results relating to geocultural differences in teaching and learning styles.

A cross all four FSLSM dimensions, results in this study correspond well with results obtained by Kolmos & Holgaard [KH08], who examined, among others, students studying Architecture & Design. This is considered indicative of a correlation between the disciplinary kinship and the students' learning profile. A considerable difference is found in dimension SEN/INT, where students in design-related studies A and B are mostly intuitive learners, directly opposed to students in technical-related studies C-G, who are mostly sensing learners. On the VIS/VRB dimension, graphic designers are generally more visual learners than students in technical fields. In fact, this study represents the highest percentage of visual learners compared to other known studies using ILS©. However, the informatics engineering students in study G show a similarly high preference toward visual learning, thereby debunking the idea that studies situated within the field of Art and Design will implicitly have a significantly larger population of visual learners in a cohort of students. Compared to the many (mostly technically oriented) studies summarized in study H, graphic design students at DMJX are almost exclusively visual (99% vs. 82%), global rather than sequential (44% vs. 60%), intuitive rather than sensing (32% vs 63%), and increasingly active (77% vs 64%) learners.

5. Implications for educators

Assuming that the learning style profile of the graphic design students at DMJX is representative of graphic design students in general, several insights can be gained from interpreting the results using the updated Felder & Silverman teaching style model [FS88] and Felder & Soloman learning styles and strategies [FS00]. In the following paragraphs, these insights have been converted into specific recommendations to inform educators and help them plan Creative Coding courses aimed at graphic design students.

Field	Institution	Act	Sen	Vis	Seq	N	Ref
A Graphic Design	The Danish School of Media and Journalism	77%	32%	99%	44%	77	This study
B Architecture & Design	Aalborg University	79%	38%	96%	32%	77	[KH08]
C Computer Engineering & Science	Aalborg University	71%	69%	81%	47%	70	[KH08]
D Mathematics	Aalborg University	50%	71%	79%	57%	14	[KH08]
E Computer Science	Lappeenranta University of Technology	62%	69%	73%	41%	118	[AS10]
F Information Systems	Massey University & Vienna University of Technology	57%	58%	87%	56%	207	[GVLK07]
G Informatics Engineering	Polytechnic Institute of Coimbra	64%	61%	96%	74%	173	[GM10]
H Multiple Fields	Multiple Institutions	64%	63%	82%	60%	2506	[FS05]

Table 2: Learning Style preferences found in this study compared to those reported in similar studies.

An overwhelming majority of students will have moderate to strong visual preferences. This emphasizes a need for teaching materials, demonstrations, and assignments to be highly visual. Students will not respond well to verbal instructions (i.e., passive auditorium lectures). To help students develop mental models of abstract programmatic constructs, they must be supported by visualizations [Pan16] or metaphors drawn from their pre-existing domain-specific knowledge (e.g., using nested for-loops to generate a 2D grid of shapes). Working processes should be presented whenever possible. Live coding is one particularly useful way to accomplish this that holds many benefits [BW18]. Another way to make teaching more visual is by incorporating premade interactive and editable code examples for the students to explore. This will give students an opportunity to learn programming by forming and testing ideas through immediate visual feedback.

Most students will have a mild sensing/intuitive preference, with an almost equal number of students being strong-moderate intuitive learners. Still, a sizable minority of students have sensing preferences and must be considered. It is essential that both types be catered to and that corresponding measures be taken when designing the course material. Educators should alternate between instructional methods best suited for each type, or, alternatively, introduce two parallel tracks in both teaching and assignments. For example, assignments could be designed to have a fixed goal but allow for two different ways of arriving at a solution: either through experimentation and novel use of new techniques to accommodate the intuitors, or through stepwise instructions that incorporate the use of memorized knowledge to accommodate the sensors. The formal and structured nature of programming implies that students must be presented a certain number of facts, but such sessions should be kept at a minimum. Also, the students' general bias towards intuitive learners instills hope in the sense that they should be able to cope with the abstract and mathematical concepts within programming – worth addressing at the beginning of the course to help alleviate any premature code-induced anxiety among the students.

A majority of students having a mild sequential/global learning preference indicates that educators must prepare themselves to help both sequential students who learn in linear steps and global learners who learn in large jumps. Educators must be careful to provide the big picture and relate it to previous knowledge before diving into the details, without missing a step in their explanation. In graphic design education, programming is not an objective in itself; it is a means to achieve a higher purpose, namely that of crafting visual output. Therefore, educators must relate every programming concept to the broader context of the students' study and future vocation. Sequential learners might regard assignments and exercises as individual activities, whereas global learners must be reassured that the tasks they are asked to solve will eventually form a coherent body of knowledge and skills. Finally, it might be helpful to explain to global students that they should not be discouraged from feeling "in the dark" when they compare themselves to their sequential classmates – they are both making progress, but their learning takes place differently.

Active learning is preferred by most students; however, as this is only a mild preference, teaching initiatives that call for reflective activities should also be integrated. Pair-programming is suggested as a good teaching practice [BW18], but it might be transgressive to students who prefer to quietly reflect in order for learning to stick. The wide spread in the results suggests that students should be given the option to either work in pairs or work alone, depending on their personal preference. Active learners, who prefer to try things out and learn from experience, should be given objects to form a basis for their discovery. These objects might be inspirational visual material, premade code snippets, or a set of digital assets to use. To support reflective learners, educators should consider supplying additional explanatory tutorials and demonstrations, preferably as video/animations to cater to the students' visual preference. These could be viewed by students at their own pace as many times as

needed until they grasped the topic presented. Not only would this leave the educator free to attend to other tasks, it might also encourage students to persist in seeking an answer.

Further implications are suggested by Silverman [Sil02], co-developer of FLSM, who later extended her research based on brain research and clinical observations. Considering the results obtained in this study, the majority of graphic designers at DMJX fit Silverman's description of "Visual Spatial Learners": "*They learn better visually than auditorally. They learn all-at-once, and when the light bulb goes on, the learning is permanent. They do not learn from repetition and drill. They are whole-part learners who need to see the big picture first before they learn the details. They are non-sequential, which means that they do not learn in the step-by-step manner in which most teachers teach.*" [Sil02]

Silverman points out that visual-spatial abilities (associated with graphic design) are the domain of the right brain hemisphere; sequential abilities (associated with programming) are in the domain of the left brain hemisphere. Teaching programming to graphic designers, in other words, becomes a cross-hemispheric endeavor that requires educators to consider initiatives meant to access the left brain in addition to their regular mainly right-brain-oriented teaching activities. Suggested activities to stimulate the students' left brain hemispheres are verbal walk-throughs of algorithms, tests that involve math and logic, quizzes, and code-related puzzles (e.g., Parsons problems [PH06]).

6. Conclusions

The learning style profile of graphic design students at The Danish School of Media and Journalism (DMJX) differs noticeably from that of students in technical fields. Graphic design students have a more pronounced preference towards an intuitive learning style, they are virtually exclusively visual learners, and they more strongly prefer active learning. These findings suggest that courses developed to fit the learning style profile of students in technical fields will fail at matching the preferred learning style of graphic designers. This implicitly underlines the need to develop customized programming courses and accompanying instructional methods for use in design schools.

During the study, preliminary results continuously informed the instructional design of the course the profiled students were enrolled in, leading to the development of a pedagogic method specifically made to suit the learning style profile discussed in this paper. Experiences employing this method are reported in [Han17].

The size of the population tested (N=77) is sufficient to render the study valid for comparison with similar studies. However, to determine a broadly anchored learning style profile of graphic designers requires similar data to be collected by administering the ILS© to graphic design students in other design schools. This could be taken up as further research in this field.

References

- [AS10] ALAOUTINEN S. , SMOLANDER K.: Are computer science students different learners? In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research — Koli Calling '10* (2010), pp. 100–105
- [BMQH98] BRIGGS-MYERS I. , MCCAULLEY M. H. , QUENK N. L. , HAMMER, A. L.: *MBTI Manual: A Guide to the Development and Use of the Myers-Briggs Type Indicator*. 3rd Edition. Consulting Psychologists Press, 1998
- [BW18] BROWN N. C. C. , WILSON G.: Ten quick tips for teaching programming. In: *PLoS Computational Biology* (2018), pp. 1–8

- [CMHE04] COFFIELD F. , MOSELEY D. , HALL E. , ECCLESTONE K.: Learning styles and pedagogy in post-16 learning. A systematic and critical review. London, UK : Learning and Skills Research Centre, 2004
- [Cro82] CROSS N.: Designerly ways of knowing. In *Design Studies* vol. 3 (1982), Nr. 82, pp. 221–227
- [FB05] FELDER R. M. , BRENT R.: Understanding student differences. In *Journal of Engineering Education* vol. 94 (2005), Nr. 1, pp. 57–72
- [FG07] FELKEL B. H. , GOSKY R. M.: A study of reliability and validity of The Felder-Soloman Index of Learning Styles for Business Students. In *Proceedings from the International Conference on Technology in Collegiate Mathematics (ICTCM)*, (2007), pp. 38–47
- [FS88] FELDER R. M. , SILVERMAN L. K.: Learning and teaching styles in engineering education (June 2002 preface). In *Engineering Education* vol. 78 (1988), pp. 674–681
- [FS00] FELDER R. M. , SOLOMAN B. A.: Index of Learning Styles (ILS®) Questionnaire. <https://www.webtools.ncsu.edu/learningstyles/> - retrieved 2018-10-10. — North Carolina State University
- [FS05] FELDER R. M. , SPURLIN J.: Applications, reliability and validity of the Index of Learning Styles. In *International Journal of Engineering Education* vol. 21 (2005), Nr. 1, pp. 103–112
- [GM10] GOMES A. , MENDES A. J.: A Study on Student Performance in First Year CS Courses. In *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education - ITiCSE '10* (2010), pp. 113-117
- [GVLK07] GRAF S. , VIOLA S. R. , LEO T. , KINSHUK: In-depth analysis of the Felder-Silverman learning style dimensions. In *Journal of Research on Technology in Education* vol. 40 (2007), Nr. 1, pp. 79–93
- [Han17] HANSEN S. M.: Deconstruction/Reconstruction: A pedagogic method for teaching programming to graphic designers. In *Proceedings of the 20th Generative Art Conference GA2017* (2017)
- [HM92] HONEY P. , MUMFORD A.: *The Manual of Learning Styles*, 1992
- [KH08] KOLMOS A. , HOLGAARD J. E.: Learning styles of science and engineering students in problem and project based education. In *Proceedings of the 36th Annual SEFI Conference*, (2008)
- [Kol84] KOLB D. A.: *Experiential Learning: Experience as the Source of Learning and Development*, 1984
- [Law05] LAWSON B.: *How Designers Think — The Design Process Demystified*. 4th Edition. Oxford, England : Architectural Press, 2005
- [PH06] PARSONS D. , HADEN P.: Parson’s programming puzzles: A fun and effective learning tool for first programming courses. In *Conferences in Research and Practice in Information Technology Series* vol. 52 (2006), Nr. January 2006, pp. 157–163
- [Pan16] PANDA P.: *Helping Designers Understand Code*. College of Design, North Carolina State University (2016)
- [Sch83] SCHON D. A.: *The Reflective Practitioner: How Professionals Think in Action*. New York : Basic Books, 1983
- [Sil02] SILVERMAN L. K.: *Upside-Down Brilliance: The Visual-Spatial Learner*. 1st. Edition. Denver, CO, USA : DeLeon Publishing, 2002

- [ZWA00] VAN ZWANENBERG N. , WILKINSON L. J. , ANDERSON A.: Felder and Silverman's Index of Learning Styles and Honey and Mumford's Learning Styles Questionnaire: How do they compare and do they predict academic performance? In *Educational Psychology* vol. 20 (2000), Nr. 3, pp. 365-380
- [Zyw03] ZYWNO M. S.: A contribution to validation of score meaning for Felder-Soloman's Index of Learning Styles. In *Proceedings of the 2003 American Society for Engineering Education Annual Conference & Exposition* (2003)

CHAPTER 7: PAPER 3

Deconstruction/Reconstruction: A Pedagogic Method for Teaching Programming to Graphic Designers

Submission history	Accepted	Publication State
SIGCSE 2017 (Paper) Creativity & Cognition 2017 (Pictorial) Generative Arts Conference 2017 (Paper)	Generative Art Conference 2017 (Paper) GASATHJ Journal (Condensed Version)	Published (Paper)

Abstract

This paper proposes, describes and exemplifies a hands-on, experiential pedagogic method, *deconstruction/reconstruction*, specifically designed to introduce graphic design students to programming in a visual context. The method uses pre-existing commercially applied graphic design specimens as its main material to contextualize programming into a domain familiar to the audience. Observations of the method used in teaching are discussed, and its potential evaluated based on feedback provided by the students.

1. Introduction

Being code-literate is considered a crucial ability in today's society. Permeating through all parts of contemporary culture, this view is also influencing the education of graphic designers, prompting students to recast their existing skills to fit the medium of the code and educators to develop new courses that help build this literacy [1, 2, 3]. However, most graphic design students perceive programming as an abstruse skill they will never be able to master, and have a hard time trying to connect the activity of programming with the essence of their profession; crafting visual artifacts. Although many attempts have been made to teach programming to a visually oriented audience, most of them use seemingly random layouts, bouncing balls or simple characters in monochrome color schemes (e.g. [4, 5, 6]) to illustrate programmatic principles. To an audience, who equate a lack of aesthetics with a lack of relevance, neglecting the importance of the visual quality causes them to lose interest. To encourage graphic designers to explore programming as a creative tool, it is vital that new teaching strategies be developed, tailored to fit how this specific audience acquires new knowledge. In a contribution towards building computational literacy among graphic designers, this paper proposes and describes a hands-on experiential pedagogic method, *deconstruction/reconstruction*, specifically designed to introduce programming in a visual context.

2. Background and influences

For nine years I have taught introductory programming classes to undergraduate graphic designers at The Danish School of Media and Journalism. During this time, I have observed some recurring critical issues that negatively affect student retention, engagement, and learning outcome:

- Students find it hard to relate the activity of programming to their line of work.
- Students feel intimidated by the prospect of working with mathematics, logic, and structure.
- Students respond poorly to a lack of aesthetic quality in the output produced by their code.
- Students are easily distracted when asked to consider aesthetic issues. They quickly obsess over design-related issues, forgetting that their primary goal is to learn how to program.
- Students lack a starting point for their knowledge construction. As novice programmers they spend their time in the bottom half of Anderson and Krathwohl's Taxonomy [7], not yet in a position where they feel confident about programming to be creative with it.
- Students respond negatively to passive auditorium lectures and abstract, verbal explanations.
- Students are deterred by strange syntax and indecipherable error messages.

Seeking to alleviate these issues, I decided to develop a new pedagogic method specifically tailored to accommodate the learning needs of my students. To inform the design of the method, I summarized my observations into a set of guidelines:

- The link between programming and crafting of visual artifacts must be clearly visible.
- The output of the programming exercises must be visual
- The output must possess an aesthetic quality that makes it useful and sellable at a professional level.
- Students must be given an "object-to-think-with" [8], a cognitive artifact to serve as a link between their pre-existing internalized mental structure ("how to create graphic design") and the formation of new abstract knowledge ("how to program").
- Students must be given a fixed goal to provide a clear focus. Also, a fixed goal can serve as a measuring stick allowing students to continuously evaluate their progress.
- Students should not be asked to consider aesthetic issues to keep them focused on learning how to program.
- Mathematics, logic, and structure should only be taught when the students encounter a need for it, preferably by letting the students investigate the topic themselves, guided by the educator.
- Students must be given the same material to encourage sharing of knowledge and discussion around a common base.
- Students must be actively engaged in the task of programming to build hands-on experience.
- Students must work in a programming environment that provides a low threshold (easy entry to usage for novices), high ceiling (powerful facilities for sophisticated users), and wide wall (a small, well-chosen set of features that support a wide range of possibilities) [9].

I chose to build the method around the recreation of pre-existing design specimens. This decision resolved several issues at once: It established a direct link between programming and design, introduced a relatable "object-to-think-with" that doubled as a fixed target, thus eliminating the risk of students losing focus by being having to make aesthetic choices.

Constructionism was chosen as the theoretical foundation of the method. Among other things, constructionism let students use the information they already know ("how to create graphic design") as a foundation for acquiring more knowledge ("how to program") in a different domain. Also, constructionism holds that learning happens most effectively when students are active in making external artifacts they can reflect upon and share with others. Finally, constructionism prescribes that the educator must take on a mediational role as opposed to an instructional role, assisting students to individually understand problems in a hands-on way.

Guzdial [10, 11] suggest that teaching programming needs to be contextualized and meet the needs of the learners. The target audience is intended to merely be "programming tourists," [12], thus a rigorous adherence to "correct" Computer Science terms was abandoned in favor of a terminology that better helped students build cognitive models of programmatic principles. Another key factor in favor of contextualization is to make apparent the usefulness of programming in the student's profession.

A term introduced by Papert [8] and later popularized by Wing [13], Computational Thinking deals with thought processes involved in formulating a problem and expressing its solution(s) in such a way that a computer—human or machine—can effectively carry out [14]. Key principles in Computational Thinking are:

- Decomposition (breaking down a complex problem into smaller, more manageable parts)
- Pattern recognition (looking for similarities among and within problems)
- Abstraction (focusing on the important information only, ignoring irrelevant detail)
- Algorithms (developing a step-by-step solution to the problem, or the rules to follow to solve the problem).

These principles influenced the design of the method and are embedded in the activities therein.

Finally, the work of Stahl [15] also informed the design of the method. According to Stahl, transforming tacit preunderstanding into a computer model happens in a series of successive steps. In his discussion, Stahl, among other things, suggests a taxonomy of classes of information [15, pp. 178-183]. This taxonomy greatly inspired the design of the method to be a number of sequential steps divided into two distinct phases.

3. Method described

The deconstruction/reconstruction method consists of two successive phases, deconstruction, and subsequent reconstruction. Each phase has three steps. Activities associated with each step are briefly described in figure 1. A detailed account of how the method is applied in practice is given in section 4 of this paper.

Phase	Step	Activity	Material	Domain
Deconstruction	1	SELECT Choose a pre-existing graphic design product specimen to be deconstructed from the sample set provided by the course instructor.	Paper and pen	Graphic Design
	2	DESCRIBE Make detailed notes about immediately visible visual components, e.g., shapes, typography, colors, scaling, rotation, grids, rhythm, and repetitions.		
	3	ANALYZE Identify and formalize invisible components, e.g., interconnections, math, logic and rules required to control the design system and make it behave as desired.		
Reconstruction	4	CONVERT Convert notes from steps 2 and 3 into code that replicate the chosen specimen. Use the original specimen as a visual reference to guide and evaluate the process.	Code	Computer Science
	5	EXPLORE Modify the variables in the design system to create alternate versions of the original specimen.		
	6	TINKER Modify (and possibly break) the code to create radical mutations of the original specimen.		

Figure 1: Schematic overview of the deconstruction/reconstruction method.

The purpose of the deconstruction phase is to keep the students in their comfort zone by letting them rely on their pre-existing knowledge of graphic design principles and terminology to deconstruct an existing design product to form the basis of the reconstruction phase. The purpose of the reconstruction phase is to let students discover programming as a practical craft acquired by incremental conversion of their notes from the deconstruction phase into code, thereby constructing a self-contained design system capable of reproducing the chosen specimen and acting as a platform for playful discovery through manipulation of variables and the code itself.

As the student completes each step, he/she gradually shifts from using their existing skills in a familiar domain (Graphic Design) toward acquiring new skills in an unknown and unfamiliar domain (Computer Science).

Material

As its main material, the method uses pre-existing commercially applied graphic design specimens. Examples of these are posters, packaging, logos, typography, signage, bank notes, stamps, etc. Specimens are handpicked by the teacher based on their ability to be deconstructed, meaning that they must exhibit distinct visual characteristics indicating that an underlying system or set of rules has played a key role in their creation. Specimens should be easily replicable using geometric primitives, basic linear transformations (e.g., translation, rotation, scaling) and control flow statements (e.g., decision-making, looping, branching). A selection of suitable specimens that meet these criteria is shown in figure 2 to provide an idea of the visual genre.



Figure 2: A selection of specimens suitable as material for the method.

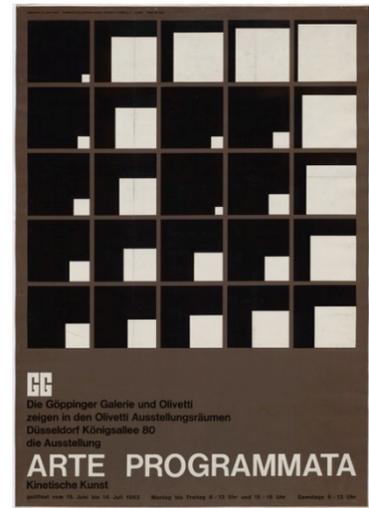


Figure 3: Poster by Enzo Mari (1963).

4. Method exemplified

In this section, the activities associated with each step of the deconstruction/ reconstruction method are discussed using Enzo Mari's 1963 poster "Arte Programmata: Kinetische Kunst" [16] (figure 3) as example. Processing [17], a popular Java-based language for learning how to code within the context of the visual arts, is used as the programming environment.

Step 1: Select

Guided by his subjective aesthetic preference, a student, Peter, chooses the Arte Programmata poster from the set of specimens provided by the teacher.

Step 2: Describe

Taking notes using pen and paper, Peter describes the poster's immediately visible components:

- "The poster is portrait format."
- "The background color is brown."
- "The upper part of the poster contains one 5x5 grid of black squares with inset spacing taking up the entire width of the poster excluding a border margin."
- "Each black square contains one white square of varying size."
- "The white squares increase then decrease in size while forming a spiral pattern."
- "The white square is fixed to the lower right corner of the black square."
- "The lower part of the poster has a white all-caps title spanning the entire width of the poster excluding the border margin + an additional black text set in a small font size aligned to the left."
- "Separating the 5x5 grid and the typography is a small white logo aligned to the left."

Peters observations are described using graphic design terminology familiar to him. Embedded in his description are clues about features that he must consider in his code (e.g. "square," "grid," "border margin," "inset spacing".)

Step 3: Analyze

Still using pen and paper as his material, Peter identifies and formalizes the underlying math, logic and rules needed to construct the poster. In the previous step, Peter loosely described a spiral pattern of oscillating white squares. In this step, he must make additional considerations to explicitly describe this spiral pattern: Is it rotating left, or right? Does it go inside out or outside in? Where are its starting and ending points? Also, looking at the oscillating squares: How many oscillations? What are the minimum and maximum size? What principle is used to calculate the rate of change in size: Sine waves? Linear interpolation? Exponential change? These observations do not translate into simple built-in commands. They require rules to be established and algorithms developed. To formalize a thing like oscillation, something that is otherwise easily (but imprecisely) verbalized, Peter is forced to look into mathematics of oscillating functions, realizing that even a seemingly simple thing like oscillating movement can be accomplished using many different techniques all of which ultimately affect the visual style of the output. No code is written yet, although, during his research, Peter comes across a pseudocode spiral algorithm that helps him understand how spiral patterns are constructed in a two-dimensional grid.

Step 4: Convert

In this step, Peter launches Processing, as he transitions from paper and pen to code. By using his notes from previous steps as starting point, Peter gets an idea of what his program must contain and do. Sampling the original artwork, he converts colors from broad descriptions to specific color codes ("Brown" = #5A4531, "White" = #F7F1E5 and "Black" = #000000). Squares are drawn using the built-in `rect()` command. The 5x5 grid is constructed using two nested `for()`-loops representing x-coordinates and y-coordinates respectively. To correctly place the black and white squares, functions like `pushMatrix()` and `popMatrix()` in conjunction with `translate()` is used. Investigating the `sin()`-function, Peter chooses a sine wave moving from 0 to π to achieve the oscillating white squares. In search of a way to mimic the spiral pattern, Peter modifies pseudocode found online to fit his needs. The typography can be made either as text or inserted as an image. Painstakingly recreating complex typography letter by letter serves no point; also, students might get distracted from programming when trying to correctly identify, download and install the font. Therefore, in this example, Peter was asked to simply cut out the original typography as a separate image using Photoshop and insert it into his program as a static image. As Peter converts his notes from steps 2 and 3, he gradually constructs a program capable of recreating the original specimen. Besides acting as an "object-to-think-with," the original poster also doubles as a visual reference used by Peter to measure his progress and evaluate the behavior of his program.

Step 5: Explore

In this step, Peter must produce alternative versions of the original poster without modifying his code. By only changing variables, in this particular case using Processing's "Tweak Mode," instant feedback is provided allowing for real-time exploration of the solution space inherently described by the code. A set of Peter's possible alternatives to the original specimen, obtained by tweaking the variables in his code, can be seen in figure 4.



Figure 4: Alternative versions obtained by tweaking variables.

Step 6: Tinker

Having gained an understanding of the "mechanics" of the code, Peter begins modifying the code itself. Now, more radical solutions emerge. The result of Peters' tinkering with his code as well as continued tweaking of the variables can be seen in figure 5.



Figure 5: Alternative versions obtained by modifying code and tweaking variables.

5. Method used in teaching

I used deconstruction/reconstruction method in two introductory programming courses taught at The Danish School of Media and Journalism. Participants were classes of 20-24 undergraduate graphic design students (ages ranging between 21-33 years, 50/50 gender ratio) with little to no prior programming experience. The aim of the courses was to equip the students with sufficient cognitive and practical skills to enable them to conceive and execute custom made code-driven design systems. The deconstruction/reconstruction method was used as a recurring daily exercise in the first week.

As prescribed in the method, I chose a sample set of 20 pre-existing graphic design specimens from a curated collection [18]. The entire set of specimens made available as handouts and digital files to the students is shown in figure 6.

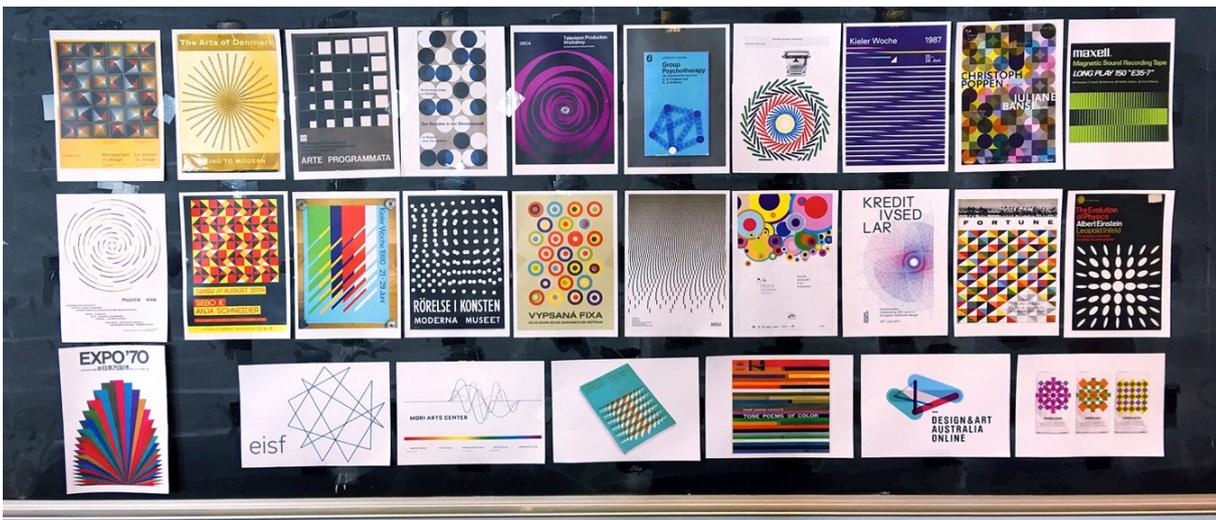


Figure 6: The collection of chosen specimens taped to the blackboard in the studio provided a quick visual overview.

Step 1: Select

Initially, choosing a specimen was a simple matter of personal preference and daily mood. Later, the students' choice was influenced by their newly acquired skills. If they had learned how to make a two-dimensional grid, students tended to choose a specimen that would allow them to reuse this programmatic feature in addition to posing a new challenge.

Step 2: Describe

The students felt confident as they began to describe their chosen specimen. Trained observers of graphic design, students had few problems describing the immediately visible components. Perhaps overly confident in their own ability to memorize their findings, I found it necessary to stress the importance of noting all observations on paper. Students spontaneously developed the habit of using Photoshop's eraser and cloning tool to remove all design components besides the background and typographic elements. This provided an authentic background to import in step 4 to make the output look almost identical to the original specimen.

Step 3: Analyze

Students began leaving their comfort zone when asked to explicitly describe the math, logic, and rules of their chosen specimen. Certain relations and behaviors were easily described using basic mathematical principles (e.g., sine/cosine, Pythagoras, linear transformations) while others relied on formulas or phenomenon one could not expect the students to know beforehand (e.g., Fibonacci series, recursion, moiré). I assisted the students in researching any formulas or techniques they might need to recreate the specimen, being careful not to provide explicit answers. This step provided a great opportunity for the students to practice and utilize Computational Thinking principles as discussed in section 2 of this paper.

Step 4: Convert

Launching Processing and converting notes into code, students gradually discovered how variables, arrays, functions, classes, as well as other programmatic building blocks, helped them extend their static system to become a fully functioning, dynamic system capable of replicating the original specimen. This step was – without a doubt – the most challenging step for the students. They spent

the majority of the time working on the daily assignment completing this step, slowly grasping programming logic, structure, looking up syntax in the language reference, and tracking down bugs.

Step 5: Explore

In this step, students used Processing's 'Tweak Mode' to manipulate variables with instant visual feedback. They would bend, stretch and inevitably break their programs. Immersing themselves in playful experimentation, students kept generating new variations from the seemingly infinite number of possibilities, always curious to discover what output their system would generate next. Students were asked to capture a visual log of their progress to show the extent of the visual diversity that their system was capable of producing. Examples from a student's visual log are shown in figure 7.

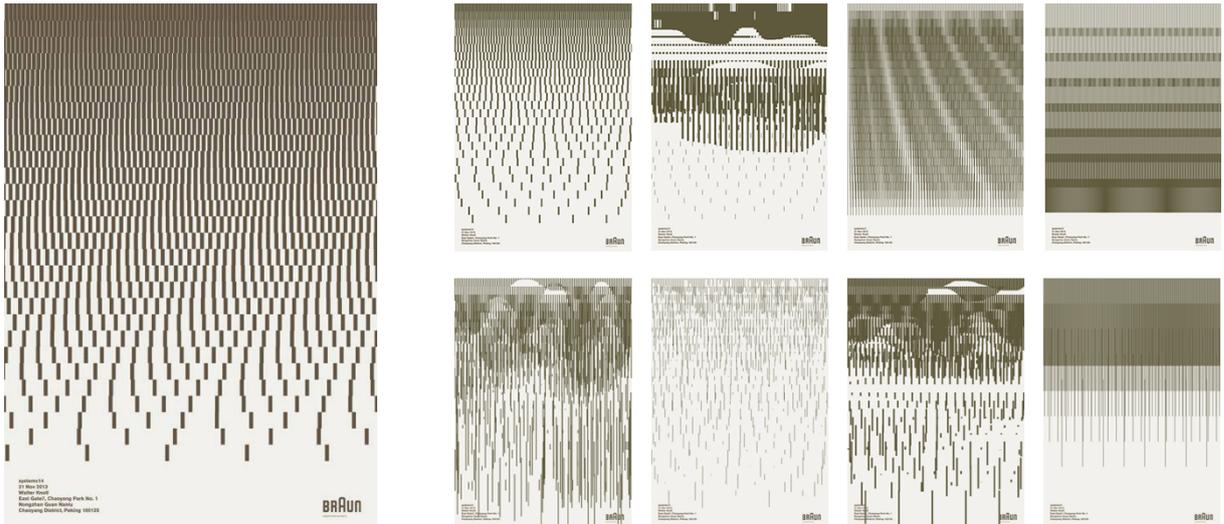


Figure 7: A student's attempt at recreating the original specimen (big image, left) [19] using code, and his subsequent experiments modifying the identified variables and the code itself to produce radically different versions (small images, right).

Step 6: Tinker

Spurred on by their active experimentation in step 5, students began to modify the code itself. Through this process, students discovered that code, although immaterial and intangible, still possess plasticity and is highly malleable. Their confidence in their abilities grew, and this kind of tinkering and hacking was encouraged to support their urge to experiment. This step gave occasion to discuss topics like version control, optimization and advanced debugging.

Most students managed to work through steps 1-6 in one day (= 7 hours of scheduled and supervised studio time). On a few occasions, students gave up trying to complete the daily assignment. This was mainly due to issues arising in step 4 as a result of their lack of experience.

True to constructionist learning theory, students were asked to share their experiences with fellow students, currently trying to solve the same specimen. This had them verbalize and explain how they had arrived at a solution, further anchoring their understanding of what they did.

6. Concluding remarks

In this paper, a pedagogic method for teaching graphic designers' programming in a visual context has been outlined and put into practice. Supported by an overall positive student response expressed in follow-up plenary interviews, the method appears as a promising way of introducing graphic design students to programming in a visual context.

The idea of contextualizing programming using pre-existing graphic design specimens was well received. Students entered their programming course with skepticism and anxiety but introducing the deconstruction/reconstruction method and explaining how it relied on familiar and well-known material defused the student's immediate aversion to code. The students also appreciated being given a real-life case as a starting point and step-by-step method to guide their learning process.

Though praised by the students, it can be argued, that repetitiously remaking work done by other graphic designers does not stimulate them to synthesize their knowledge into new independent creations. While this might be true, the deconstruction/reconstruction method is primarily designed to keep students engaged and motivated while introducing them to the nuts and bolts of programming. If students, by the rote learning and repetitive practice implicitly inscribed in the method, manage to cognitively link visual patterns with basic programmatic techniques, they have established a solid basis for taking full advantage of the creative potential of computational media in their future line of work.

To further put the social and learning-through-sharing ideas of constructive learning theory in play, one possible future improvement would be to make the deconstruction phase group-based to incite discussion and make problem-solving a more verbal exercise. Moving to the reconstruction phase, shifting to individual work will still allow for a personal hands-on experience with programming. Having multiple students working individually in parallel to implement a jointly deconstructed specimen will further increase the chances of students helping and learning from each other.

7. References

1. Tober, B. (2012): *Making the Case for Code: Integrating Code-Based Technologies into Undergraduate Design Curricula*. Abstracts & Proceedings from the Eighth Annual UCDA Design Education Summit.
2. Pettiway, K. (2012): *The New Media Programme: Computational thinking in Graphic Design Practice and Pedagogy*. Journal of the New Media Caucus, CAA Conference Edition 2012.
3. Freyermuth, S. S. (2016): *Coding As Craft: Evolving Standards in Graphic Design Teaching and Practice*. Plot(s), Volume 3, 2016, pp. 57-71. Parsons School of Design, New York, USA.
4. Reas, C. & Fry, B. (2014): *Processing: A Programming Handbook for Visual Designers*, Second Edition. MIT Press, Cambridge, Massachusetts, USA.
5. Shiffman, D. (2015): *Learning Processing, Second Edition: A Beginner's Guide to Programming Images, Animation, and Interaction*. Morgan Kaufmann, Burlington, Massachusetts, USA.
6. Reas, C. & Fry, B. (2015): *Make: Getting Started with Processing*, Second Edition. Maker Media, San Francisco, California, USA.
7. Anderson, L. W., & Krathwohl, D. R. (2001): *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*. New York: Longman.
8. Papert, S. (1980): *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc.
9. Resnick, M. et. al. (2005): "Design Principles for Tools to Support Creative Thinking" in "Creativity Support Tools - A workshop sponsored by the National Science Foundation June 13-14, 2005, Washington, DC."
10. Guzdial, M. (2007): *Contextual computing education increasing retention by making computing relevant*. White paper, Georgia Institute of Technology.
11. Guzdial, M. (2010): *Does contextualized computing education help?* ACM Inroads, 1(4), 4-6.
12. Amiri, F. (2011): *Programming as design: The role of programming in interactive media curriculum in art and design*. International Journal of Art and Design Education, 30(2), 200-210.
13. Wing, J. (2006): *Computational thinking*. Communications of the ACM, 49(3), 33-35.

14. Wing, J. (2014): *Computational Thinking Benefits Society*. 40th Anniversary Blog of Social Issues in Computing. (Retrieved November 4, 2017, from <http://socialissues.cs.toronto.edu/index.html%3Fp=279.html>)
15. Stahl, G. (1993): Interpretation In Design: *The Problem of Tacit and Explicit Understanding in Computer Support of Cooperative Design*. PhD dissertation in Computer Science, University of Colorado, August 1993.
16. Mari, E. Arte Programmata, *Kinetische Kunst*. (1963) Printed by Officina d'Arte Grafica A. Lucini e C, Milan (Retrieved November 3, 2017, from <https://www.moma.org/collection/works/8052?locale=en>)
17. *Processing*. (Retrieved November 3, 2017, from: <http://www.processing.org/>)
18. *Pinterest*. Computational Graphic Design Inspiration. (Retrieved November 3, 2017, from: <https://www.pinterest.dk/stixan/computational-graphic-design-inspiration/>)
19. Lee, J (2014). *systems 14*. (Retrieved November 7, 2017, from <http://www.leejaemin.net/systems-14>)

CHAPTER 8: CONCLUSIONS

In this chapter, I revisit my research questions, answering each of them individually. I aggregate the cumulative knowledge acquired during my study and distill my findings in a list of pragmatic and applicable heuristics. After discussing both theoretical and practical implications of my research, I suggest future research to be conducted.

8.1 Introduction

Having arrived at the conclusion to the dissertation, I will now revisit my research questions. I had one overarching research question (RQ) which I chose to sub-divide into three specific research questions (SRQs). The SRQs were investigated in the three separate papers presented in the previous chapters, each of which offered a conclusion on the performed study. Although not explicitly mentioned within the papers themselves, several underlying hypotheses (sections 3.5.1–3.5.3) provided the motivation behind them. In the following section, I will either refute or support these hypotheses based on the findings highlighted in each paper.

An answer to the dissertation's main RQ results from looking at the answers to the SRQs in unison.

8.2 Answering research questions

8.2.1 SRQ1: How is Creative Coding currently taught in graphic design education?

Based on an in-depth, comprehensive, quantitative analysis (chapter 5) of Creative Coding syllabi, I concluded that the planning for the majority of introductory programming courses currently taught in graphic design schools is mainly guided by the technical properties and affordances of the chosen programming environment. This techno-centric perspective involuntarily establishes a hierarchy in which topics relevant within graphic design are considered subsidiary to those of computer science. Moreover, this *code-first-design-second perspective* causes issues that arise as a result of either ignoring, underprioritizing, or skewing the needs of graphic designers so that they can be answered using a certain pre-chosen programmatic construct.

Further negative consequences of the current practice include (but are not limited to):

- requiring compatibility with rigid institutional class scheduling, which causes most courses to be taught through passive lecture-style classes once or twice a week. Instructional strategies like blended learning and flipped classrooms are only sporadically employed as a way to combat the prevalent didactic teaching.
- placing disproportionate emphasis on programming, which forces graphic designers to recast and restrain their pre-existing, domain-specific skills to fit the programming languages.

- offering inadequate explanation in most courses of how graphic designers will recognize, utilize, extend, and augment their pre-existing, domain-specific skills.
- failing to connect the activity of programming with the graphic design student's future vocational occupation and production of visual artifacts for use in commercial contexts.
- omitting metaphors for relating programming concepts to well-known graphic design concepts, which provides little help to students in their cognitive model-making.
- insufficiently recognizing and using visual aesthetics as an important motivational factor.
- failing to explore media-specific computational aesthetics, which discourages students from cultivating new forms of visual artifacts.
- ignoring fundamental graphic design methods (e.g., typography, color mixing) either due to the non-design educator's ignorance or because it is deemed impossible to exercise them with a desired level of finesse due to a coarseness and lack of supporting options within the chosen development environments.
- not paying enough attention to debugging strategies and troubleshooting, which results in an inability to identify and solve problems, causing frustration and discouraging students when they become stuck doing their homework assignments.

Overall, I have concluded that there is an urgent need to put graphic design theory and practice first and subsequently introduce programming concepts as needed. I consider this approach to be essential for effectively promoting and embedding programming as an established practice in graphic design education.

The hypothesis for this specific research question was that contemporary Creative Coding courses are mainly taught using a computer science approach that is not beneficial to graphic design students. I consider the hypothesis to be supported.

8.2.2 SRQ2: How should Creative Coding be taught to accommodate how graphic design students learn?

As observed in SRQ1, a *code-first* approach is used by most design educators to inform and guide the planning of their Creative Coding courses. This entails course structures originally intended for a non-design audience being transferred to graphic design students in a form that has only been moderately adapted. I was interested in investigating whether there were any notable differences in the learning style profile of graphic design students and students within technical fields. A quantitative analysis (chapter 6) of Felder-Soloman Learning Style Index (ILS®) questionnaires, completed by 77 graphic design students, showed that their learning style profile differed noticeably from that of students in technical disciplines. This implicitly underlines the need to develop customized programming courses and accompanying instructional methods for use in design schools.

In response, the following specific recommendations were made:

- Teaching materials, demonstrations, and assignments must be highly visual.
- Visualizations or metaphors tied to the graphic design students' pre-existing, domain-specific knowledge must be used to help them develop mental models of abstract programming constructs.
- Live coding and visual demonstrations should be used whenever possible.
- Exercises and assignments should be designed to have a fixed goal but allow for different ways of arriving at a solution either through experimentation or through stepwise instructions.
- Educators must be careful to provide the big picture and relate it to the students' previous knowledge before going into details.

- Teaching must actively engage the students; however, initiatives that call for reflective activities should also be integrated.
- Students should be given the option to work either in pairs or alone, depending on their personal preference.
- Students should be given objects to assist their exploratory discovery; this might be visual specimens, premade code snippets, or a set of digital assets to use.
- Providing additional explanatory tutorials and demonstrations, preferably in the form of video/animations, is highly encouraged.
- Other suggested activities to support graphic designers in building cognitive models of programming constructs are verbal walk-throughs of algorithms, tests that involve math and logic, quizzes, and code-related puzzles.

The hypothesis for this specific research question was that contemporary Creative Coding courses aimed at graphic designers are not taught in the most optimal way and do not account for how they learn. I consider this hypothesis to be (partially) supported.

8.2.3 SRQ3: How can graphic design students be motivated and supported as they are introduced to programming?

This question holds many possible answers. As a deliberate decision, arising partly from my choice of pragmatism as my paradigmatic stance and partly from the study's initial call for practically applicable contributions, I refrained from answering this question by engaging in a broad theoretical discussion. Instead, I chose to provide one possible answer in the form of a specific pedagogic method. Through gradual interweaving of the findings from SRQ1 and SRQ2, a contextualized pedagogic method, *deconstruction/reconstruction*, specifically devised to teach graphic designers programming in a visual context was developed and tested. The method relied on familiar and well-known materials to defuse the graphic design students' initial aversion to programming. Further supporting them was a six-step method to guide their learning process.

Though they cannot be claimed to constitute an exhaustive and unified answer to the specific research question, several partial findings were, however, deduced from the empirically gained observations made while iteratively testing the proposed method. The following are based on a summary of these findings and indicate ways in which the learning of graphic design students can be motivated and supported:

- Planning course structure and content using a *design-first* perspective (see paper 1, chapter 5)
- Employing instructional strategies that match the students' preferred learning style (see paper 2, chapter 6)
- Easing the transition from well-known graphic design domain-specific knowledge to the unfamiliar and unknown programming domain to avoid abrupt learning barrier thresholds.
- Including real-life cases that involve relatable commercial graphic design products. Such cases resonate with the students and give them something to aspire to.
- Engaging students in practical hands-on activities to the widest extent possible. Didactic lectures meant to introduce and demonstrate a given programmatic principle must be kept short or even possibly replaced by pre-recorded lectures and on-demand instructional material.
- Maintaining a high level of aesthetic quality in instructional material, exercises, and assignments.
- Emphasizing utilitarian value to help establish a direct link between programming and the students' primary field of study.

- Making programming a social activity that involves the creation of shareable visual artifacts which, when presented, can spark discussion and admiration among the students.
- Establishing a fear-free environment and relieving any initial anxiety felt by the students. Typically, this involves a reassurance that the quality of their code will not affect their grade and that failure to successfully complete an assignment is allowed, provided that a persistent attempt was made.
- Providing a step-wise method to guide the students' progression towards the fixed goal. However, it must be emphasized that students are free to deviate from—or even abandon—the method to arrive at the fixed goal through individual and alternative paths.
- Limiting the students' creative freedom by providing fixed goal and associated assets as a deliberate strategy to prevent opportunities for making aesthetic decisions from derailing students' focus on learning to program.
- Providing a shared pool of assignments—all to be solved but in no particular order—as a way to promote and encourage knowledge-sharing among students.
- Focusing on building students' competencies in bug tracking and problem identification strategies to enable them help themselves when they get stuck.

The hypothesis for this specific research question was that contextualizing programming to fit the students' predominant learning style and pre-existing, domain-specific skills would improve the graphic design students' motivation and desire to learn programming. Additional studies on contextualized pedagogic methods for use in graphic design education are required to either refute or support this hypothesis.

8.2.4 RQ: How should programming ideally be taught to graphic designers to account for how they learn and how they intend to integrate programming into their vocational practice?

Providing a succinct answer to the study's main research question can be done by synthesizing the answers given to the three specific research questions. However, it is important to stress, that due to the open-ended nature of the research question, the answer provided in this dissertation is not to be seen as the only *true* answer, rather as one *possible* answer arising from the findings accounted for in the research papers and outlined in the sections above.

Recapitulating this study's conclusions into one condensed paragraph, a broad answer to the main research question is as follows:

Programming should ideally be taught to graphic designers as a studio-based social activity practiced in a safe environment using relatable metaphors, familiar materials, visual examples, and live, interactive demonstrations. Courses should be planned utilizing a design-first perspective to allow the graphic design students to use their pre-existing, domain-specific skills to leverage their acquisition of programming knowledge and relate it to their vocational practice. Ideally, flipped classroom and blended learning instructional strategies should be used advantageously to move transferable knowledge out of the classroom, allowing instead for contact time being used to engage in plenary discussions, presentations, troubleshooting, and transfer of tacit knowledge through experiential, hands-on discovery learning. Assignments should be available in varying difficulties, solvable individually or in pairs/groups, provide a fixed goal, include premade assets, and limit the need for aesthetic decisions to ensure students stay focused on learning to program. Also, assignments should have a clear utilitarian purpose within a graphic design workflow and mimic restrictions found in the professional industry.

8.3 Theoretical implications

As a whole, this study extends a line of research (Amiri 2011; Tober 2017; Maeda 2002; Freyermuth 2016; Tober 2012b; Pettiway 2012) that emphasizes the importance of introducing programming as a core part of the graphic design curriculum. More specifically, the study makes two major contributions to the literature on programming in graphic design education. First, it introduces two opposing approaches to planning Creative Coding courses: *code-first* versus *design-first*.

Understanding the distinction between these approaches and what they each entail will help inform and clarify the ongoing debate. Second, it determines the differences in the way graphic designers prefer to learn compared to students in technical disciplines. This knowledge must be considered by researchers who seek to contribute to the epistemological framework for teaching programming to visually inclined non-programmers that is currently missing from the literature.

Findings obtained in this study can encourage and elevate dialogue among researchers, educators, scholars, and practitioners, mainly within the field of graphic design, but not limited to this. Due to the multi- and interdisciplinary nature of my research area, this dissertation's theoretical contributions are also relevant and useful within many disciplines beyond graphic design (e.g., to computer science educators who want to teach introductory programming using visual examples, and to researchers within the field of applied pedagogy investigating pedagogic and didactic measures related to teaching programming).

8.4 Practical implications

Because it was explicitly stated as mandatory in the call for proposals for this study, arriving at an outcome that can be put to immediate, practical use has been an aim—as well as a personal desire—of my research. Hence, this study proposes several initiatives with direct practical implications, of which the most relevant are discussed below.

The pedagogic method, *deconstruction/reconstruction*, proposed in paper 3, can be employed as is in courses that teach introductory programming through visual materials. Though it has been tailored specifically for use in Creative Coding courses taught in design schools, the method can also be used by educators within the field of computer science who use art and visual design as contextualized approaches in their teaching.

Already mentioned as having theoretical implications, the proposed notion of a *code-first* versus *design-first* approach also has practical implications. Bringing this schism to educators' attention, enables them to consciously take on and uphold a *design-first* approach in their planning, in turn affecting their choice of structure and content and supporting instructional strategies.

The study of graphic designers' learning style profile and how it relates to programming courses interprets and convert its findings into a number of practically applicable didactical suggestions. When implemented by educators, these suggestions will help lower—or entirely remove—knowledge acquisition barriers that arise from discrepancies in how programming courses lifted from technical disciplines are modeled and how graphic design students prefer to learn.

Though still far from exhaustive, the heuristics in their present form offer educators who teach Creative Coding in design schools some much needed guidance to help them critically reflect on and question their own planning, ultimately resulting in better courses.

8.5 Limitations

As is always the case when coming to the end of a research study, things come to mind that could have been done differently or should have been included, such as additional perspectives, analyses,

or explanations. Below, I will briefly discuss the study's limitations and give suggestions for how they can be overcome.

8.5.1 Paper 1

The study in paper 1 (chapter 5) was limited by the number of Creative Coding syllabi available online. Peer reviewers suggested that syllabi could be gathered from a defined list of top-ten design universities; however, this suggestion presents a number of obstacles. First, upon further investigation, not all of these design schools offer Creative Coding courses. Next, to provide a broad overview of contemporary Creative Coding courses, I found it necessary to include Creative Coding courses from trade schools, colleges, and universities as graphic design education is not exclusively taught at universities. Furthermore, as most of the analyzed courses spanned an entire semester (16 weeks) and were physically located across three continents, it was logistically impossible to attend the classes to do observation studies. Considering it an alternative approach rather than a limitation, I acknowledge that a narrower range of courses permitting observational studies would likely yield a more-detailed and nuanced understanding of how Creative Coding courses are taught.

8.5.2 Paper 2

In paper 2 (chapter 6), 77 graphic design students were profiled using the Felder Soloman ILS[®] questionnaire. Compared to other similar studies, the number of respondents qualifies the study as a valid contribution. However, to make any conclusions regarding the learning style of graphic designers in general, a significantly larger body of respondents would be required. Also, these should preferably represent design schools across the globe to account for cultural differences in teaching and learning style. Additionally, the results obtained in my study could be further strengthened through statistical analysis; however, this requires access to the raw datasets of the referenced studies. Although an independent and useful contribution, the quantitative results obtained in the study must be complemented by qualitative insights from other related studies if the aim is to gain a broad, holistic, and comprehensive understanding of how graphic designers learn.

8.5.3 Paper 3

Due to the asynchronous development of the paper (described in section 3.6), paper 3 (chapter 7) as it appears in this dissertation does not include the latest modifications to the proposed pedagogic method made on the basis of findings in paper 1 and paper 2 as well as the students' feedback. While the paper provides a practically applicable method, it is less specific in its description of the instructional strategies that surround the method. Even though they do not appear in the paper, several of such strategies are discussed in this dissertation's roundup of recommendations (chapter 9). Also, the evaluation of the usefulness and efficiency of the method is based solely on qualitative accounts of the students' personal experience of working with it and my personal empirical observations from previous courses where the method was not employed. To thoroughly assess its effectiveness, the method must be compared against other pedagogic methods for teaching programming. Also, performing a range of quantitative tests to examine the method's ability to retain, motivate, and engage the students, as well as teach them fundamental programming skills in a context that encourages them to pursue programming, will further qualify the method as a valuable addendum to the graphic design curriculum. Ideally, these tests should also be performed in several other design schools to ensure a broad coverage in the results obtained.

8.6 Future Research

Building upon the findings in this dissertation, several avenues for future research emerge.

Generally, more initiatives are needed that explore new pedagogic and didactic approaches to teaching programming to graphic designers. As we continue to develop a better understanding of how graphic designers prefer to learn (to which paper 2 is making a contribution), we will be able to refine and optimize our teaching activities and instructional strategies accordingly. The holy grail of how best to teach programming to graphic designers has yet to be found.

One possible study could involve a detailed examination of how computational thinking (CT) principles could be contextualized to fit the graphic design curriculum. As the nature of graphic design is inherently systemic, CT can easily be introduced in classes teaching typography, color theory, layout, animation, and more. Infusing traditional graphic design subjects with CT, enables students to construct cognitive models of their acquired skills in a way that is compatible with and transferrable to code.

A second area of study concerns the role of code in graphic design education. Code can be used in numerous ways: from quickly sketching ideas, conducting experiments to spark ideas, and providing single-purpose tools to be used in-house, to providing refined and complex design systems delivered to a client. A study devoted to developing a deeper understanding of the many ways code can be used is essential to further justify and integrate code as an integral part of graphic design education.

Another study could focus on developing a generic Creative Coding syllabus specifically intended for use within graphic design education. Through my study of Creative Coding courses, I observed a host of useful algorithms, topics, exercises, assignments, pedagogic methods, and instructional strategies. However, these were wildly scattered and arbitrary, suited to fit the individual courses. Additionally, my study also revealed an absence of particular content useful to graphic designers. A systematic synthesization and augmentation of current best practice resulting in a bespoke syllabus with accompanying teaching materials and assignments would be highly welcomed among educators.

A study to determine if programming should be taught as short, week-long workshops or semester-long, one-day lectures also seems highly relevant. Currently, most design schools teach programming as semester-long electives alongside traditional design courses. Other design schools have opted to introduce programming through intense, week-long workshops with no other concurrent courses. What are the consequences associated with each approach? Are short, intense courses better at making learning stick, or is it then quickly forgotten? Do longer courses make learning stick permanently, despite the students' inability to devote their entire attention to programming?

A longitudinal study examining how graphic designers adopt programming into their workflow is also relevant. Teaching designers how to program is only worthwhile if they manage to integrate it into their everyday practice. Tracking how graphic design students use programming while they are in school and when they have graduated and entered the graphic design industry can reveal the long-term effects of programming as a subject taught in graphic design education.

Several interesting future studies can also be derived from the proposed heuristics, as will be explained in chapter 9.

Finally, other relevant research could include critical studies on how computation contributes to the evolution of the graphic design discipline, as well as studies to determine the future roles of the computationally literate graphic designer within the industry.

CHAPTER 9: HEURISTICS

9.1 Introduction

During this research project, I have encountered and absorbed extensive knowledge and gained invaluable insight through scientific publications, books, informal interviews, classroom experiences, experiments, discussions at conferences, and workshops, to name a few. To aggregate my cumulative findings, gathered over four years of research, I will now elaborate on the answers to my research questions by interweaving other relevant and influential sources not directly mentioned in my papers. I will convey this “knowledge spillover” in the shape of a list of pragmatic and directly applicable heuristics aimed at design educators who teach Creative Coding to graphic design students.

1: Remember that graphic designers are not artists

To frame and situate these tips, it is key to distinguish between art and graphic design. The resolution of this distinction is a major philosophical conundrum, and an on-going and convoluted debate attempting to define and isolate both concepts has yet to produce a clear and widely agreeable definition (Avital 2017). Art and graphic design share similar qualities and are closely associated (Monnier 1995) and thus are not mutually exclusive. However, some characteristics that distinguish graphic designers from artists are crucial to consider. For instance, graphic designers

- are problem framers (Dorst 2010), and problem solvers (Clohessy 2011) who use a broad range of theory, methods, and tools to tackle *wicked problems* (Rittel & Webber 1973; Buchanan 1992);
- use their acquired skills to craft visual communication that conveys the same message to an audience in order to motivate, inform, persuade, or incite;
- have a developed sense of beauty and aesthetics, and qualify their choices of visual elements based on objective insights regarding their efficiency in a given circumstance (Flores 2016);
- direct the observer to take a certain action;
- operate based on a fixed starting point (client’s problem) and are goal-driven (client’s expectations);
- operate in a commercial context guided by a client's needs and have to account for numerous limitations;
- are aware that their work is often part of a larger construct;
- realize that their work serves a specific function which objectively informs their design decision.

These traits are important to account for as they will dramatically influence the course’s structure, topics, instructional material, teaching style, assessment, and learning outcomes. For Creative Coding courses in graphic design education to be relevant, they must conform to the many issues of the trade, which the artist does not need to worry about: consistency, identity, homogeneity, identification, staying on brand, conveying the right message, legibility, functionality, applicability, reproducibility, flexibility, adaptability, technical limitations, and client feedback.

2: Use relatable materials

Constructionism (Papert & Harel 1991) prescribes that students use their existing knowledge to acquire new knowledge (Papert 1980; Kafai 2005). Papert (1980) has also stressed the importance of relating what is new and to be learned to something that is already known. In the case of teaching

programming to graphic designers, *new* is programming, and *what is already known* is graphic design. By using materials that students can immediately relate to, students need not first establish a link between the materials and their trade and then subsequently establish a link between their trade and programming. Choosing relatable materials allows for a direct understanding of how programming is relevant. Being able to recognize and relate to the teaching materials also helps build confidence (Keller 2010) and reduces the anxiety towards programming often felt by students with no prior experience of it (Byrne & Lyons 2001; Rogerson & Scott 2010). A final proponent for the use of relatable materials is Keller (2010), who has argued that *familiarity* (referring to the ability to tie the instruction to the learners' experience) plays a crucial role in the students' motivation.

Examples of materials that graphic designers can intuitively relate to are given by Hansen (2017) who uses them as the core foundation of his pedagogic model. Also, it should be remembered that relatable materials should not include art (see #1) or images/illustrations lifted from non-design contexts (see #12).

3: Arouse and inspire

A successful course should grab and maintain the students' attention. Keller (2010) suggests a number of strategies to accomplish this, two of which pertain to arousal. Arousal can be achieved through relatable visual examples originating within the student's field of study (see #2) or by stimulating an attitude of inquiry. Consider an assignment built around the visual topic of moiré. Showing carefully chosen examples similar to the assignment (pre-existing posters using moiré patterns), a completed version of the assignment (interactive moiré pattern explorer), or asking questions to be answered by completing the assignment ("how can we work with moiré as a visual expression?") can serve as a hook to capture the students' interest. It gives them something to aspire to as well as an idea of what they will be able to achieve when they complete the assignment's learning objectives. According to Ryan & Deci (2000), students are more likely to invest the time necessary to complete the assignment if they enjoy it or find it interesting. Hidi & Baird (1986) also found that students could better recall content of what was interesting to them. Finally, Dewey (1913) argued that interest must be coupled with effort before real learning will take place.

4: Highlight utilitarian value

Students are more motivated when they can see the usefulness of what they are learning and when they can use that information to do something that has an impact on others (National Research Council 2000, 61). The ARCS Motivational Model (Keller 2010) also discusses the importance of relating the assignment to the learners' goals. Specifically, it distinguishes between present worth and future value (Keller 2010, 127). When highlighting the utilitarian value of an assignment, it should be related to *both* the student's present situation ("how can I use this in my ongoing studies?") and future occupation ("how can I use this in my future work as graphic designer?") The assignment's learning outcome should be framed using words that describe how the acquired knowledge and skills are directly applicable to the students' workflow (e.g., use sine/cosine to make a tool that computationally produces vector shapes to import in Illustrator) and what the advantage/effect is over their existing tools (e.g., ability to iterate faster through solutions and take an exploratory approach to producing shapes).

Makers at heart, graphic design students will assess the learning outcome of an assignment using a pragmatic view (Flores 2016). Failing to highlight the utilitarian value will negatively affect the students' motivation to learn (Keller 2010, 97).

5: State a vocational purpose

The defined disciplinary purpose of the assignment should be explained by relating it to how it can be used in a real-life professional workflow. Even assignments meant to teach fundamental computational topics (e.g., loops), should always be accompanied by an explanation of their usefulness in a design process (e.g., ability to iterate faster, make grids, repetition design principle). If the sole purpose given in an assignment is to solve it in order to advance in the course, student motivation will drop. Understanding an assignment's intended use in a professional workflow (beyond the classroom) increases the likelihood that students will find the assignment valuable (see #4).

Also, stating a defined disciplinary purpose provides an opportunity to discuss the multitude of use scenarios for code-based graphic design tools in existing workflows. How will it be used, and by whom? Will it be a temporary sketch used as an intermediary towards understanding a larger idea? Will it be a crude tool intended to be used once, then discarded? Will it be a tool that the graphic designer develops for himself/herself and makes multiple use of and continues to refine? Will it be used to facilitate a collaborative process between the designer and a client? Will it be the actual output that is sold to the client for their internal use? Will it be used by a broader audience, thus requiring it to be stable, intuitive, and attractive?

6: Provide authentic restrictions

A tenet of constructivist learning theory is that learning results from participation in authentic activities based on problems that students might encounter in "the real world" (Alesandrini & Larson 2002, 119). As mentioned in # 1, graphic designers are typically given a start-point and a desired end-goal in the shape of a creative brief. Rittel & Webber (1973) argued that most of the problems addressed by designers are *wicked problems*. These wicked problems, Buchanan explained, have their "wickedness" taken out of them when someone specifies briefs in great detail (Buchanan 1992, 17). However, intentionally stripping the "wickedness" by providing authentic restrictions in the form of creative briefs can help students maintain focus on the learning objectives. Creative briefs normally come with an abundance of restrictions: message, tone-of-voice, visual appearance, economy, audience, and so forth. Emulating this (to an extent that is not counterproductive) in assignments has two immediate benefits: first, creative briefs mimic the graphic design students' future work scenario, thus providing opportunity to practice receiving and interpreting these creative briefs; and second, once understood by the students, the assignment's inherent restrictions define a solution space for them to operate within. In addition, Brown & Wilson (2018, 4), referring to Guzdial (2013), argued that the use of authentic tasks is preferable, particularly in cases when an informal end-user programmer can be motivated by a contextualized approach to learning programming. Generally, learners find authentic tasks more engaging than abstracted examples (Brown & Wilson 2018, 4).

7: Limit the need for aesthetic decisions to keep the focus on programming

Working within the confines of a given creative brief (see #6) limits the amount of aesthetic choices a graphic design student needs to make. While this might be perceived as a negative consequence, it can actually be beneficial. Given too much liberty and freedom in choosing the visual look and feel of an assignment, students tend to devote an inordinately large portion of the allotted time to making choices about fonts, illustrations, nuances of colors, cropping images, or similar decisions regarding aesthetics. Students are easily led astray, becoming lost in decisions relating to their "comfort zone," thereby derailing their focus on the primary objective of the assignment: learning to program. Replicating existing graphic design specimens one-to-one (Hansen 2017) or adhering to visual

guidelines already decided upon help students stay on track and focus on programming. However, the chance to practice artistic freedom should not be removed; some slack should be allowed—assignments should not become straight-jackets.

8: Stimulate positive aesthetic responses

Graphic designers are visual *feinschmeckers* with a well-developed aesthetic sense. Making a conscious effort to evoke positive aesthetic responses is an important factor in motivating the students to pick up programming (see #3). Failure to do so can lead students to make the wrongful assumption that programming is incapable of producing beautiful output that meets the professional standards demanded by their future clients. To avert this, it is important to establish and maintain a high aesthetic standard in the assignments.

Ulrich (2006) conjectured that *"a positive aesthetic response is more likely to lead to a positive ultimate preference, than if the initial aesthetic response were negative."* In the case of Creative Coding courses, to build a positive preference for programming, both instructional materials and assignments must stimulate positive aesthetic responses. Also, Ulrich (2006) has argued that *"an initially positive aesthetic response may result in a greater chance of further analysis and exploration by the user. A negative aesthetic response may dissuade the user from ever learning more about the artifact and therefore reduces the chance that an ugly, but otherwise preferred, artifact will ever be fully evaluated."* If graphic design students in Creative Coding courses are given assignments that evoke a positive aesthetic response, they will be more inclined to analyze and explore the topic of the assignment. While some assignments that successfully illustrate a computational concept work well on non-designers, graphic design students' susceptibility to an assignment's sub-par visual quality causes them to not fully evaluate its usefulness. Last, Ulrich (2006) suspected that *"aesthetic preferences are 'sticky.' That is, positive aesthetic judgements create a positive bias that persists even in the face of mounting negative analytical evidence. Conversely, negative aesthetic judgments persist even when further analysis reveals highly positive attributes."* Again, relating this to Creative Coding courses, once graphic design students have developed a negative bias toward the aesthetic quality of visuals produced using code, it is difficult to overturn it and can severely obstruct any attempt by the educator to prove students wrong in their perception of programming as a worthwhile skill to learn.

Asking students to work on visually meaningless, insignificant, or boring results will make students connect their experience of the output being ugly to the activity of programming itself ("I hate this generated layout"), the process that created it ("I hate my code"), and possibly extend it onto the act of programming itself ("I hate programming"), which may, in turn, result in programming being loaded with a host of negative connotations.

Granted, graphic design students have individual perceptions of what constitutes beauty; however, to the largest extent possible, educators should strive to use materials that students find beautiful as well as have the students aspire to pursue beauty in the output they create.

9: Allow students to enjoy their work

Once students manage to produce beautiful artifacts using code, they should be given ample time to appreciate and enjoy the fruit of their labor. Also, by dwelling on their success and receiving verbal feedback from educators and classmates, students reinforce their intrinsic motivation to continue learning programming. In a study investigating digital design students' motivation to learn programming, Takemura et al. (2008, 355) concluded that *"to maintain or raise students' motivation it is more important to allow students to enjoy viewing the final results (artwork) of the programming than to make them strive to create more beautiful artwork."* Promoting a feeling of satisfaction involves

taking time to appreciate the students' outcomes and providing positive recognition from educators, fellow students, and external peers (Keller 2010, 188).

Another strategy is to have graphic design students show their output to each other. Such exhibitions manifest a core tenet of constructionist learning theory: that learning is tied to the active construction of a public entity (Papert & Harel 1991), and that learning results by engaging socially and reflectively while sharing both the produced artifact and the creation process (Ackermann 2001; Stager 2001). Exhibitions can foster fruitful discussion among students, providing them with a chance to verbalize and reflect on the process, share experiences, and receive acknowledgments from their peers, as well as display the diversity of possible solutions to the same assignment.

10: Assess aesthetic quality— not code quality

Keep in mind that graphic designers are not programmers, nor should they be. Amiri (2011, 205) described graphic designers who code as "programming tourists," and Dorn & Guzdial (2006) referred to graphic designers as "end-user programmers." While students studying computer science should have a deep foundational knowledge of aspects related to computing, graphic designers can adopt a more informal and exploratory approach to programming. To them, programming is just a means to a higher purpose: crafting visual communication (Amiri 2011, 205). Excessive focus on technical aspects of the code will cause teaching to depart from the main topic—graphic design—and venture into a proverbial jungle of technical jargon that can seriously impede students' motivation.

When graphic design students in a Creative Coding course manage to produce visually beautiful output, their feeling of success should not be devalued by only pointing out the flaws in their code. Also, following the crit tradition of studio teaching, graphic design students (rightfully) expect to receive feedback that addresses the aesthetic aspects of their work. Failure to assess aesthetic quality leaves students unfulfilled. This is not to say that the quality of the code should not be assessed, but it should be critiqued according to a less-strict standard.

Techniques like refactoring and optimizing are beyond the scope of introductory courses. However, it is fair to argue that a basic understanding of how an unintentional or counterproductive structuring and grouping of computationally exhaustive calculations can substantially slow down a program is required.

11: Contextualize programming

A way to help graphic design students connect programming to graphic design is through *contextualization*. A contextualized programming course is one in which one or more application domains provide the motivation for learning computing and inspire the design of learning activities (Lukkarinen & Sorva 2016). Guzdial (2010) posited that the point of a context is to explain the usefulness (see #4) of what is being learned and that contextualization both provides relevance and helps improve retention in courses that teach programming to be used within a single domain. According to Forte & Guzdial (2005), tailored introductory programming courses, designed to accommodate the students' interest and background, can offer a motivating and engaging context for the learning of programming. Positive consequences thereof are less anxiety, increased interest, and higher achievement (Forte & Guzdial 2005).

Contextualizing a Creative Coding course targeting graphic designers involves "translating" abstract logical and mathematical programming concepts to fit the graphic design students' cognitive models that revolve around graphic design. Do not use examples that are decontextualized and demonstrate programming concepts in a generic setting. Instead, use examples that more closely

resemble artifacts (intended for communicative purposes) made by graphic design students. Not only are they more relatable (#2), they are also more aesthetically pleasing (#8), both of which are qualities that have a positive impact on the student's motivation.

Contextualization will also affect the course content. Hansen (2018b), mapping out contemporary Creative Coding courses, found that only a few took a design-first perspective when selecting and prioritizing content based on what is relevant to graphic designers.

12: Show how programming concepts have been used in graphic design

When introducing a new programming concept, contextualized examples of the concept in use should be included. For example, when discussing recursion, show a number of different graphic design products that make use of this principle. Not only will this spark the students' imagination, but it will also illustrate how the same core concept of recursion can be applied to visual products in diverse ways to achieve a multitude of looks and expressions.

Also, credit the designers who made the examples shown, not solely because of ethical and credit-where-credit-is-due motifs, but also to help students identify "role models" (Keller 2010, 131). Many Creative Coding courses apply the strategy of assigning students with the task of having to spot, research, and present graphic design products that make use of computational principles to their peers.

13: Remember that graphic design students are visual learners

Compared to students across other disciplines (Felder & Spurlin 2005), graphic design students exhibit three notable distinct characteristics: they exhibit a strong preference for visual learning (Hansen 2018a) and have a highly developed spatial thinking aptitude (Sutton & Williams 2010). This suggests that a visual approach to demonstrations and assignments (see #2, #3, #15) is both a preferable and effective way to introduce them to programming. Second, graphic design students are predominantly intuitors, but sometimes have a preference for sensing. This suggests that assignments should provide a stepwise path to completing the task, but it should not be mandatory to follow it. Third, graphic design students show a less-strong preference toward sequential learning; instead, they show an increased-mild preference. This suggests that students should be given the option to either work in pairs or work alone depending on their personal preference (see #14).

Research by Sorva et al. (2013) indicates that visualization techniques are beneficial for students. Hence, to the widest extent possible, educators should explain abstract and theoretical programming constructs using visual aids to leverage understanding, whether made specifically to illustrate a single construct (Panda 2016) or taken from the students' domain (#2).

14: Allow for pair programming AND individual programming

Pair programming is a software development practice where two programmers share one computer and take turns typing and giving suggestions. While pair programming is an effective way to teach (McDowell et al. 2006), it also has its drawbacks. As a social activity, it requires participants to be extroverted, outspoken, and open-minded. Students who are introverted, shy, or uncomfortable being watched might find pair programming very intimidating. The use of pair programming by Hansen (2017) yielded mixed responses from the students, which likely relates to the individual students' preferred learning style (#13). To avoid frustration and to honor the individual students' preferences, assignments should allow for pair programming AND individual programming, letting each student decide how he wants to work.

15: Use live coding

Live coding, which involves the educator programming in front of the students, is as good as, if not better than, teaching code via static examples (Rubin 2013). Moving from showing completed solutions on a slide to students to detailing the process by coding it "live" in front of the class, significantly improves the pedagogy of teaching programming (Gaspar & Langevin 2007). Live coding enables an active dialogue between educator and students (Brown & Wilson 2018, 2). Also, live coding facilitates unintended knowledge transfer by watching the educator work (Brown & Wilson 2018, 2). Having to type in the program slows the educator down and makes it easier for students to keep up (Brown & Wilson 2018, 2); however, the educator must keep a steady pace and avoid typing in too much code (a way to mitigate this is discussed in #21). Live coding also lets students see how to diagnose and correct mistakes, a major topic that is often omitted in Creative Coding courses and textbooks (Brown & Wilson 2018, 3). Watching educators make mistakes helps establish a "safe environment" where making and talking about mistakes is acceptable (perhaps even encouraged!). However, live coding is still mostly passive from the students' point of view (Gaspar & Langevin 2007) and should not be used as a standalone feature (Victor 2012). One way to implement live coding as a participative activity is by having students make predictions of what the code will do before executing it (Brown & Wilson 2018, 3). Another way is to introduce student-led live coding (Gaspar & Langevin 2007), which exposes the students' thought process for the educator to provide feedback on.

Students might ask that live coding sessions be recorded. While seemingly an good idea, educators must be aware that recording these sessions is making students increasingly less interactive (Gaspar & Langevin 2007). However, they do provide students with an opportunity to revisit, scrutinize, or catch up on the material discussed in class. Educators must also be aware that recorded live coding sessions cannot be substituted for carefully planned and pre-recorded instructional videos.

16: Build on existing graphic design history

Graphic design has a substantial body of knowledge developed and accumulated over centuries. Madsen (2016) has advocated that this knowledge not be ignored, but rather understood in order to be deliberately bent later. Graphic design students attend design schools to learn the theory and methods of the trade. It is vital that Creative Coding courses manage to latch onto this to be perceived as vehicles for students to further develop and extend their graphic design skills. That is why "artistic" learning activities that do not incorporate the established theory, models, and methods of the graphic design field are thought of as "fun-but-useless," detached exercises with little to no value to their continuing education (see #1).

The tradition and legacy of graphic design should be honored in Creative Coding courses. Referring to canonical theories, models, and methods also referenced in other classes makes the students realize that what they learn in the course of their study can also be applied in the medium of code. Consider the topic of color. Simply because a given programming language requires colors to be defined in hexadecimal notation or described using an HSB-model, does not eliminate the need to connect computationally defined colors to centuries of established color theory. *Au contraire*, a possible assignment regarding color might concern the transfer of traditional non-digital color models into the computational medium, sparking discussions about additive and subtractive properties of paper versus screens.

Throughout history, the making of custom design tools has been a regular component of the graphic design trade (Hansen 2012). By referencing this tradition in Creative Coding courses,

educators can encourage and nurture the students' desire to continue this tradition with digital media.

Finally, reflecting back on the history of graphic design enables Creative Coding to be viewed as a contemporary pinnacle of the trade. This helps educators situate code literacy and programming skills as mandatory and indispensable for students to wish to exert any influence on the future evolution of the graphic design trade.

17: Show commonalities between graphic design and programming

Despite ontological discrepancies, graphic design and programming have many commonalities (Tober 2012a; Shim 2016a). Throughout history, procedural "design systems" have governed and guided the production of visual communication. Recent canonical examples are the works of Gerstner (1964, 1972), Müller-Brockmann (1981), and Kapitzski (1980), whose systemic approach to graphic design closely mimics the sequential and rule-based execution of computer software, thereby highlighting the connection between programming and design and between coding and designing. Also arguing for the inherent connection between the two disciplines is Stiny (2001), who has considered "seeing" as a form of "visual calculating," thereby implying that some sort of "computation" takes place.

When framing Creative Coding courses, educators should mention that the courses are not merely designed to make students code-literate. At a meta level, courses are meant to introduce a way of thinking that enables visual ideas to be explicitized and formalized, with the aim of enlisting the computer as a tool in the formation and exploration of design artifacts. Accomplishing this can be greatly aided by introducing computational thinking, a set of problem-solving methods recently popularized by Wing (2006; 2010; 2014). Students should be encouraged to adopt computational thinking and backpropagate this knowledge in the way they conceive and execute graphic design. While programming languages may come and go, acquiring a generalized and platform-agnostic way of thinking computationally about graphic design will help students transition into meta-designers (Madsen 2015), i.e., designers who are as visually talented as they are technically proficient and to whom programming is a natural medium for creating modern dynamic, non-linear, and procedural design products.

18: Introduce programming concepts relevant to computational graphic design

Picking up from #16, it is important to realize that computation, and its derived influence on workflows, aesthetics, and the designer's own ontological perspective, represents a paradigm shift in graphic design as a discipline. The continued development of the graphic design discipline relies on curious students who challenge existing assumptions, methods, tools, and *modi operandi*. As educators, we carry an obligation to lead students into uncharted territory by introducing new concepts and ideas that inspire their quest into discovering what is possible to achieve through computation. This involves investigating the aesthetics created using dedicated computational methods (i.e., methods that involve far too complex and numerous calculations to perform by hand, rather than automated versions of tasks otherwise carried out by hand by the designer).

Creative Coding courses should, therefore, introduce programming concepts that are relevant for computationally assisted graphic design. These concepts can include algorithms that deal with spatial issues (e.g., dividing, distributing, packing, filling, aligning, intersecting), explore a pre-made set of building blocks (combinatorics, permutations), or create complex patterns out of simple rule-based systems (automaton, L-systems, context-free grammars). Some emulate natural phenomena (flocking behavior, vectors/gravity, kinematics, harmonious motion), others produce visual output (ASCII-art, fractals), retrieving/parsing/modifying data (fetching data from external API's, using XML/JSON data

as input, reading sensors from connected hardware, webcams, computer vision, Kinect/Leap/VR) and some are even meant to corrupt, bend, and break data (glitch, datamoshing, databending).

These algorithms can be provided as pre-made elements (#21) because the aim of the assignment is not to have each student implement, for example, a Poisson Disc Sampling algorithm from scratch, rather it is to explore what visual output can be made by the algorithm. However, some algorithms are surprisingly simple to implement, and purposely letting students experience what amazing complexity can be achieved with a few lines of code is beneficial to their inclination to explore other algorithms or even make their own.

To summarize: Assignments should introduce students to post-digital computational aesthetics. Therein lies a great justification for accepting programming as a unique and valuable addendum to their skillset.

19: Make programming concepts fit graphic design

Contemporary Creative Coding courses tend to make graphic design topics fit into a cognitive and conceptual frame originating in computer science as well as the syntactical affordances of the chosen programming environment/language (Hansen 2018b). Being forced to restrict and recast their domain-specific knowledge to fit the confines and limitations imposed by a programming environment has no positive impact on the graphic design students' view on the usefulness of programming in their existing workflow. Programming should be perceived as an extension of their skills, not as a limitation.

Creative Coding courses must be planned utilizing a *design-first* perspective. This involves letting graphic design praxis dictate what is needed and then choosing (and possibly adapting) the computing concepts accordingly. Also, if features fundamental to graphic design production are missing from the chosen programming environment, educators should make an effort to either 1) provide an explanation as to how the students can implement it themselves or 2) provide a utility as a pre-built element to give students the functionality they expect. Otherwise, they risk adjusting and limiting their thinking and ways of working to match the affordances of the programming environment, potentially forgetting that their skills obtained as graphic designers extend way beyond what they are capable of creating as programmers.

20: Make assignments "hard fun"

Many students enter programming courses with the preconceived notion that programming is "hard and boring" (Repenning 2017).

Turning "boring" into "exciting" is an affective challenge (Repenning 2017). Students become more engaged if they are given the opportunity to program objects that matter to them (see #2). Graphic design students have no passionate interest in understanding the construct of a double nested for-loop, but when it is shown in a contextualized example (see #11) where it is used to create a two-dimensional grid of objects, suddenly the same construct can excite the student. Appealing to the students' emotions by showing the exciting prospect of what they can achieve when they learn the objective of the assignment provides them with a worthwhile reason to invest the monumental effort needed to learn programming.

Turning "hard" into "easy" is a cognitive challenge (Repenning 2017). To the majority of graphic designers, programming is a totally new and unfamiliar domain. They will face a steep learning curve, and despite all instructional and pedagogical efforts by the educator to reduce the incline, students will need to work hard to achieve the desired learning outcome. There is no quick shortcut to go from "hard" to "easy"—just practice, practice, and more practice. It is important to address this in class; so

too the variations in the pace by which students progress as students tend to compare their own progress with that of their classmates. This, however, is an unfortunate habit as all students learn differently (Hansen 2018a). Explicitly explaining this can help students relax when they see what they believe to be "hard" is already being solved by others.

Students will inarguably have a perception of learning to program as being "hard" for a prolonged period of time (cognitive challenge). This is fine as long as the assignments are "exciting" or "fun" (affective challenge). The constellation of hard and fun is also discussed by Papert (quoted in Martinez & Stager 2019): *"We learn best and we work best if we enjoy what we are doing. But fun and enjoying doesn't mean 'easy.' The best fun is hard fun. Our sports heroes work very hard at getting better at their sports. The most successful carpenter enjoys doing carpentry. The successful businessman enjoys working hard at making deals."* Equally, Creative Coding educators should strive to have the students enjoy programming despite it being cognitively taxing.

To ensure that "hard" is not perceived as "impossible," it is important that assignments are solvable by accounting for the students' level and knowledge. This can be done by offering pre-made elements (#21) or making the assignment available in different levels (#23). Also, educators should try to establish an environment where failure to successfully complete an assignment is accepted, as long as a valid attempt was made.

21: Offer pre-made elements

It is a relic of a bygone era to assume that all assignments should begin with an empty editor and that students should write all the code themselves (Brown & Wilson 2018, 4). Just as training wheels help children keep their balance as they learn to ride a bike, so too can graphic designers benefit from a little support in their effort towards learning programming. One way is to offer pre-made elements in the form of visual assets (images, video, fonts), code (libraries, classes, snippets, templates, boilerplates), or math/algorithms (formulas, pseudo-code algorithms).

Pre-made and well-documented code, whether partially complete or fully written, provided by the educator can detract from the students' temptation to copy/paste code found on social code sharing platforms such as Stackoverflow, openProcessing, or Codepen without thinking about the logic behind the snippet. Furthermore, pre-made code can relieve students of the mental burden of first having to implement a given algorithm before they can use it. Rather, the algorithm should be provided as pre-made code, and graphic design students should be allowed to explore *what it can do* as a way to stimulate their subsequent inquiry into *how it works*. Finally, the use of pre-made code somehow also mimics the "messy" way students will work with code in real life, scavenging and reusing available code fragments from other programs in a bricolage style of programming (Guo 2013).

Offering pre-made visual assets can alleviate the risk of students spending too much time on issues (e.g., retracing logos, choosing fonts, mixing palettes) not directly related to learning programming, a strategy purposely employed by Hansen (2017).

22: Use interactive demonstrations

Regular analog textbooks (e.g., Vantomme 2012; Gradwohl 2013; Fry, Reas, & Maeda 2007; McCarthy, Fry, & Reas 2015) are used in many Creative Coding courses; however, research by Palmer (2011) indicates that it is difficult to correlate written code in books to visual results on the screen. A better way to help students understand a programming concept is by offering interactive demonstrations as cognitive support. In Creative Coding courses, these interactive demonstrations typically consist of pre-written editable code that is continuously executed to reflect any changes made. Using such

interactive demonstrations genuinely engages students in both active (playing with it) and reflective (thinking about it) processes, both of which, according to constructionist learning theory (Papert & Harel 1991; Kafai & Resnick 1996; Kafai 2005), are necessary for learning to take place most efficiently.

Interactive demonstrations using Donald Schön's notions of *reflection-in-action* and *reflection-on-action* (1983) establish a safe sandbox that allows students to actively modify code firsthand and evaluate the consequences using immediate visual feedback, which provides students with the opportunity to reflect on what they are doing while they are doing it (i.e., reflection-in-action). Supporting the interactive demonstration with follow-up questions encourages students to reflect on what they just did and what they experienced (i.e., reflection-on-action). These questions might be: What were you doing? Why were you doing that? What was the outcome? Can you explain what was happening? Why/how is this relevant or important to you in your work?

Some contemporary Creative Coding courses (e.g., Bakse 2018; Belluscio 2017) and interactive textbooks (Shiffman 2012) provide a rich supply of interactive examples—editable directly in a web browser—as a way for students to directly experience the functions and mechanics of the code through direct manipulation and immediate visual feedback (see #13).

In short, letting students interact with a demonstration and think about their experience afterwards is beneficial to their acquisition of new knowledge.

23: Introduce assignments with different difficulty levels

Graphic design students learn at different paces and in different ways (Hansen 2018a). This suggests the need for differentiated instruction to assure that students feel confident and able to accomplish the tasks given to them. One way to do so is by introducing assignments with varying difficulty levels. This will make weak students feel supported and strong students feel challenged (see #20). Students should themselves decide which level they want to undertake. Students still struggling with basic issues need scaffolding, while students who have gained a better understanding should also find the assignment challenging in order to stay motivated. Two levels (e.g., "novice" and "competent") will normally suffice, but larger classes might need be provided with an additional level (e.g., "proficient") to account for the larger spread between weak and strong groups of students. Support sessions should be planned to match different levels, and commonly experienced problems (technical, cognitive, conceptual) should be addressed by that group without having weaker/stronger students spend their time sitting in on discussions that they perceive as too complex or too simple (see #20).

A way to do this relates to #21. When setting an assignment in, say, three levels, pre-made elements in the shape of boilerplate code can be offered to the novice, partial code to the intermediate learner, and no code to the advanced programmer. Also, novices should not worry about interfaces, while expert programmers can be challenged to provide an intuitive interface to interact with their product. Issues such as performance and stability should not be of concern to the novices but made mandatory for the advanced programmers.

Knowing that the challenge level of the assignment can be chosen to match their competencies will help reduce the stress and anxiety associated with being asked to complete a programming assignment. Also, to non-novices, being able to complete an assignment without taking any of the help offered to lower levels instills a sense of achievement and accomplishment in the student, further motivating him to push forward.

24: Spend more time helping, less time lecturing

Learning to program is a path riddled with obstacles: mysterious bugs, unescapable loops, missing libraries, cumbersome syntax, and convoluted control structures. Becoming stuck is unavoidable, but students grow frustrated and become demotivated if their programs do not work as intended. However, in keeping with constructionist teaching methodology, it is not the educator's role to be the tow truck that arrives to winch up the car from the ditch and then leave. Instead, educators must help students get themselves unstuck. A number of tactics to achieve this can be considered, as explained in the following paragraphs.

Most importantly, educators must prioritize teaching general problem-solving and debugging strategies to allow students to help themselves. While these strategies can be taught theoretically in lectures, they must be practiced and honed through hands-on experience with support readily available. Students should therefore be given ample time in class to work on exercises, assignments, and projects with the educator close by to assist when a problem occurs. However, if the educator tries to take adequate time to assist each student to solve their problem by themselves, a bottleneck can quickly form, resulting in a queue of students waiting for help. Possible ways to combat this include enlisting teaching assistants, solving similar and frequently occurring problems in plenum, and encouraging students to help each other.

To free up time for helping, new teaching methodologies brought about by the use of technology in the classrooms should also be considered. For example, the concept of blended learning (Friesen 2012) combines traditional classroom methods with digital learning resources (see #15, #22). By using a combination of digital instruction and one-on-one time, students can work on their own with new concepts, which frees up educators to circulate and support individual students who may need individualized attention. A specific blended learning instructional strategy is the flipped classroom (Flipped Learning Network 2014). A flipped classroom reverses the traditional learning environment. It moves activities that may traditionally have been considered homework into the classroom, and similarly moves traditional lectures, teaching activities, and instructional content out of the classroom—most often online. Having transformed his introductory Creative Coding course to fit the flipped classroom model, Shiffman (2018) has made his lectures available online as mandatory pre-class preparation, freeing up valuable time spent on questions, demonstration of homework, workshops, and wrap-ups.

When designing assignments, typical problems that might occur when trying to complete the given task can be addressed and (depending on difficulty level, see #23) a remedy provided to ensure that stuck students can quickly solve the problem and move on.

Students should also be taught the appropriate etiquette of how to ask for help (e.g., how to describe a problem, including a code example, listing relevant system specifications) and where to ask for help (e.g., in class, course site, from educator during office hours, programming forums, or social platforms).

FINAL REMARKS

This dissertation has investigated the overlapping intersection of graphic design, programming, and pedagogy. In addition to three individual studies and a summarizing discussion, the dissertation also contributes valuable knowledge in the shape of 24 heuristics specifically aimed at design educators who plan to teach Creative Coding to graphic designers.

Code as a creative medium and material holds a vast and yet barely explored potential and must be included in the disciplinary epistemology if graphic design is to stay current and relevant. As programming and graphic design continue their convergence and form an even stronger symbiotic relationship, it naturally calls for computation to be integrated into graphic designers' thinking and doing. By learning to program, graphic designers can become *creators* of unique and custom-made digital design tools and not just *users* of ready-made, industry-standard tools provided by the software industry. By learning to program, graphic designers can reap the benefits of adding computation to all stages of their workflow. By learning to program, graphic designers can acquire a new perspective and appreciation of their trade as they explicate their tacit knowledge. By learning to program, graphic designers can understand how software works and adapt the way they conceptualize their designs to fit the computational media. By learning to program, graphic designers can push the disciplinary boundaries forward as they infuse the essence of their trade with the power of technology. Finally, and perhaps most importantly, by learning to program, graphic designers can also learn when programming is NOT the answer to their needs and their computer is better left switched off.

If this dissertation in any possible way or form contributes to the development of bespoke Creative Coding programming courses in future Graphic Design education, my four years spent as *designer-become-researcher* has been worthwhile.



Stig Møller Hansen
Roskilde, February 2019

GLOSSARY

Algorithmic Design: A design process that involves specific procedures or formulas to generate output.

Blended Learning: An educational approach that combines online educational materials with traditional place-based classroom methods and independent study in a new hybrid teaching methodology.

Coding: The activity of translating an algorithm to a specific programming language. This requires an understanding of the syntactical aspects of the language. It also requires direct interaction with the programming environment.

Computational Graphic Design: The act of using calculations and instructions, typically performed using computers, as part of a graphic design process or workflow.

Computational Thinking: The thought processes involved in expressing solutions as computational steps or algorithms that can be carried out by a computer. Originating in computer science but universally applicable across disciplines.

Creative Coding: A type of computer programming in which the goal is to create something expressive instead of something functional. Mitchell & Bown (2013, 143) define Creative Coding as “*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.*”

Debugging: The routine process of identifying and resolving errors within a computer program. Can be done manually or supported by debugging tools.

Demoscene: A computer art subculture focused on producing demos (i.e., self-contained computer programs that produce audio-visual presentations). The purpose of a demo is to show off programming, visual art, and musical skills.

Flipped Classroom: An instructional strategy and type of blended learning that reverses the traditional learning environment. Students are introduced to content at home (usually through online media) and practice working through it in the classroom.

Generative Design: An iterative design process in which an initial state of a design system is changed either computationally or by input from the designer to create seemingly endless variations.

GUI (Graphical User Interface): A form of user interface that allows users to interact with computers through graphical icons, windows, mouse cursors, and other visual representations instead of typing commands or navigating text-only screen.

IDE (Integrated Development Environment): A software application used by programmers for software development. An IDE normally consists of a source code editor, build tools, and a debugger.

Instructional Design: The practice of creating learning experiences that make acquisition of knowledge and skills more efficient, effective, and appealing.

Meta-design: An approach to design in which no specific artifact is designed; it is rather a collection of design rules and actions, which, when formalized (often using software) and executed, is capable of automatically generating final artifacts without much effort.

Parametric Design: A design process driven by parameters where properties of existing models are modified to vary multiple outcomes.

Programming: The activity of translating a problem-solving approach to algorithms using programming constructs. This does not require any knowledge of specific syntax. Also, programming does not necessarily require interacting with a programming environment.

Recursion: The process in which a function calls itself.

Tertiary Education: (also referred to as *postsecondary education*) An educational level focusing on learning endeavors in specialized fields. Includes both public and private universities, academies, seminaries, colleges, and vocational schools. Culminates in the receipt of certificates, diplomas, or academic degrees.

Unicorns: Nickname for a new generation of designers who excel equally in both designing and programming.

FIGURES AND TABLES

Extended summary

Figure 1.1: The study's subject field, research fields, and position of the author	15
Figure 3.1: Design research as a subset of design practice at large (Faste & Faste 2012).....	46
Figure 3.2: The study's specific research questions positioned in Fallman's (2008) Interaction Design Research Triangle.....	47
Figure 3.3: Schematic overview of research activities in the study.....	53
Table 3.1: Types of research in creative arts and design.....	45
Table 3.2: Main rationales and benefits of Mixed Methods Research.....	49
Table 3.3: Summary of research design chosen to answer each SRQ	52
Table 4.1: Progression of temporalities in the study's papers.....	56

Paper 1

Figure 1: Populated matrix, providing an overview of the analyzed courses.....	59
Table 1: Search query matrix.....	58
Table 2: Average order in which computer science topics and graphic design topics were taught	60

Paper 2

Figure 1: Active-Reflective distribution of the respondents.....	69
Figure 2: Visual-Verbal distribution of the respondents.....	69
Figure 3: Sensing-Intuitive distribution of the respondents.....	69
Figure 4: Sequential-Global distribution of the respondents.....	69
Table 1: Strengths of preferences.....	69
Table 2: Learning Style preferences found in this study compared to those reported in similar studies.....	70

Paper 3

Figure 1: Schematic overview of the deconstruction/reconstruction method.....	78
Figure 2: A selection of specimens suitable as material for the method.....	79
Figure 3: Poster by Enzo Mari (1963).....	79
Figure 4: Alternative versions obtained by tweaking variables.....	81
Figure 5: Alternative versions obtained by modifying code and tweaking variables.....	81
Figure 6: The collection of chosen specimens taped to the blackboard in the studio	82
Figure 7: A student's attempt at recreating the original specimen using code, and his subsequent experiments modifying the identified variables and the code itself to produce radically different versions	83

REFERENCES

- Abeysekera, Lakmal & Dawson, Phillip. 2015. "Motivation and Cognitive Load in the Flipped Classroom: Definition, Rationale and a Call for Research." *Higher Education Research & Development* 34 (1): 1–14.
- Ackermann, Edith. 2001. "Piaget's Constructivism , Papert's Constructionism: What's the Difference?" *Future Learning Group Publication*.
- AIQA. 2017. "What Is Graphic Design?" American Institute of Graphic Arts. Accessed from <https://www.aiga.org/guide-what-is-graphic-design>.
- Akker, Jan van den, Gravemeijer, Koeno, McKenney, Susan & Nienke, Nieveen. 2006. *Educational Design Research*. 1st Edition. Routledge.
- Alesandrini, Kathryn & Larson, Linda. 2002. "Teachers Bridge to Constructivism." *The Clearing House* 75 (3): 118–121.
- Allen, W. Clayton. 2006. "Overview and Evolution of the ADDIE Training System." *Advances in Developing Human Resources* 8 (4): 430–441.
- Amiri, Famarz. 2011. "Programming as Design: The Role of Programming in Interactive Media Curriculum in Art and Design." *International Journal of Art and Design Education* 30 (2): 200–210.
- Amresh, Ashish, Carberry, Adam R. & Femiani, John. 2013. "Evaluating the Effectiveness of Flipped Classrooms for Teaching CS1." *Proceedings of the Frontiers in Education Conference, FIE, 733–735*.
- Andersen, Peter Bøgh, Bennedsen, Jens, Brandorff, Steffen, Caspersen, Michael E. & Mosegaard, Jesper. 2003. "Teaching Programming to Liberal Arts Students – a Narrative Media Approach." In *Proceedings of the Conference on Innovation and Technology in Computer Science Education*, 8:109–113. ACM Press.
- Anderson, Eike Falk. 2018. "Turtle Fractals and Spirolaterals: Effective Assignments for Novice Graphics Programmers." In *Proceedings of the 39th Eurographics Conference 2018*, 39–42. The Eurographics Association.
- Archer, Bruce. 1995. "The Nature of Research." *Co-Design, Interdisciplinary Journal of Design*, no. January: 6–13.
- Artut, Selcuk. 2017. "Incorporation of Computational Creativity in Arts Education: Creative Coding as an Art Course." In *ERPA International Congresses on Education 2017*, edited by E. Masal, I. Önder, S. Beşoluk, H. Çalışkan, and E. Demirhan. Vol. 37. EDP Sciences.
- Atwood, Jeff. 2012. "Please Don't Learn to Code." Accessed from <https://blog.codinghorror.com/please-dont-learn-to-code/>.
- Avital, Tsion. 2017. *The Confusion between Art and Design*. 1st Edition. Vernon Press.
- Bain, John D., Ballantyne, Roy, Packer, Jan & Mills, Colleen. 1999. "Using Journal Writing to Enhance Student Teachers' Reflectivity During Field Experience Placements." *Teachers and Teaching* 5 (1): 51–73.
- Bakse, Justin. 2018. "Hello, Comp Form!" Accessed from <http://compform.net/>.
- Barab, Sasha & Squire, Kurt. 2004. "Design-Based Research: Putting a Stake in the Ground." *Journal of the Learning Sciences* 13 (1): 1–14.
- Belluscio, Antonio. 2017. "Computer Graphics Con P5.Js." Accessed from <https://www.exframes.net/cg-p5js/>.
- Bender, Diane M. & Vredevoogd, Jon D. 2006. "Using Online Education Technologies to Support Studio Instruction." *Educational Technology & Society* 9 (4): 114–122.

- Bennedsen, Jens. 2008. "Introduction to Part I." In *Reflections on the Teaching of Programming: Methods and Implementations*, edited by Jens Bennedsen, Michael E. Caspersen, and Michael Kölling, 3–5. Springer-Verlag.
- Benoit, Francis. 2017. "The Web Browser as a Tool: A Programmatic Approach to Graphic Design on the Web." York University, Toronto, Canada.
- Biesta, Gert & Burbules, Nicholas C. 2003. *Pragmatism and Educational Research*. 1st Edition. Rowman & Littlefield Publishers.
- Blauvelt, Andrew. 2008. "Towards Relational Design." DesignObserver. Accessed from <http://designobserver.com/feature/towards-relational-design/7557/>.
- Blauvelt, Andrew. 2011. "Tool (Or, Post-Production for the Graphic Designer)." In *Graphic Design: Now in Production*. Walker Art Center.
- Blum, Bruce I. 1996. *Beyond Programming: To a New Era of Design*. 1st Edition. Oxford University Press.
- Brennan, Karen & Resnick, Mitchel. 2012. "New Frameworks for Studying and Assessing the Development of Computational Thinking." In *2012 Annual Meeting of the American Educational Research Association*, 1–25.
- Brewer, Jess H. 1998. "So What Is a HyperTextBook?" Accessed from <http://jick.net/~htb/HTB/HTB3/>.
- Bridle, James. 2012. "#sxaesthetic." booktwo.org. Accessed from <http://booktwo.org/notebook/sxaesthetic/>.
- Briggs-Myers, Isabel, McCaulley, Mary H., Quenk, Naomi L. & Hammer, Allen L. 1998. *MBTI Manual: A Guide to the Development and Use of the Myers-Briggs Type Indicator*. 3rd Editio. Consulting Psychologists Press.
- Brown, Bruce, Gough, Paul & Roddis, Jim. 2004. "Types of Research in the Creative Arts and Design." Discussion Paper.
- Brown, Neil C. C. & Wilson, Greg. 2018. "Ten Quick Tips for Teaching Programming." *PLoS Computational Biology* 14 (4): 1–8.
- Buchanan, Richard. 1992. "Wicked Problems in Design Thinking." *Design Issues* 8 (2): 5–21.
- Buchanan, Richard. 2001. "Design Research and the New Learning." *Design Issues* 17 (4): 3–23.
- Buchanan, Richard. 2007. "Strategies of Design Research: Productive Science and Rhetorical Inquiry." In *Design Research Now*, edited by Ralf Michel, 55–66. Birkhäuser Verlag AG.
- Byrne, Pat & Lyons, Gerry. 2001. "The Effect of Student Attributes on Success in Programming." *ACM SIGCSE Bulletin* 33 (3): 49–52.
- Cáceres, Carlos H. 2017. "Re-Educating The Reflective Practitioner: A Critique of Donald Schön's Reflective Practice and Design Education For Engineering."
- Cannaerts, Corneel. 2016. "Coding as Creative Practice." *Ecaade 2016: Complexity & Simplicity, Vol 1* 1: 397–404.
- Capretz, Luiz Fernando. 2002. "Implications of MBTI in Software Engineering Education." *ACM SIGCSE Bulletin* 34 (4): 134–137.
- Carlsson, Anders. 2009. "The Forgotten Pioneers of Creative Hacking and Social Networking—Introducing the Demoscene." In *Re:Live Media Art Histories 2009*, 16–20.
- Carmen, Luke. 2000. "Cyberschooling and Technological Change: Multiliteracies for New Times." In *Multiliteracies: Literacy Learning and the Design of Social Futures*, edited by Bill Cope and Mary Kalantzis, 69–91. Macmillan.
- Carter, Adam S. & Hundhausen, Christopher D. 2011. "A Review of Studio-Based Learning in Computer Science." *The Journal of Computing Sciences in Colleges* 27 (1): 105–111.

- Caspersen, Michael E. & Christensen, Henrik Bærbak. 2000. "Here, There and Everywhere - on the Recurring Use of Turtle Graphics in CS1." In *ACSE '00 Proceedings of the Australasian Conference on Computing Education*, 34–40.
- Chng, Sue Inn. 2018. "International and Multidisciplinary Perspectives Incorporating Reflection into Computing Classes: Models and Challenges." *Reflective Practice* 19 (3): 358–375.
- Christensen, Kimberly & West, Richard E. 2013. "The Development of Design-Based Research." In *Foundation of Learning and Instructional Design Technology*. University of Georgia.
- Clohessy, Deanna L. 2011. "Creating Visual Solutions: Using Creative Problem Solving Techniques in Graphic Design." State University of New York.
- Cobb, Paul, Confrey, Jere, DiSessa, Andrea, Lehrer, Richard & Schauble, Leona. 2003. "Design Experiments in Educational Research." *Educational Researcher* 32 (1): 9–13.
- Coffield, Frank, Moseley, David, Hall, Elaine & Ecclestone, Kathryn. 2004. *Learning Styles and Pedagogy in Post-16 Learning. A Systematic and Critical Review*. Learning and Skills Research Centre. Learning and Skills Research Centre.
- Collins, Allan, Brown, John Seely & Newman, Susan E. 1989. "Cognitive Apprenticeship: Teaching the Craft of Reading, Writing, and Mathematics." In *Knowing, Learning, and Instruction*, edited by L. B. Resnick, 453–494. Lawrence Erlbaum Associates.
- Connolly, Cornelia, Murphy, Eamonn & Moore, Sarah. 2009. "Programming Anxiety Amongst Computing Students — A Key in the Retention Debate?" *Control* 52 (1): 52–56.
- Cooper, Alan. 2017. "Should Designers Code? No, Part 1, 2 & 3." medium.com. Accessed from <https://medium.com/@MrAlanCooper/should-designers-code-f7b745b8cd03>.
- Cousin, Glynis. 2006. "An Introduction to Threshold Concepts." *Planet* 17 (1): 4–5.
- Creswell, John W. 2009. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. SAGE Publications.
- Creswell, John W. 2014. *A Concise Introduction to Mixed Methods Research*. 1st Edition. SAGE Publications.
- Creswell, John W. & Plano Clark, Vicki L. 2007. "Choosing a Mixed Methods Design." In *Designing and Conducting Mixed Methods Research*, 62–79. SAGE Publications.
- Cross, Nigel. 1982. "Designerly Ways of Knowing." *Design Studies* 3 (82): 221–227.
- Cross, Nigel. 2007. "From a Design Science to a Design Discipline: Understanding Designerly Ways of Knowing and Thinking." In *Design Research Now*, edited by R. Michel, 41–54. Birkhäuser.
- Crouwel, Wim. 1967. "New Alphabet." Accessed from <https://www.moma.org/collection/works/139322>.
- Daichendt, G. James. 2010. "The Bauhaus Artist-Teacher: Walter Gropius's Philosophy of Art Education." *Teaching Artist Journal* 8 (3): 157–164.
- Davis, Meredith. 1998. "How High Do We Set the Bar for Design Education." In *The Education of a Graphic Designer*, edited by Steven Heller, 25–30. Allworth Press.
- Denning, Peter J. 2004. "The Field of Programmers Myth." *Communications of the ACM* 47 (7): 15–20.
- Denning, Peter J. 2017. "Remaining Trouble Spots with Computational Thinking." *Communications of the ACM* 60 (6): 33–39.
- Dewey, John. 1913. *Interest and Effort in Education*. 1st Edition. Houghton Mifflin.
- Dewey, John. 1933. *How We Think: A Restatement of the Relation of Reflective Thinking to the Educative Process*. Henry Regnery Co.
- Dewey, John. 1934. *Art as Experience*. Minton, Balch & Company.
- DiSessa, Andrea A. & Cobb, Paul. 2004. "Ontological Innovation and the Role of Theory in Design Experiments." *Journal of the Learning Sciences* 13 (1): 77–103.

- Dodgson, Neil A. & Chalmers, Andrew. 2017. "Designing a Computer Graphics Course for First Year Undergraduates." In *Proceedings of the 38th Eurographics Conference 2017*, 9–15. The Eurographics Association.
- Dorn, Brian & Guzdial, Mark. 2006. "Graphic Designers Who Program as Informal Computer Science Learners." *Proceedings of the 2006 International Workshop on Computing Education Research*, 127–134.
- Dorn, Brian, Tew, Allison Elliott & Guzdial, Mark. 2007. "Introductory Computing Construct Use in an End-User Programming Community." In *VLHCC '07 Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing*, 27–32. IEEE Computer Society.
- Dorst, Kees. 2010. "The Nature of Design Thinking." In *Symposium Proceedings of DTRS8: Interpreting Design Thinking*, edited by Kees Dorst, Susan Stewart, Ilka Staudinger, Bec Paton, and Andy Dong, 131–139. University of Technology Sydney.
- Downton, Peter. 2003. *Design Research*. RMIT University Press.
- Doyle, Louise, Brady, Anne-Marie Marie & Byrne, Gobnait. 2016. "An Overview of Mixed Methods Research – Revisited." *Journal of Research in Nursing* 21 (8): 623–635.
- Drake, Peter & Sung, Kelvin. 2011. "Teaching Introductory Programming with Popular Board Games." In *SIGCSE '11 Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, 619–624. ACM.
- Dubberly, Hugh. 1990. "An Introduction to Hypermedia and the Implications of Technology on Graphic Design Education." In *Proceedings of the Graphic Design Education Association (GDEA) Annual National Symposia 1989-1990*.
- Dwiggins, William Anderson. 1922. "New Kind of Printing Calls for New Design." *Boston Evening Transcript*, August 29, 1922.
- Ericson, Barbara J., Guzdial, Mark & Morrison, Briana B. 2015. "Analysis of Interactive Features Designed to Enhance Learning in an Ebook." In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research - ICER '15*, 169–178.
- Ettinger, Linda E. 1988. "Art Education and Computing: Building a Perspective." *Studies in Art Education* 30 (1): 53–62.
- Faison, Brenda Smith. 1995. "Graphic Design Educators and Practitioners in Transition: From Traditional Tools and Applications to the Computer-Based Tools of Multimedia Design." The Ohio State University.
- Falkner, Katrina, Falkner, Nickolas, Szabo, Claudia & Vivian, Rebecca. 2016. "Applying Validated Pedagogy to MOOCs: An Introductory Programming Course with Media Computation." In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, 326–331. ACM Press.
- Fallman, Daniel. 2003. "Design-Oriented Human-Computer Interaction." In *Proceedings of the Conference on Human Factors in Computing Systems - CHI '03*, 225–232. ACM Press.
- Fallman, Daniel. 2008. "The Interaction Design Research Triangle of Design Practice, Design Studies, and Design Exploration." *Design Issues* 24 (3): 4–18.
- Fallman, Daniel. 2017. "Do You Need to Know How to Code to Be a Designer?" Accessed from <http://www.dfallman.com/journal/2017/3/17/do-you-need-to-know-how-to-code-to-be-a-designer>.
- Faste, Trygve & Faste, Haakon. 2012. "Demystifying 'Design Research': Design Is Not Research, Research Is Design." In *Proceedings of the IDSA Education Symposium 2012*.
- Fee, Samuel B. & Holland-Minkley, Amanda M. 2010. "Teaching Computer Science through Problems, Not Solutions." *Computer Science Education* 20 (2): 129–144.
- Feilzer, Martina Yvonne. 2010. "Doing Mixed Methods Research Pragmatically: Implications for the Rediscovery of Pragmatism as a Research Paradigm." *Journal of Mixed Methods Research* 4 (1): 6–16.

- Fekete, Alan, Kay, Judy, Kingston, Jeff & Wimalaratne, Kapila. 2000. "Supporting Reflection in Introductory Computer Science." *ACM SIGCSE Bulletin* 32 (1): 144–148.
- Felder, Richard M. & Brent, Rebecca. 2005. "Understanding Student Differences." *Journal of Engineering Education* 94 (1): 57–72.
- Felder, Richard M. & Silverman, Linda Kreger. 1988. "Learning and Teaching Styles." *Engineering Education*.
- Felder, Richard M. & Soloman, Barbara A. 1997. "Index of Learning Styles (ILS®) Questionnaire." North Carolina State University. Accessed from <https://www.webtools.ncsu.edu/learningstyles/>.
- Felder, Richard M. & Spurlin, Joni. 2005. "Applications, Reliability and Validity of the Index of Learning Styles." *International Journal of Engineering Education* 21 (1): 103–112.
- Fleischmann, Katja. 2013. "Big Bang Technology: What's next in Design Education, Radical Innovation or Incremental Change?" *Journal of Learning Design* 6 (3): 1–17.
- Fleischmann, Katja. 2018. "Online Design Education: Searching for a Middle Ground." *Arts and Humanities in Higher Education*, March, 1–22.
- Fleming, Neil D. & Mills, Colleen. 1992. "Not Another Inventory, Rather a Catalyst for Reflection." *To Improve the Academy* 11 (1): 137–155.
- Flipped Learning Network. 2014. "Definition of Flipped Learning." FLIP Learning. Accessed from <https://flippedlearning.org/definition-of-flipped-learning/>.
- Flores, René Pedroza. 2016. "Personality Dominant Values in Graphic Design Students in Their Educational Practice." *Higher Education Studies* 6 (1): 101–109.
- Florin, Fabrice. 1985. "Hackers: Wizards of the Electronic Age." Public Broadcasting Service.
- Forte, Andrea & Guzdial, Mark. 2004. "Computers for Communication, Not Calculation: Media as a Motivation and Context for Learning." In *37th Annual Hawaii International Conference on System Sciences*. IEEE.
- Forte, Andrea & Guzdial, Mark. 2005. "Motivation and Nonmajors in Computer Science: Identifying Discrete Audiences for Introductory Courses." *IEEE Transactions on Education* 48 (2): 248–253.
- Frankel, Lois & Racine, Martin. 2010. "The Complex Field of Research: For Design, through Design, and about Design." In *Proceedings of the 43th Design Research Society (DRS) International Conference*, 518–529. Université de Montréal.
- Frayling, Christopher. 1993. "Research in Art and Design." *Royal College of Art Research Papers* 1 (1): 1–5.
- Freyermuth, Sherry S. 2016. "Coding As Craft: Evolving Standards in Graphic Design Teaching and Practice." *Plot(s) Journal of Design Studies* 3: 58–71.
- Friedman, Ken. 2003. "Theory Construction in Design Research Criteria: Approaches, and Methods." *Design Studies* 24 (6): 507–522.
- Friesen, Norm. 2012. "Report: Defining Blended Learning."
- Fry, Ben, Reas, Casey & Maeda, John. 2007. *Processing: A Programming Handbook for Visual Designers and Artists*. 1st Edition. The MIT Press.
- FutureLearn. 2014. "Creative Coding - Online Course." Accessed from <https://www.futurelearn.com/courses/creative-coding>.
- Gage, Nathaniel Lees. 1989. "The Paradigm Wars and Their Aftermath: A 'Historical' Sketch of Research on Teaching Since 1989." *Educational Researcher* 18 (7): 4–10.
- Galanter, Philip. 2009. "Thoughts on Computational Creativity." In *Proceedings of the Computational Creativity: An Interdisciplinary Approach*.

- Gaspar, Alessio & Langevin, Sarah. 2007. "Active Learning in Introductory Programming Courses through Student-Led 'Live Coding' and Test-Driven Pair Programming." In *EISTA 2007, Education and Information Systems, Technologies and Applications*.
- George, Susan E. 2002. "Learning and the Reflective Journal in Computer Science." *Australian Computer Science Communications* 24 (1): 77–86.
- Gerstner, Karl. 1964. *Programme Entwerfen*. 1st Edition. Arthur Niggli Verlag.
- Gerstner, Karl. 1972. *Kompendium Für Alphabeten: Systematik Der Schrift*. 1st Edition. Arthur Niggli Verlag.
- Gibbs, Graham. 1998. *Learning by Doing: A Guide to Teaching and Learning Methods*. Oxford Brookes University.
- Goldkuhl, Göran. 2011. "Design Research in Search for a Paradigm: Pragmatism Is the Answer." In *Practical Aspects of Design Science*, edited by M. Helfert and B. Donnellan, 84–95. Springer-Verlag.
- Gradwohl, Nikolaus. 2013. *Processing 2: Creative Coding Hotshot*. Packt Publishing Ltd.
- Graham, Paul. 2004. *Hackers and Painters*. O'Reilly Media Inc.
- Greenberg, Ira, Kumar, Deepak & Xu, Dianna. 2012. "Creative Coding and Visual Portfolios for CS1." In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education - SIGCSE '12*, 247–252.
- Grushka, Kath, McLeod, Julie Hinde & Reynolds, Ruth. 2005. "Reflecting upon Reflection: Theory and Practice in One Australian University Teacher Education Program." *Reflective Practice* 6 (2): 239–246.
- Gu, Xiaoqing, Wu, Bian & Xu, Xiaojuan. 2015. "Design, Development, and Learning in e-Textbooks: What We Learned and Where We Are Going." *Journal of Computers in Education* 2 (1): 25–41.
- Guglielmetti, Mark & McCormack, Jon. 2017. "Between Code and Culture: Developing a Creative Coding Massive Open Online Course." In *Teaching Computational Creativity*, edited by Michael Filimowicz and Veronika Tzankova, 1st ed., 193–209. Cambridge University Press.
- Guo, Philip. 2013. "Teaching Real World Programming." BLOG@CACM. Accessed from <https://cacm.acm.org/blogs/blog-cacm/159263-teaching-real-world-programming/fulltext>.
- Guzdial, Mark. 2003. "A Media Computation Course for Non-Majors." In *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education - ITiCSE '03*, 104–108.
- Guzdial, Mark. 2009. "Teaching Computing to Everyone." *Communications of the ACM* 52 (5): 31–33.
- Guzdial, Mark. 2010. "Does Contextualized Computing Education Help?" *ACM Inroads* 1 (4): 4–6.
- Guzdial, Mark. 2013. "Exploring Hypotheses about Media Computation." In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research - ICER '13*, 19–26. ACM Press.
- Guzdial, Mark. 2015a. *Learner-Centered Design of Computing Education: Research on Computing for Everyone*. 1st Edition. Morgan & Claypool Publishers.
- Guzdial, Mark. 2015b. "Top 10 Myths about Teaching Computer Science." *Communications of the ACM*. Accessed from <http://cacm.acm.org/blogs/blog-cacm/189498-top-10-myths-about-teaching-computer-science/fulltext#>.
- Hall, Ralph. 2013. "Mixed Methods: In Search of a Paradigm." In *Conducting Research in a Changing and Challenging World*, edited by Thao Le and Quynh Le, 71–78. Nova Science Publishers, Inc.
- Hannaford, Joey. 2012. "Changing Contexts in Graphic Design." In *Catch22: Eighth Annual UCDA Design Education Summit Abstracts & Proceedings*, 271–275. University & College Designers Association.
- Hansen, Stig Møller. 2012. "Den Grafiske Designer Som Værktøjsmager." Aarhus Universitet.

- Hansen, Stig Møller. 2017. "Deconstruction/Reconstruction: A Pedagogic Method for Teaching Programming to Graphic Designers." In *Proceedings of the 20th Generative Art Conference GA2017*.
- Hansen, Stig Møller. 2018a. "Assessing Graphic Designers' Learning Style Profile to Improve Creative Coding Courses." *Submitted to Eurographics 2019*.
- Hansen, Stig Møller. 2018b. "Mapping Creative Coding Courses: Towards Bespoke Programming Curricula in Graphic Design Education." *Submitted to Eurographics 2019*.
- Hardman, Paul. 2017. "Framing Design Education within the Contemporary Paradigm." In *Proceedings of 5th EIMAD Meeting of Research in Music, Arts and Design*, 1–15.
- Hausman, Jerome. 1991. "Computers, Video-Discs, and Art Teachers." *Art Education* 44 (3): 6–14.
- Heddy, Benjamin C. & Pugh, Kevin J. 2015. "Bigger Is Not Always Better: Should Educators Aim for Big Transformative Learning Events or Small Transformative Experiences?" *Journal of Transformative Learning* 3 (1): 52–58.
- Heller, Steven. 2002. *The Graphic Design Reader*. 1st Edition. Allworth Press.
- Heller, Steven & Womack, David. 2007. *Becoming a Digital Designer: A Guide to Careers in Web, Video, Broadcast, Game and Animation Design*. Wiley.
- Hemmendinger, David. 2010. "A Plea for Modesty." *ACM Inroads* 1 (2): 4–7.
- Hidi, Suzanne & Baird, William. 1986. "Interestingness - A Neglected Variable in Discourse Processing." *Cognitive Science A Multidisciplinary Journal* 10 (2): 179–194.
- Hjorth, Maria. 2017. "Strengths and Weaknesses of a Visual Programming Language in a Learning Context with Children."
- Hokanson, Brad. 2012. "The Design Critique as a Model for Distributed Learning." In *The Next Generation of Distance Education: Unconstrained Learning*, edited by L. Moller and J. B. Huett, 1st ed., 71–83. Springer Science & Business Media.
- Hokanson, Brad, Miller, Charles & Hooper, Simon. 2008. "A Contemporary Perspective for Innovation in Instructional Design." *TechTrends* 52 (6): 36–43.
- Honey, Peter & Mumford, Alan. 2000. *The Learning Styles Questionnaire: 80-Item Version*. Peter Honey.
- Hoxsey, Rich. 2003. "Code Dependency." *PRINT* 57 (5): 39–45.
- Humphrey, Watts S. 2000. "The Personal Software Proces (PSP)."
- Iskrenovic-Momcilovic, Olivera. 2017. "Choice of Visual Programming Language for Learning Programming." *International Journal of Computers* 2: 250–254.
- Johansson, Roger. 2007. "Are We Designers or Developers?" Accessed from https://www.456bereastreet.com/archive/200708/are_we_designers_or_developers/.
- Johns, Christopher. 1995. "Framing Learning through Reflection within Carper's Fundamental Ways of Knowing in Nursing." *Journal of Advanced Nursing* 22 (2): 226–234.
- Johnson, R. Burke & Onwuegbuzie, Anthony J. 2004. "Mixed Methods Research: A Research Paradigm Whose Time Has Come." *Educational Researcher* 33 (7): 14–26.
- Johnson, R. Burke, Onwuegbuzie, Anthony J. & Turner, Lisa A. 2007. "Toward a Definition of Mixed Methods Research." *Journal of Mixed Methods Research* 1 (2): 112–133.
- Jonas, Wolfgang. 2007. "Design Research and Its Meaning to the Methodological Development of the Discipline." In *Design Research Now*, 187–206. Birkhäuser Basel.
- Jones, Sue & Burnett, Gary. 2008. "Spatial Ability and Learning to Program." *Human Technology* 4 (May): 47–61.
- Jury, David. 2012. *Graphic Design before Graphic Designers: The Printer as Designer and Craftsman 1700-1914*. Thames & Hudson.

- Järvelä, Sanna & Renninger, K. Ann. 2014. "Designing for Learning: Interest, Motivation, and Engagement." In *The Cambridge Handbook of the Learning Sciences*, edited by R. Keith Sawyer, 2nd ed., 668–685. Cambridge University Press.
- Kafai, Yasmin B. 2005. "Constructionism." In *The Cambridge Handbook of the Learning Sciences*, edited by R. Keith Sawyer, 35–46. Cambridge University Press.
- Kafai, Yasmin B. & Resnick, Mitchel. 1996. *Constructionism in Practice: Designing, Thinking, and Learning in a Digital World*. Edited by Yasmin B. Kafai and Mitchel Resnick. 1st Edition. Routledge.
- Kalantzis, Mary & Cope, Bill. 2010. "The Teacher as Designer: Pedagogy in the New Media Age." *E-Learning* 7 (3): 200–222.
- Kapitzki, Herbert W. 1980. *Programmiertes Gestalten: Grundlagen Für Das Visualisieren Mit Zeichen*. 1st Edition. Verlag Dieter Gitzel.
- Kaplan, Andreas M. & Haenlein, Michael. 2016. "Higher Education and the Digital Revolution: About MOOCs, SPOCs, Social Media, and the Cookie Monster." *Business Horizons* 59 (4): 441–450.
- Kay, Jennifer S & Road, Mullica Hill. 2011. "Contextualized Approaches to Introductory Computer Science: The Key to Making Computer Science Relevant or Simply Bait and Switch?" *SIGCSE '11 Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, 177–182.
- Keller, John M. 2010. *Motivational Design for Learning and Performance: The ARCS Model Approach*. 1st Edition. Springer US.
- Kelly, Rob Roy. 2001. "Constraint vs. Restraint: Graphic Design Education And The Computer." In *Everything Is a Work in Progress: The Collective Writings of Rob Roy Kelly on Graphic Design Education*, 1st ed., 147–157. DesignLab, Rochester Institute of Technology, School of Design.
- Killion, Joellen & Todnem, Guy R. 1991. "A Process for Personal Theory Building." *Educational Leadership* 48 (6): 14–16.
- Kim, Lam A. 2016. "The Myth and Reality of Studio-Based Learning in Communication Design Education: The Potential of Integrating into an e-Learning Environment." Swinburne University of Technology, Australia.
- King, Robin G. 1988. "Computer Graphics and Animation as Agents of Personal Evolution in the Arts." *Leonardo. Supplemental Issue 1*: 43.
- Knochel, Aaron D. & Patton, Ryan M. 2015. "If Art Education Then Critical Digital Making: Computational Thinking and Creative Code." *Studies in Art Education* 57 (1): 21–38.
- Knuth, Donald Ervin. 1979. *TEX and METAFONT: New Directions in Typesetting*. Digital Press.
- Kolb, David A. 1984. *Experiential Learning: Experience as the Source of Learning and Development*. 1st Edition. Prentice-Hall.
- Kolko, Jon. 2012a. "Code Is Material: Why Designers Must Learn to Code." Austin Center for Design. Accessed from <http://www.ac4d.com/2012/06/code-is-material-why-designers-must-learn-to-code/>.
- Kolko, Jon. 2012b. "Transformative Learning in the Design Studio." *Interactions* 19 (6): 82–83.
- Kroeger, Michael. 2008. *Paul Rand: Conversations with Students*. 1st Edition. Princeton Architectural Press.
- Lage, Maureen J., Platt, Glenn J. & Treglia, Michael. 2000. "Inverting the Classroom: A Gateway to Creating an Inclusive Learning Environment." *The Journal of Economic Education* 31 (1): 30–43.
- Lawson, Bryan. 1980. *How Designers Think*. 1st Edition. Architectural Press.
- Lawson, Bryan. 2004. *How Designers Think - The Design Process Demystified*. 4th Edition. Architectural Press.
- Lehni, Jürg. 2008. "Soft Monsters." *Perspecta* 40 (Monster): 22–27.

- Lehni, Jürg & Puckey, Jonathan. 2011. "Teaching in the Spaces between Code and Design." *Eye Magazine* 81: 90–91.
- Leitão, António & Santos, Luís. 2011. "Programming Languages for Generative Design: Visual or Textual?" In *In Respecting Fragile Places: 29th ECAADe Conference Proceedings*, edited by Tadeja Strojan Zupancic, Matevz Juvancic, Spela Verovsek, and Anja Jutraz, 549–557. University of Ljubljana.
- Levit, Briar. 2016. *Graphic Means*. Tugg.
- Levy, Steven. 2010. *Hackers*. 1st Edition. O'Reilly Media.
- Liberal Arts Computer Science Consortium. 2007. *A 2007 Model Curriculum for a Liberal Arts Degree in Computer Science. Journal on Educational Resources in Computing*. Vol. 7.
- Liston, Daniel & Zeichner, Kenneth. 2013. *Reflective Teaching*. Lawrence Elbaum Associates. Routledge.
- Logan, Cheri D. 2006. "Circles of Practice: Educational and Professional Graphic Design." *The Journal of Workplace Learning* 18 (6): 331–343.
- Lukkarinen, Alekski & Sorva, Juha. 2016. "Classifying the Tools of Contextualized Programming Education and Forms of Media Computation." In *Koli Calling 2016*, 51–60. ACM Press.
- Lye, Sze Yee & Koh, Joyce Hwee Ling. 2014. "Review on Teaching and Learning of Computational Thinking through Programming: What Is next for K-12?" *Computers in Human Behavior* 41 (December): 51–61.
- Madsen, Rune. 2015. "On Meta-Design and Algorithmic Design Systems." Accessed from <https://runemadsen.com/blog/on-meta-design-and-algorithmic-design-systems/>.
- Madsen, Rune. 2016. "Printing Code | Intro Lecture." Accessed from <http://printingcode.runemadsen.com/lecture-intro/>.
- Maeda, John. 1999. *Design by Numbers*. 1st Edition. MIT Press.
- Maeda, John. 2002. "Design Education in the Information Age." *Design Management Journal* 13 (3): 39–45.
- Maeda, John. 2004. *Creative Code*. Thames & Hudson.
- Maeda, John. 2009. "John Maeda: My Journey in Design, from Tofu to RISD." TED. Accessed from <https://www.youtube.com/watch?v=uNYMw9O2bu4>.
- Maeda, John. 2018. "Design in Tech Report 2018." Accessed from <https://designintechreport.wordpress.com/>.
- Majoros, Ádám, Iván, József & Matusik, Szilárd. 2012. *Moleman 2 - Demoscene - The Art of the Algorithms*. Moleman Film.
- Manovich, Lev. 2005. "Generation Flash." In *Total Interaction*, edited by G. M. Buurmann. Birkhäuser-Verlag.
- Manovich, Lev. 2013. *Software Takes Command. International Texts in Critical Media Aesthetics*. 1st Edition. Bloomsbury Academic.
- Martinez, Sylvia Libow & Stager, Gary S. 2019. *Invent to Learn: Making, Tinkering, and Engineering in the Classroom*. 2nd Edition. Constructing Modern Knowledge Press.
- Mateas, Michael. 2005. "Procedural Literacy: Educating the New Media Practitioner." Edited by Drew Davidson. *On the Horizon* 13 (2): 101–111.
- Matthíasdóttir, Ásrún & Geirsson, Hrafn J. 2011. "The Novice Problem in Computer Science." In *Proceedings of the 12th International Conference on Computer Systems and Technologies - CompSysTech '11*, 570–576. ACM Press.
- Maurer, Luna, Paulus, Edo, Puckey, Jonathan & Wouters, Roel. 2013. *Conditional Design Workbook*. Valiz.

- Maxwell, Bruce A. & Taylor, Stephanie R. 2017. "Comparing Outcomes Across Different Contexts in CS1." In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, 399–403.
- Mazur, Eric. 1997. *Peer Instruction: A User's Manual*. Prentice-Hall, Inc.
- McCarthy, B. 1990. "Using the 4MAT System to Bring Learning Styles to Schools." *Educational Leadership* 48 (2): 31–37.
- McCarthy, Lauren, Fry, Ben & Reas, Casey. 2015. *P5.js*. 1st Edition. Maker Media Inc.
- McCormack, Jon, Bown, Oliver, Dorin, Alan, McCabe, Jonathan, Monro, Gordon & Whitelaw, Mitchell. 2014. "Ten Questions Concerning Generative Computer Art." *Leonardo* 47 (2): 135–141.
- McCoy, K. 1998. "Education in an Adolescent Profession." In *The Education of a Graphic Designer*, edited by Steven Heller, 3–12. Allworth Press.
- McDowell, Charlie, Werner, Linda, Bullock, Heather E. & Fernald, Julian. 2006. "Pair Programming Improves Student Retention, Confidence, and Program Quality." *Communications of the ACM* 49 (8): 90–95.
- McKim, Courtney A. 2017. "The Value of Mixed Methods Research: A Mixed Methods Study." *Journal of Mixed Methods Research* 11 (2): 202–222.
- McLean, Alex & Wiggins, Geraint. 2008. "Bricolage Programming in the Creative Arts." In *22nd Psychology of Programming Interest Group*.
- McNamara, Patrick. 2015. "The Influence of MOOCs to Enhance Graphic Design Education." *Art, Design & Communication in Higher Education* 14 (1): 57–69.
- Meggs, Philip B. & Purvis, Alston W. 2006. *Meggs' History of Graphic Design*. 4th Edition. Wiley.
- Melles, Gavin. 2008. "An Enlarged Pragmatist Inquiry Paradigm for Methodological Pluralism in Academic Design Research" II (1): 3–13.
- Mertens, Donna M. 2007. "Transformative Paradigm: Mixed Methods and Social Justice." *Journal of Mixed Methods Research* 1 (3): 212–225.
- Meyer, Jan H. F. & Land, Ray. 2003. "Threshold Concepts and Troublesome Knowledge: Linkages to Ways of Thinking and Practising within the Disciplines." In *Improving Student Learning – Ten Years On*, edited by C. Rust, 1–16. Oxford Centre for Staff and Learning Development (OCSLD).
- Mezirow, Jack. 1991. *Transformative Dimensions of Adult Learning. The JosseyBass Higher and Adult Education Series*. Wiley.
- Mezirow, Jack. 1997. "Transformative Learning: Theory to Practice." *New Directions for Adult and Continuing Education* 197 (74): 5–12.
- Mishra, Punya & Yadav, Aman. 2013. "Of Art and Algorithm: Rethinking Technology & Creativity in the 21st Century." *TechTrends* 57 (3): 10–14.
- Mitchell, Mark C. & Bown, Oliver. 2013. "Towards a Creativity Support Tool in Processing: Understanding the Needs of Creative Coders." In *Proceedings of the 25th Australian Computer-Human Interaction Conference on Augmentation, Application, Innovation, Collaboration - OzCHI '13*, 143–146. ACM Press.
- Mittendorf, Jan. 2000. "Toolspace." Accessed from <http://lettererror.com/writing/toolspace/>.
- Monnier, Antoinette. 1995. "The Interrelationship of Graphic Design and Fine Art." Rochester Institute of Technology.
- Montfort, Nick. 2016. *Exploratory Programming for the Arts and Humanities*. The MIT Press.
- Moszkowicz, Julia. 2013. "American Pragmatism and Graphic Design: Retrieving the Historical and Philosophical Constitutions of a 'Non-Theoretical' Approach." *The Design Journal* 16 (3): 315–338.
- Müller-Brockmann, Josef. 1981. *Raster Systeme Für Die Visuelle Gestaltung*. 1st Edition. Niggli Verlag.

- Nagappan, Nachiappan, Williams, Laurie, Ferzli, Miriam, Wiebe, Eric, Yang, Kai, Miller, Carol & Balik, Suzanne. 2003. "Improving the CS1 Experience with Pair Programming." In *SIGCSE '03 Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, 359–362. ACM Press.
- Nardi, Bonnie A. 1993. *A Small Matter of Programming: Perspectives on End User Computing*. MIT Press.
- National Research Council. 2000. *How People Learn: Brain, Mind, Experience, and School: Expanded Edition*. National Academies Press.
- Nelson, Wayne A. 2013. "Design, Research, and Design Research: Synergies and Contradictions." *Educational Technology* 53 (1): 3–11.
- Nichols, Mark. 2016. "Reading and Studying on the Screen: An Overview of Literature Towards Good Learning Design Practice." *Journal of Open Flexible and Distance Learning* 20 (1): 33–43.
- Nolan, Keith & Bergin, Susan. 2016. "The Role of Anxiety When Learning to Program." In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research - Koli Calling '16*, 61–70.
- Nuutila, Esko, Törmä, Seppo & Malmi, Lauri. 2005. "PBL and Computer Programming — The Seven Steps Method with Adaptations." *Computer Science Education* 15 (2): 123–142.
- Palmer, Cathy. 2011. "Teaching Code to Creatives: Removing Learning Barriers to Incorporating Automation into Graphic Design." In *E-Learn: World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education*, edited by C. Ho and M. Lin, 2066–2072. Association for the Advancement of Computing in Education (AACE).
- Panda, Payod. 2016. "Helping Designers Understand Code." Department of Graphic and Industrial Design, North Carolina State University.
- Pannafino, James. 2013. "AIGA Design Educators Community | Learn That Over There ... Do Design Students Need to Learn to Code within a Design Curriculum?" AIGA Design Educators Community. Accessed from <https://educators.aiga.org/learn-that-over-there-do-design-students-need-to-learn-to-code-within-a-design-curriculum/>.
- Papert, Seymour. 1971. "Teaching Children to Be Mathematicians vs. Teaching About Mathematics." *International Journal of Mathematical Education in Science and Technology* 3 (3): 249–262.
- Papert, Seymour. 1980. *Mindstorms: Children, Computers, and Powerful Ideas*. 1st Edition. Basic Books Inc.
- Papert, Seymour. 1993. *Mindstorms: Children, Computers, And Powerful Ideas*. 2nd Edition. Basic Books Inc.
- Papert, Seymour & Harel, Idit. 1991. "Situating Constructionism." In *Constructionism*, edited by Seymour Papert and Idit Harel, 1st ed., 1–11. Ablex Publishing Corporation.
- Papert, Seymour & Turkle, Sherry. 1990. "Epistemological Pluralism: Styles and Voices within the Computer Culture." *Signs* 16 (1): 128–157.
- Pears, Arnold, Seidman, Stephen, Malmi, Lauri, Mannila, Linda, Adams, Elizabeth, Bennedsen, Jens, Devlin, Marie & Paterson, James. 2007. "A Survey of Literature on the Teaching of Introductory Programming." *ACM SIGCSE Bulletin* 39 (4): 204–223.
- Pearson, Matt. 2011. *Generative Art - A Practical Guide Using Processing*. Manning Publications Co.
- Pearson, Matt. 2013. "Everyone Hates Creative Coders." In *OFFF 2013*, 72–75. OFFF.
- Pepler, Kylie A. & Kafai, Yasmin B. 2005. "Creative Coding: Programming for Personal Expression." Accessed from <https://download.scratch.mit.edu/CreativeCoding.pdf>.
- Pettitway, Keon. 2012. "The New Media Programme: Computational Thinking in Graphic Design Practice and Pedagogy." *Journal of the New Media Caucus*.
- Philadelphia Education Research Consortium. 2014. "Blended Learning - Defining Models and Examining Conditions to Support Implementation."

- Post, Leda Van Der. 2010. "A Computing Studio Method for Teaching Design Thinking." Nelson Mandela Metropolitan University.
- Poulin, Richard. 2011. *The Language of Graphic Design: An Illustrated Handbook for Understanding Fundamental Design Principles*. Rockport Publishers.
- Prediger, Susanne, Gravemeijer, Koeno & Confrey, Jere. 2015. "Design Research with a Focus on Learning Processes: An Overview on Achievements and Challenges." *ZDM - Mathematics Education* 47 (6): 877–891.
- Radošević, Danijel, Orehovački, Tihomir & Lovrenčić, Alen. 2009. "New Approaches and Tools in Teaching Programming." *Ceciis '09*, no. September: 49–57.
- Reed, David & Davies, Joel. 2006. "The Convergence of Computer Programming and Graphic Design." *Journal of Computing Sciences in Colleges* 21 (3): 179–187.
- Reichardt, Jasja & Institute of Contemporary Arts. 1969. *Cybernetic Serendipity: The Computer and the Arts*. 1st Edition. Praeger.
- Reimer, Yolanda J. & Douglas, Sarah A. 2003. "Teaching HCI Design With the Studio Approach." *Computer Science Education* 13 (3): 191–205.
- Reiser, Robert A. & Dempsey, John V. 2007. *Trends and Issues in Instructional Design and Technology*. 2nd ed. Prentice-Hall, Inc.
- Repenning, Alexander. 2017. "Moving Beyond Syntax: Lessons from 20 Years of Blocks Programming in AgentSheets." *Journal of Visual Languages and Sentient Systems* 3.
- Resnick, Mitchel, Myers, Brad A., Nakakoji, Kumiyo, Shneiderman, Ben, Pausch, Randy, Selker, Ted & Eisenberg, Mike. 2005. "Design Principles for Tools to Support Creative Thinking." In *Proceedings of the Workshop on Creativity Support Tools*.
- Richardson, Andrew Grant. 2006. "New Media, New Craft?" In *SIGGRAPH Boston 2006: Electronic Art and Animation Calalogue*, 157–159.
- Richardson, Andrew Grant. 2010. "Truth to Material: Moving from Software to Programming Code as a New Material for Digital Design Practice." University of Sunderland.
- Richardson, Andrew Grant. 2016. *Data-Driven Graphic Design: Creative Coding for Visual Communication*. Bloomsbury Academic.
- Rittel, Horst W. J. & Webber, Melvin M. 1973. "Dilemmas in a General Theory of Planning." *Policy Sciences* 4 (2): 155–169.
- Rockinson-Szapkiw, Amanda J., Courduff, Jennifer, Carter, Kimberly & Bennett, David. 2013. "Electronic versus Traditional Print Textbooks: A Comparison Study on the Influence of University Students' Learning." *Computers & Education* 63 (April): 259–266.
- Rogerson, Christine & Scott, Elsje. 2010. "The Fear Factor: How It Affects Students Learning to Program in a Tertiary Environment." *Journal of Information Technology Education: Research* 9 (1): 147–171.
- Ross, Rockford J. & Grinder, Michael T. 2002. "Hypertextbooks: Animated, Active Learning, Comprehensive Teaching and Learning Resources for the Web." In *Software Visualization, Lecture Notes in Computer Science, Vol 2269*, edited by S. Diehl, 269–283. Springer.
- Rossum, Just Van & Blokland, Erik Van. 1990. "FontShop | FF Beowolf." Accessed from <https://www.fontshop.com/families/ff-beowolf>.
- Roth, Susan. 1999. "The State of Design Research." *Design Issues* 15 (2): 18–26.
- Rubin, Marc J. 2013. "The Effectiveness of Live-Coding to Teach Introductory Programming." In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education - SIGCSE '13*, 651–656.
- Rushkoff, Douglas. 2010. *Program or Be Programmed: Ten Commands for a Digital Age*. 1st Edition. OR Books.

- Ryan, Richard M. & Deci, Edward L. 2000. "Intrinsic and Extrinsic Motivations: Classic Definitions and New Directions." *Contemporary Educational Psychology* 25 (1): 54–67.
- Schneider, Michael G. 1978. "The Introductory Programming Course in Computer Science – Ten Principles." In *Papers of the 9th SIGCSE/CSA Technical Symposium on Computer Science Education*, 107–114. ACM Press.
- Schön, Donald A. 1983. *The Reflective Practitioner: How Professionals Think in Action*. 1st Edition. Basic Books.
- Schön, Donald A. 1987. *Educating the Reflective Practitioner: Toward a New Design for Teaching and Learning in the Professions*. 1st Edition. Wiley.
- Scott, Terry & Ursyn, Anna. 2006. "A Web Design Course Team Taught by Professors in Art and Computer Science." *Journal of Computing Sciences in Colleges* 22 (1): 205–210.
- Shaffer, David Williamson. 2007. "Learning in Design." In *Foundations for the Future in Mathematics Education*, edited by R.A. Lesh, J.J. Kaput, and E. Hamilton, 99–126. Lawrence Erlbaum Associates.
- Shannon-Baker, Peggy. 2016. "Making Paradigms Meaningful in Mixed Methods Research." *Journal of Mixed Methods Research* 10 (4): 319–334.
- Shiffman, Daniel. 2012. *The Nature of Code*. Self-published.
- Shiffman, Daniel. 2018. *Learning to Teach 2018 – Daniel Shiffman*. School for Poetic Computation.
- Shim, Kyuha. 2016a. "Computation for Graphic Designers." Medium. Accessed from <https://medium.com/@qshim/computation-for-graphic-designers-23629ec63dc0>.
- Shim, Kyuha, ed. 2016b. *GRAPHIC #37: Introduction To Computation*. GRAPHIC. Propaganda Press.
- Simon, Beth, Kohanfars, Michael, Lee, Jeff, Tamayo, Karen & Cutts, Quintin. 2010. "Experience Report: Peer Instruction in Introductory Computing." In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education – SIGCSE '10*, 341–345. ACM Press.
- Smith, Annabel, Young, Robert A. & Raeside-Elliot, Fiona. 2015. "Teaching Business Concepts Using Visual Narrative." In *Proceeding of the 3rd International Conference for Design Education Researchers*, 1552–1568.
- Sorva, Juha, Karavirta, Ville & Malmi, Lauri. 2013. "A Review of Generic Program Visualization Systems for Introductory Programming Education." *ACM Transactions on Computing Education* 13 (4): 1–64.
- Stager, Gary S. 2001. "Constructionism as a High-Tech Intervention Strategy for At-Risk Learners." In *National Educational Computing Conference, "Building of the Future,"* 1–11.
- Sterling, Bruce. 2012a. "An Essay on the New Aesthetic." WIRED. Accessed from <https://www.wired.com/2012/04/an-essay-on-the-new-aesthetic/>.
- Sterling, Bruce. 2012b. "Generation Generator (New Aesthetic)." WIRED. Accessed from <https://www.wired.com/2012/04/generation-generator-new-aesthetic/>.
- Stevenson, D. E. 2001. "Problem-Based Learning Applied to Programming Instruction." In *Submitted to SIGCSE 2001*.
- Stinson, Elizabeth. 2017. "John Maeda: If You Want To Survive In Design, You Better Learn To Code." WIRED. Accessed from <https://www.wired.com/2017/03/john-maeda-want-survive-design-better-learn-code>.
- Stiny, George. 2001. "When Is Reasoning Visual?"
- Stolterman, Erik. 2008. "The Nature of Design Practice and Implications for Interaction Design Research." *International Journal of Design* 2 (1): 55–65.
- Stoop, Judith, Kreutzer, Paulien & Kircz, Joost G. 2013. "Reading and Learning from Screens versus Print: A Study in Changing Habits (Part 1)." *New Library World* 114 (9/10): 371–383.

- Sutton, Ken & Williams, Anthony. 2010. "Implications of Spatial Abilities on Design Thinking." In *Design Research Society International Conference*.
- Takemura, Yasuhiro, Nagumo, Hideo, Huang, Kuo Li & Tsukamoto, Hidekuni. 2008. "Assessing the Learners' Motivation in the E-Learning Environments for Programming Education." In *Advances in Web Based Learning - ICWL 2007*, edited by H. Leung, F. Li, and Q. Li, 4823 LNCS:355–366. Springer.
- Tashakkori, Abbas & Teddlie, Charles. 2010. *Sage Handbook of Mixed Methods in Social & Behavioral Research*. SAGE Publications.
- Teddlie, Charles & Tashakkori, Abbas. 2009. *Foundations of Mixed Methods Research. Foundations of Mixed Methods Research*. SAGE Publications.
- Tedre, Matti, Malmi, Simon & Malmi, Lauri. 2018. "Changing Aims of Computing Education: A Historical Survey." *Computer Science Education* 28 (2): 158–186.
- Terzidis, Kostas. 2009. *Algorithms for Visual Design Using the Processing Language*. 1st Edition. Wiley.
- Tew, Allison Elliott & Guzdial, Mark. 2010. "Developing a Validated Assessment of Fundamental CS1 Concepts." In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education - SIGCSE '10*, 97–101.
- Tober, Brad. 2011. "New Tools of the Trade: An Exploration of Interactive Computational Graphic Design Processes." York University, Toronto, Canada.
- Tober, Brad. 2012a. "Creating with Code: Critical Thinking and Digital Foundations." In *Mid-America College Art Association Conference 2012 Digital Publications*, Paper 16.
- Tober, Brad. 2012b. "Making the Case for Code: Integrating Code-Based Technologies into Undergraduate Design Curricula." In *Catch22: Eighth Annual UCDA Design Education Summit Abstracts & Proceedings*, 224–229. University & College Designers Association, Smyrna, TN, USA.
- Tober, Brad. 2013. "Text: Keeping the Balance: Copyright, Plagiarism, and Creative Code in the Classroom." *Art + Copyright*. Accessed from <http://artcopyright.interartive.org/copyright-plagiarism-creative-code/>.
- Tober, Brad. 2014. "Creative Code in the Design Classroom - Preparing Students for Contemporary Professional Practice." In *An Illinois Sampler: Teaching and Research on the Prairie*, edited by Mary-Ann Winkelmes, Antoinette Burton, and Kyle Mays, 1st ed. University of Illinois Press.
- Tober, Brad. 2017. "Teaching for the Design Singularity: Toward an Entirely Code-Based Design Curriculum." In *Teaching Computational Creativity*, edited by Michael Filimowicz and Veronika Tzankova, 1st ed. Cambridge University Press.
- Tomayko, James E. 1991. "Teaching Software Development in a Studio Environment." In *Proceedings of the 22nd SIGCSE Technical Symposium on Computer Science Education - SIGCSE '91*, 23:300–303.
- Tucker, H. A. 1988. "Desktop Publishing." In *Advances in Computer Graphics III*, edited by Maurice M. de Ruiter, 293–322. Springer-Verlag.
- Tzankova, Veronika & Filimowicz, Michael. 2017. "Introduction: Pedagogies at the Intersection of Disciplines." In *Teaching Computational Creativity*, edited by Michael Filimowicz and Veronika Tzankova, 1st ed., 1–17. Cambridge University Press.
- Ulrich, Karl T. 2006. "Aesthetic in Design." In *Design: Creation of Artifacts in Society*, 1st ed. Pontifica Press.
- Ursyn, Anna, Scott, Terry, Hobgood, Benjamin R. & Mill, Lizette. 1997. "Combining Art Skills with Programming In Teaching Computer Art Graphics." *Computer Graphics August*: 60–61.
- Vantomme, Jan. 2012. *Processing 2 : Creative Programming Cookbook*. Packt Publishing Ltd.
- Victor, Bret. 2012. "Learnable Programming: Designing a Programming System for Understanding Programs." Accessed from <http://worrydream.com/LearnableProgramming/>.

- Vihavainen, Arto, Paksula, Matti & Luukkainen, Matti. 2011. "Extreme Apprenticeship Method in Teaching Programming for Beginners." In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education - SIGCSE '11*, 93–98.
- Walsh, Gemma. 2016. "Screen and Paper Reading Research – A Literature Review." *Australian Academic and Research Libraries* 47 (3): 160–173.
- Warburton, Chantelle. 2017. "An Evaluation of Blended Learning for Critical Reflection in Graphic Design Higher Education." Durban University of Technology.
- Ward, Adrian. 2001. "Life & Oblivion." In *Generative Design: Beyond Photoshop*, 66–81. Friends of ED Ltd.
- Wasco, Al. 2008. "Teaching Design, Teaching Technology: Time to Rethink Our Approach." In *Massaging Media 2 Conference*.
- Watz, Marius. 2003. "Teaching - Computational Design and Generative Art." Accessed from <http://workshop.evolutionzone.com/old/>.
- Watz, Marius. 2010. "Closed Systems: Generative Art and Software Abstraction." In *MetaDesign*, edited by Lab[au]. Les Presses du Réel.
- Watz, Marius, Chayka, Kyle, Minard, Jonathan, Borenstein, Greg, George, James & McDonald, Kyle. 2012. "In Response To Bruce Sterling's 'Essay On The New Aesthetic.'" The Creators Project. Accessed from <http://www.thecreatorsproject.com/blog/in-response-to-bruce-sterlings-essay-on-the-new-aesthetic>.
- Webb, Noreen M. 1985. "Cognitive Requirements of Learning Computer Programming in Group and Individual Settings." *AEDS Journal* 18 (3): 183–194.
- Weinman, Lynda. 2001. "Education of a Digital Designer." In *The Education of an E-Designer*, edited by Steven Heller, 60–62. Allworth Press.
- Wick, Rainer K. 2000. "Teaching at the Bauhaus." Hatje Cantz Publishers. Accessed from https://www.bauhaus.de/en/das_bauhaus/45_unterricht/.
- Williams, Laurie, Kessler, R.R., Cunningham, Ward & Jeffries, Ron. 2000. "Strengthening the Case for Pair Programming." *IEEE Software* 17 (4): 19–25.
- Willingham, Daniel T., Hughes, Elizabeth M. & Dobolyi, David G. 2015. "The Scientific Status of Learning Styles Theories." *Teaching of Psychology* 42 (3): 266–271.
- Willis, Holly. 2007. "Toward an Algorithmic Pedagogy." *Fibreculture Journal*, no. 10.
- Wing, Jeannette M. 2006. "Computational Thinking." *Communications of the ACM* 49 (3): 33–35.
- Wing, Jeannette M. 2008. "Computational Thinking and Thinking about Computing." *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 366 (October): 3717–3725.
- Wing, Jeannette M. 2014. "Computational Thinking Benefits Society." Social Issues in Computing: 40th Anniversary Blog. Accessed from <http://socialissues.cs.toronto.edu/index.html%3Fp=279.html>.
- Wing, Jeannette M., Cuny, Jan & Snyder, Larry. 2010. "Demystifying Computational Thinking for Non-Computer Scientists." Unpublished Manuscript.
- Winograd, Terry. 1990. "What Can We Teach About Human-Computer Interaction?" In *CHI '90 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 443–449. ACM.
- Womack, David & Lehni, Jürg. 2006. "Tools to Make or Break." *Eye Magazine* 60.
- Xu, Dianna, Blank, Doug & Kumar, Deepak. 2008. "Games, Robots, and Robot Games: Complementary Contexts for Introductory Computing Education." In *GDCSE '08 Proceedings of the 3rd International Conference on Game Development in Computer Science Education*, 66–70. ACM.

- Xu, Dianna, Greenberg, Ira, Kumar, Deepak & Wolz, Ursula. 2016. "Creative Computation for CS1 and K9-12." In *Envisioning the Future of Undergraduate STEM Education: Research and Practice Symposium (AAAS EnFuse)*, 1–6.
- Xu, Dianna, Wolz, Ursula & Greenberg, Ira. 2018. "Updating Introductory Computer Science with Creative Computation." In *Proceedings of SIGCSE '18*, 167–172. ACM Press.
- Young, David. 2001. "Why Designers Need To Learn Programming." In *Education of an E-Designer*, edited by Steven Heller, 64–67. Allworth Press.
- Zarestky, Jill & Bangerth, Wolfgang. 2014. "Teaching High Performance Computing: Lessons from a Flipped Classroom, Project-Based Course on Finite Element Methods." In *2014 Workshop on Education for High Performance Computing*, 34–41. IEEE.
- Zee, Natalie. 2001. "The Design Technologist: Creating Interactive Experiences." In *The Education of an E-Designer*, edited by Steven Heller, 72–73. Allworth Press.
- Zimmerman, John, Forlizzi, Jodi & Evenson, Shelley. 2007. "Research through Design as a Method for Interaction Design Research in HCI." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '07*, 493–502. ACM Press.