CASTING THE RUNES AND PARSING THEM

UNPACKING SOFTWARE MEDIATION, INTERACTIONS, AND COMPUTATIONAL LITERACY IN NON-CONVENTIONAL PROGRAMMING CONFIGURATIONS

PHD DISSERTATION F2023
BJARKE VOGNSTRUP FOG
AARHUS UNIVERSITY
SCHOOL OF COMMUNICATION AND CULTURE

AARHUS UNIVERSITY

Århus University

Department of Digital Design and Information Studies

Doctoral dissertation

Casting the runes and parsing them: Unpacking software mediation, interactions, and computational literacy in non-conventional programming configurations

Bjarke Vognstrup Fog

Supervisor
Clemens Nylandsted Klokmose

Co-supervisor
Ole Sejer Iversen

PhD programme director
Iben Have

June 2023

DOI: 10.7146/aul.501
ISBN: 978-87-7507-548-5

Acknowledgements

This dissertation only has me on the author list. Yet it would not exist had it not been for so many people around me. First and foremost, I wish to thank Dr. Kate Howland and Prof. Dr. Andreas Mühling for taking the time to read and assess the present dissertation as part of the assessment committee. The committee would also not be complete without Minna Pakanen who, besides taking the time to head the committee, has been a source of great talks and office candy over the years. A special thank goes to Kasper Ostrowski for making the front page and to Jonas and Frida for giving me feedback on various parts of the dissertation.

I also need to thank my supervisors, Clemens Nylandsted Klokmose and Ole Sejer Iversen, for all the supervision, support, and guidance they have provided over the years, even after their official obligation to supervise me ended. I have not always been the easiest PhD student to supervise, yet you never gave up on me. Clemens, thank you for being a great supervisor, co-author, co-teacher, travel buddy, concert goer, and friend. Your approach to research has been inspiring. Ole, thank you for the great chats, for challenging me at every step, and for forcing me to take on some of your "method fundamentalism" which has helped ground me when my head was in the clouds.

Since the beginning of my PhD in 2018, I have almost religiously been attending every Friday meeting at CCTD. Not only because of the academic content and the morning rolls, but because of the people who always made me feel included as a valued colleague. Despite COVID, I managed to get to know and visit Gregor, Andreas, and all the other wonderful people from the *Arbeitsgruppe Didaktik der Informatik* at Christian-Albrechts-Universität in Kiel. Thank you so much for letting me visit and for providing nice academic and personal experiences. I still think fondly of the trip to Haithabu.

I have been lucky to be surrounded with many great colleagues. Among my fellow PhD students at IMCJ, I found friends and partners-in-suffering. The Christmas party in particular was one of the highlights of last year. A lot of other people have made my life at the office pleasant and enjoyable. I cannot possibly mention all of you here, but I especially enjoyed the company of Liam and Ignacio (and their partners) at our many bouldering outings. Of course, Line Have and Malle need to be recognized for their ways of being and for listening to my many woes over the years. Morten and Iben likewise deserve a thought of gratitude for helping me through the hard times. At CHC, I have met other people whose company is both joyful and provides me with the opportunity for growth. Thank you all for being you. Especially Peter has been vital for my interest in programming since I first worked with you as a teaching assistant eight years ago.

Of course, none of the articles in this dissertation would be possible without my various co-authors over the years: Clemens, Midas, Marcel, Carla, James, ML, Line Have, Marianne, Blanka, and Alberte. It has been a joy to work with all of you. All the participants in our research deserve thanks, too. But not only have I been a researcher during the last five years, I have also been a teacher. Thank you to all of my students for providing me with good experiences, for challenging me, and for showing me that what I say actually matters.

Despite this extensive list, I have probably forgotten some of those that have contributed to my personal and academic development during the last five years. Even so, I have appreciated all the people that I have met over the years. All of you who engage in chats at the coffee machine, the secretaries, the cleaning staff, the casual acquaintances at various gatherings. And of course, my numerous office mates (9!) with whom I have had the pleasure of sharing a space.

On a more private note, my sister Kristine deserves a shoutout. Thank you for being there when everything else fell apart. The same goes for Lisbeth, Martin, Thomas, Kasper, Johnny, Anne, Carina, and the rest of the Gang. I know that many of you have been on the receiving end of my complaints, qualms, and frustrations. And finally, what you are currently reading would have never existed without the support of my dear Lotte. You make me the best version of myself. Thank you for being by my side through all the long days and nights of work.

Bjarke Vognstrup Fog
Århus
June 2023

List of publications

Paper 1	**Bjarke Vognstrup Fog** & Clemens Nylandsted Klokmose (2019). Mapping the landscape of literate computing. In *Proceedings of the 30th Annual Workshop of the Psychology of Programming Interest Group, ser. PPIG.*
Paper 2	Midas Nouwens, Marcel Borowski, **Bjarke Vognstrup Fog** & Clemens Nylandsted Klokmose (2020). Between Scripts and Applications: Computational Media for the Frontier of Nanoscience. In *CHI '20: Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems.*
Paper 3	Marcel Borowski, **Bjarke Vognstrup Fog**, Carla F. Griggio, James Eagan & Clemens Nylandsted Klokmose (2022). Between Principle and Pragmatism: Reflections on Prototyping Computational Media with Webstrates. In *TOCHI: ACM Transactions on Computer-Human Interaction.*
Paper 4	Marie-Louise Stisen Kjerstein Sørensen, **Bjarke Vognstrup Fog**, Line Have Musaeus & Marianne Graves Petersen (2022). KnitxCode: Exploring a Craftsmanship-driven Approach to Computational Thinking. In *NordiCHI '22: Adjunct Proceedings of the 2022 Nordic Human-Computer Interaction Conference.*
Paper 5	**Bjarke Vognstrup Fog**, Blanka Pálfi, Alberte Uhre Mortensen & Line Have Musaeus (2023, submitted). Computational self-concept: Towards an understanding of students' identities, attitudes, and beliefs. Submitted to *Education and Information Technologies.*

Table 1: Published and submitted research articles

Other research output

Eva Eriksson, Ole Sejer Iversen, Gökçe Elif Baykal, Maarten Van Mechelen, Rachel Charlotte Smith, Marie-Louise Wagner, **Bjarke Vognstrup Fog**, Clemens Nylandsted Klokmose, Bronwyn Cumbo, Hermes Arthur Hjorth, Line Have Musaeus, Marianne Graves Petersen & Niels Olof Bouvin (2019). Widening the Scope of FabLearn Research: Integrating Computational Thinking, Design and Making. In *FabLearn Europe '19: Proceedings of the FabLearn Europe 2019 Conference.*

Bjarke Vognstrup Fog (2021, August 16–19). *Computational self-concept — and the issue of having people identify as computationally competent* [Lightning talk]. ICER 2021, virtual.

Marie-Louise Stisen Kjerstein Sørensen, **Bjarke Vognstrup Fog**, Line Have Museaus & Marianne Graves Petersen (2022). Knit x Code - Exploring a Craftsmanship-driven Approach to Computational Thinking. *Dutch Design Week, TU Delft.* https://www.4tu.nl/du/columns/Knit%20x%20Code/.

Table 2: Other research dissemination

Abstract

This dissertation is an investigation of computational literacy and how it is shaped by software use and mediation. Early visionaries such as Perlis and Naur recognized the need for everyone to learn computing, but these ideals are yet to be fully realized. Arguably, a narrow focus on computational thinking is the more popular approach in contemporary computing education research and policymaking. Another branch of researchers, in particular Kay and diSessa, have argued for the need for providing the right media for computing. In line with them, I argue that a more materially grounded literacy is a necessary step. By extension, this means providing a better understanding of how these material conditions (e.g., software) influence the development of computational literacy.

Through eight studies, I have employed a mix of qualitative methods and constructive design research. The qualitative methods fall under ethnography, technography, and retrospective autoethnography. The empirically grounded research draws from interviews with five humanities students, interviews and observations of four biomolecular scientists, interviews with 12 experienced programmers, and a workshop and observations of 12 experienced knitters. These interviews focused on their experiences with programming, their ability to use and appropriate unfamiliar software, and their feelings of mastery and disempowerment. This is supplemented with technographic investigations of computational media, literate computing environments, and programming interfaces that focuses on the mediating qualities of software for programming such as interaction, semiotics, ethics, and transformation.

My work has shown the importance of the material foundations of computational literacy in these contexts. More specifically, the material conditions affect this literacy in multiple ways such as the dissonance between software visions, people's expectations, and the practical implementations. People experience disempowerment and crises and resolve those through various means such as enrolling a more capable peer or incorporating supporting artifacts. The dissertation further presents computational media as a promising, yet fragile software paradigm and shows how this paradigm blends use and development, inscribes particular user roles, and balances between evoking trust and alienation in its users. Finally, by emphasizing a theoretical lens of self-concept in the context of computational literacy, the dissertation provides a view of literacy as a product of continuous experiences and confirmations from people's social and material lifeworlds.

These findings should resonate with scholars of new media, human-computer interaction, and computing education, as the dissertation explores the complex mutual relationships between people's cultural, social, and material environments as well as their ongoing and sometimes contradictory ways of seeing themselves. Computational literacy can be emancipatory for everyone, not just for computer scientists, yet the development of literacy demands adequate conditions. This dissertation is an argument for the importance of those conditions.

Dansk resumé

At riste og råde runerne: En udlægning af mediering, interaktion og datalogiske kompetencer i ukonventionelle programmeringskonfigurationer

Denne afhandling er skrevet på engelsk. Jeg benytter engelske begreber, som desværre er vanskelige at oversætte uden at miste noget af deres betydning. Eksempelvis er der på engelsk en forskel mellem *literacy* og *kompetencer*, som ikke findes på samme måde på dansk. *Computational* har jeg nogle steder i det følgende oversat til *datalogisk*, selvom der er en mindre begrebsmæssig forskel. Ligeledes har jeg oversat *computing* til *datalære*, hvor det er hensigtsmæssigt.

Afhandlingen er en undersøgelse af datalogiske kompetencer (en. *computational literacy*), og hvordan de formes af softwarebrug og -mediering. Tidlige visionærer som Perlis og Naur indså behovet for, at alle skulle lære datalære, men disse idealer er endnu ikke opfyldt. Et snævert fokus på datalogisk tænkning (en. *computational thinking*) er tilsyneladenede en mere populær tilgang i aktuel uddannelsesforskning og på den politiske dagsorden. En forskningsgren, anført af blandt andre Kay og diSessa, har argumenteret for behovet for de rigtige medier i datalære (en. *computing*). I tråd med disse argumenter italesætter jeg også det nødvendige i mere materielt funderede kompetencer. Det betyder altså i denne sammenhæng, at der er behov for en bedre forståelse af, hvordan de materielle betingelser (fx software) påvirker udviklingen af datalogiske kompetencer.

Gennem otte studier har jeg anvendt en kombination af kvalitative metoder og konstruktiv designforskning. De kvalitative metoder er mere specifikt etnografi, teknografi og retrospektiv autoetnografi. Min empirisk funderede forskning trækker på interviews med fem humanistiske studerende, interviews og observationer med fire biomolekylære forskere, interviews med tolv erfarne programmører og en workshop og observationer med tolv erfarne strikkere. Fokus i interviews og observationer var deltagernes erfaringer med programmering, deres evne til at bruge og tilegne sig ukendt software samt deres følelse af mestring og umyndiggørelse. Dette blev suppleret med teknografiske undersøgelser af *computational* medier, *literate computing*-miljøer og programmeringsinterfaces med fokus på de medierende kvaliteter i programmeringssoftware, eksempelvis interaktion, semiotik, etik og transformation.

Mit arbejde har demonstreret betydningen af det materielle fundament for datalogiske kompetencer i disse kontekster. Mere specifikt påvirker de materielle betingelser kompetencerne på flere måder, eksempelvis i form af dissonans mellem softwarevisioner, forventninger og praktiske implementeringer. Deltagerne oplever umyndiggørelse og kriser, som de løser på forskellige måder, blandt andet ved indrullering af mere kompetente ligesindede eller inkorporering af andre artefakter. Afhandlingen præsenterer derudover *computational* medier som et lovende, men skrøbeligt softwareparadigme og viser, hvordan paradigmet blander brug og udvikling, indskriver bestemte brugerroller og balancerer mellem tillid og fremmedgørelse for de mennesker, som har med det at gøre. Slutteligt bidrager afhandlingen med et perspektiv på kompetencer, der bygger på selvbegreb som et teoretisk indgangsvinkel. I dette perspektiv er kompetence et produkt af folks løbende erfaringer og bekræftelser fra deres sociale og materielle livsverdener.

Mine resultater skulle gerne give genlyd blandt forskere med interesse i nye medier, menneske-maskin-interaktion og informatikundervisning. Dette skyldes især, at afhandlingen udforsker de komplekse og gensidige forhold mellem folks kulturelle, sociale og materielle omverden samt de løbende og ofte selvmodsigende måder, de ser sig selv. Datalogiske kompetencer kan være frigørende og dannende for alle, ikke bare for dataloger og programmører, men udviklingen af disse kompetencer forudsætter altså tilstrækkelige betingelser. Denne afhandling er et argument for vigtigheden af disse betingelser.

Contents

Part I

Overview article

Introduction 1

Before starting the dissertation proper, I must explain the title. In the early days of computing history, more specifically in the AI labs of MIT and Stanford among other computing research centers in the 50s, a set of insider slang developed among those "in the know". This list grew and was codified in the 70s as what became known as the Jargon File. The Jargon File has thus become a kind of cultural lexicon of a now largely dissolved subculture. Its glossary includes, among hundreds of other entries, the following definitions:

Runes "Anything that requires *heavy wizardry* or *black art* to *parse*: core dumps, JCL commands, APL, or code in a language you haven't a clue how to read"[1]

Casting the runes "What a *guru* does when you ask him or her to run a particular program and type at it because it never works for anyone else; esp. used when nobody can ever see what the guru is doing different from what J. Random Luser does"[2]

Parse "More generally, to understand or comprehend. 'It's very simple; you just kretch the glims and then aos the zotz.' 'I can't parse that'."[3]

These entries illustrate the close connections between reading, writing, understanding, and expertise in the computational realm. Besides the references to other insider slang such as *guru* and the imagined persona called *J. Random Luser*, the choice of a runic metaphor is interesting; in prehistoric Scandinavia, only a select few were able to read, let alone write, runic letters in this largely oral culture, sometimes even imbuing them with seemingly magical properties ("Sayings of the High One" 2014). In a similar vein, for many people today, computers are essentially black boxes, only accessible to the few people who possess the "magical powers" to read and write their esoteric commands.

> *I tried to get it to work on my home computer and ran into some difficulties and decided to just give up and that it's better not to work from home.* (A nanomolecular researcher working at the cutting-edge of computational RNA folding).

One consequence of the spread and development of digital technologies is that they are no longer only for the initiated. Leisure, governance, work, education, and basically all aspects of civic life are mediated through digital technologies, tending towards mandatory rather than discretionary use (Grudin 2017, pp. 42–43). This demands a new perspective on the kind of Bildung that have for centuries been a central part of the educational goal in many parts of Europe (C. Schulte and Budde 2018). Even in countries not traditionally familiar with the concept of Bildung, the winds have changed towards empowering people with the skills and competencies to comprehend digital technologies beyond use (Tissenbaum, Sheldon, Seop, et al. 2017; Weintrop, Holbert, and Tissenbaum 2020). This points to a need for an emancipatory computational literacy that allows people to read and write these technologies. This is the larger goal that the dissertation is oriented towards. Concretely, there is a need to better understand the material, social, cultural, and cognitive *conditions* for computational literacy.

This dissertation therefore investigates the relationship between the materialities of computing and computational literacy, empowerment, crises, and recovery. More specifically, it is an inquiry into the *non-conventional programming configurations* of contemporary human-computer relations. By *programming configurations* I mean those human-computer assemblies where computation and programming (rather than simple *use*) is central. Conversely, *non-conventional* are those configurations where either the human or the software (or both) fall outside the conventional programmer-environment

[1] http://www.catb.org/jargon/html/R/runes.html (visited Jun. 8 2023)
[2] http://www.catb.org/jargon/html/C/casting-the-runes.html (visited Jun. 8 2023)
[3] http://www.catb.org/jargon/html/P/parse.html (visited Jun. 8 2023)

forms, for instance humanities students who learn programming in a mandatory course; nanoscientists who are not trained programmers; computational media that scaffolds computing activities; and novel environments for software development. The quote by the nanomolecular scientist above illustrates someone who is not able to "cast the runes" despite their dependence on those runes for both daily work and future career opportunities.

The issue cannot be reduced to a purely mental phenomenon. Despite the popularity of computational thinking in computing education and the encouragements that everyone benefits from learning to think through processes of abstractions, generalizations, and algorithms (Wing 2006; Curzon et al. 2019), the troubles experienced by nanoscientists, students, and programmers in my research seemingly do not come from a lack of computational thinking skills. Rather, there is something in their *environment*, in their computational lifeworlds, that demands attention. In turn, this suggests a reorientation towards the material conditions, as advocated by various new media scholars (e.g., T. H. Nelson 2003; Kay 1984; Kay 2013a; Mateas 2005).

Computational media as envisioned by, among other diSessa (2001), suggests precisely this kind of reorientation towards new media forms and the ways that literacy grows from them. Computational media is a software paradigm that dissolves any *a priori* distinction between use and development and between document and application, and diSessa envisioned that these media forms might be suitable for the development of material intelligence. Vee (2013), drawing on, among other, diSessa, argues that once technologies become infrastructural in societies, they demand not only material intelligence but literacy. However, *literacy* is not enough. For instance, digital literacy offers mainly use-oriented perspectives on the digital realm (Bundsgaard 2017) and does not take into account the computational capabilities of the computer beyond information processing. According to Dirckinck-Holmfeld, Nielsen, and Webb (1988), such a literacy is adaptive and requires the individual to conform to its circumstances. Instead, they bring forth an admirable counterpoint: True literacy is emancipatory, enabling the individual to transcend and transform themselves and their world in the process.

Parallel to the historical change from an emancipatory perspective of computational literacy towards a utilitarian view of computing as is the current Zeitgeist in much of the US and Europe, the perception of software and media followed along. The grand visions of computational media that blend development and use and allowed for transformative experiences were never realized on a larger scale. Instead, the focus largely turned to efficiency, usability, and incremental innovations (diSessa 2001, p. 232). A renewed focus on the mediating aspect of computers thus allows us to transcend the narrow image of human-computer relationships in terms of *use* and *interaction* as also emphasized by, e.g., Suchman (1987) and Hornbæk, Mottelson, et al. (2019). The visions from the history of computing combined with the empirical groundedness of software studies and the postphenomenological insistence on mediation all serve to remind us that computing is *more* than use. By providing the proper media, the mutual transformation of both software, world, and individual can culminate in a better understanding of all three. This is the potential of emancipation through literacy, afforded by the proper social, cultural, and material conditions.

1.1 Research question and contributions

Through my research, I have therefore been seeking to contribute to a better understanding of the foundations of computational literacy through three prongs: An investigation into computational literacy as concept and educational goal; an investigation into the mediating qualities of software for programming; and an empirically founded investigation of the higher-order challenges and opportunities of computational literacy and software for programming. This leads me to the following research question:

Research question How do the mediating qualities of software for programming contribute to the development of computational literacy?

This research question frames my work in terms of the material conditions of computational literacy. More specifically, it allows me to go beyond a simple use-development dichotomy and address the mediation afforded by computational media and other software for programming. A focus on mediation emphasizes the mutual co-creation of software and people and reopens questions of programming paradigms, mastery and empowerment, and inscribed user roles. The mediating qualities of software go beyond use-oriented qualities and software affordances by engaging in interactions, semiotics, distribution, ethics, and transformative effects. In this respect, mediating qualities do not reside *in* the artifacts, but emerge through continued interaction over time. Further, I do not mean to suggest a purely causal relationship between these qualities and the development of computational literacy. It is rather a matter of gaining a deep understanding of the possible ways that software acts in and shapes the human-computer relationship (Dourish 2014a).

Through answering the research question, I provide the following contributions:

1. A theoretical model of self-concept as an explanatory framework for computational literacy development
2. A holistic model of technological mediation bridging science and technology studies, human-computer interaction, and computational literacy
3. A mediation analysis of a selection of distinctive human-computer configurations in programming history
4. An empirically grounded analysis of the mediating qualities of computational media
5. An empirical investigation of the relationship between computational literacy, technological mediation, and programming software
6. A taxonomy of the foundation for computational literacy

1.1.1 Positioning the contributions

My contributions are of particular interest to researchers interested in new media and computational literacy regardless of their fields. More concretely, the dissertation provides a valuable contribution for those who push for computational thinking in curricula and policymaking. Similarly, it serves as a counterpoint to utilitarian views of computing by emphasizing the emancipatory opportunities in computational literacy development. In this respect, it also serves to highlight the limits of a focus on fostering digital literacy and computer literacy, respectively.

HCI researchers interested in the nature of human-computer interaction will also benefit from my contributions, particularly the historical analyses of computing configurations and the investigations of computational media as alternative software paradigms. Further, the interdisciplinary character of the work should be relevant to philosophers of technology and researchers in software studies interested in a closer coupling with HCI of their respective fields. Specifically, this coupling provides new opportunities for empirically grounded investigations of mediation.

Finally, the self-concept framework and the contributions to computational literacy is useful for computing education researchers. By providing a holistic model of computational literacy and the conditions necessary, researchers of computer science education and computing education gain a more thorough view of the complex interplay between individuals and their computational lifeworlds.

1.2 Overview of the dissertation

Being an article-based dissertation, there are two parts. The first part is the overview article, while the second part are research articles. The overview article is structured in the following way

Chapter 1 is the introduction which provides context for my research and frames the dissertation. The introduction also presents my contributions and positions them in the larger research fields.

Chapter 2 is methodology. Here, I first position my work in an epistemological frame, after which I present and contextualize the methods I have used. Finally, I present an overview of my research and a more concrete elaboration of the individual inquiries.

Chapter 3 first presents a definition of computational literacy in contrast to other literacies and related concepts such as material intelligence and competencies. Then, a self-concept framework is introduced as a promising explanatory model of the development of computational literacy.

Chapter 4 brings together interaction and mediation. After presenting both concepts from HCI and postphenomenology, respectively, I bridge these fields in an integrated model of mediation with differentiated aspects.

Chapter 5 is about computational media and software mediation and consists of three parts. I have selected distinctive programming modalities from computing history and present mediation analyses of them. Afterwards, I present computational media and literate computing[4] as alternative materiality and activity for computational literacy, respectively. This is followed by an empirically founded analysis of computational media as technological mediation in practice.

Chapter 6 connects computational literacy and the mediating qualities of software through a number of empirically grounded themes. The relationship between computational media, literacy, and visions is examined which is followed by a discussion of computational culture. Afterwards, I present the crises, workarounds, fluency and success related to computational literacy in action as experienced by people in my research. The chapter ends with a taxonomy of requirements for computational literacy to flourish and implications for HCI and computing education, respectively.

Chapter 7 provides a conclusion and an answer to the research question.

The second part of the dissertation consists of the published and submitted research articles that form the foundations of my work, attached as appendices. An overview of these can be found in table 1 on page iii.

1.2.1 Some notes on reading the dissertation

The structure of the dissertation does not reflect the chronological nature of the research activities that preceded it. For a chronological presentation of the concrete activities and how they connect in a hermeneutical manner, see section 2.3. The research activities are a series of studies, but one study can consist of multiple research engagements. For instance, the study of nanoscientists consisted of both an ethnographic fieldwork phase and a constructive design research and evaluation. For this reason, the smallest unit of research in the dissertation is the *inquiry*. When referencing my research activities, I therefore sometimes refer to the individual inquiries rather than the larger studies. An overview can be found in table 2.1 on page 16.

As I have conducted ethnographic fieldwork with multiple groups of participants, I distinguish them in the following way: S1–5 are the students, N1–11 are the nanoscientists, and P1–12 are the programmers from the game design study. This nomenclature should make it a little easier for the reader. Finally, the reader is advised that the two empirically grounded chapters (chapters 5 and 6) are interspersed with quotes. As a phenomenological study of literacy and mediation *in action*, the quotes serve to illustrate the richness of people's lived experiences as well as preserve the ambiguities and complexity of these.

1.3 A short reflection on the development of the research design

To some degree, I have allowed the ongoing progress of my research project to dynamically unfold in the ways that I deemed most promising. Concretely, that means that I let the insights from one inquiry feed into the next in an iterative process. This is in part due to me being open to opportunities and promising research directions, and partly due to factors outside my control, for instance the COVID lockdown which prevented me from doing some planned activities, making it necessary to adjust the original plan significantly. A map of this development can be found in figure 2.1 on page 15.

[4]Literate computing is a computing paradigm that blends computing and rich media such as text and multimedia (see section 5.4 for a more thorough presentation)

1.4 Brief summary of the articles

"Mapping the Landscape of Literate Computing" (Fog and Klokmose 2019) is an investigation of so-called literate computing environments. Through an artifact analysis of commercial products, research prototypes, and other software artifacts that exhibit literate computing characteristics, we framed these environments through a series of relevant themes such as communities, use, development, etc. The article concludes with a set of design considerations for developers of these kinds of environments.

"Between Scripts and Applications: Computational Media for the Frontier of Nanoscience" (Nouwens, Borowski, et al. 2020) was the result of a multiphase study of biomolecular nanoscientists who work on the cutting-edge of science. The first part of the study was an empirical inquiry of the material and sociocultural working conditions of the nanoscientists, i.e., their computational tools, scripts, and environments and the computational culture(s) in which they work. The second part was the development, introduction, and evaluation of a computational lab book for their work, based on the principles of computational media.

"Between Principle and Pragmatism: Reflections on Prototyping Computational Media with Webstrates" (Borowski, Fog, et al. 2022) encompasses multiple studies. The first half is a retrospective inquiry into a family of computational media developed and used by the research group that I have been part of. From this inquiry we draw out a series of tensions between visions and practicalities of computational media. The second half is an empirically grounded inquiry into the possibilities and challenges of the newest iteration of this family, namely CODESTRATES v2. This was conducted through what we called a game design challenge that sought to leverage the unique characteristics of computational media. Based on this work, we present an analysis of the challenges and opportunities of computational media and a series of lessons learned for future designers of such media.

"KnitxCode: Exploring a Craftsmanship-driven Approach to Computational Thinking" (Sørensen et al. 2022) presents results from a workshop conducted with experienced knitters that aimed to leverage crafts and arts as a springboard towards computational competencies. In the workshop, which involved 12 knitters without previous programming experience, we utilized knitting patterns and material knowledge to introduce computational concepts such as loops, variables, and algorithms. This was followed by a physical computing exercise in which participants created artifacts relevant to their knitting practice.

"Computational self-concept: Towards an understanding of students' identities, attitudes, and beliefs" (Fog, Pálfi, et al. 2023) presents computational self-concept as a psychological construct which provides a framework for understanding how people develop identities and competencies in computing. Part of the article is a theory transfer from psychology, while the latter half is an empirical study of five humanities students and how they build competencies and identities throughout a computing course.

Methodology and research design 2

Dealing with failure is easy: Work hard to improve. Success is also easy to handle: You've solved the wrong problem. Work hard to improve.

Perlis (1982)

In this chapter, I will present and discuss the methodological considerations that my research question gives rise to. To reiterate, my research question is as follows:

Research question How do the mediating qualities of software for programming contribute to the development of computational literacy?

In the coming chapter, I address three different but interrelated aspects of the way I have answered said question. First, I characterize the research question and describe my epistemological positioning. In other words, I show what kind of research question that is at play and the possible ways of answering that question within my epistemological frame of reference. Having provided an epistemological framing of the research question, in the second part I position my research methodology. This consists of a discussion of specific research traditions and methods and the concrete methods employed in my studies. Finally, I present my research design and strategy for the dissertation as a whole. This section answers the question of my specific engagement with the world in an intentional and deliberate way of seeking answers to my research question. This following chapter will, in other words, establish accountability and transparency for the approach taken to answer the research question.

2.1 Epistemological reflections

Epistemology denotes what kinds of knowledge can be claimed. Already the phrasing of my research question demarcates a line of reasoning that defines the type of answer that can be given. And while the research question could be answered in a multitude of ways, some are more valid than others. My dissertation draws on a knowledge framing from postphenomenology which takes its epistemological and ontological position from pragmatism and phenomenology. In this section I elaborate why this is the case, what it means for my research, and how this choice affects the way I can engage with the world as a researcher. Different epistemologies do not necessarily conflict. Of course, any particular inquiry must be interpreted within a given epistemological frame, but the dissertation as a whole, as it draws together multiple studies conducted over several years, can meaningfully be built from multiple epistemological positions (Creswell 2014, p. 82). Rather than diminishing each other's claims to knowledge, multiple ways of knowing can co-exist and positively supplement each other in answering a question (Tracy 2013, p. 47).

My studies largely draw on a postphenomenological approach to knowledge creation which posits a particular arrangement between the knowing and the known. Postphenomenology, as it originates in pragmatism and phenomenology, sees no *a priori* world out there. There is no world without interpretation (Overgaard and Zahavi 2009). A consequence of this position is that there is no inherent separation between ontology and epistemology (Jacobsen, Tanggaard, and Brinkmann 2020). Another consequence of my epistemological position is that it allows me to say some things and not say others. I do not claim that any of the findings in the present dissertation are absolute and unquestionable. They are deeply dependent on the contexts in which they were created and presented. Further, they shape my space for action in that only some methodologies and concrete methods fit in well with this particular epistemology.

This dissertation does not aim to provide any definitive, falsifiable answers to the question. I am not working within a positivist (or even post-positivist) research paradigm that seeks to understand the world *as is* through research ideals such as replicability, reproducibility, and validity. There is no "view from nowhere" in my research (Haraway 1988). The ideals to which my dissertation is to be measured by are instead criteria such as transparency, rigor, sincerity, and credibility (Tracy 2010). Considering my epistemological positioning and the character of qualitative research, the answers will be contextual and affected by my position as a scholar and a human being. While I do not seek to answer my research question in a way that claims totality and complete generalizability, my contributions are still valid in their own right. They must be understood as products of a particular engagement with the world through me, the researcher, as a tool, but they nonetheless bear significance, coherence, and transferability.

Having so far laid out my epistemological considerations and characterized my research question, the next section will position my research methodology.

2.2 Positioning the research methodology

Most of my research has been empirically informed. One exception, the creation of the self-concept framework, is not treated in this section, but is addressed in the later section 3.4. One important facet of using postphenomenology as ontological and epistemological frame of reference is that,

> (...) there is no strict postphenomenological methodology that scholars could follow. Postphenomenology comes in just as many flavors as there are scholars in the field. (Rosenberger and Verbeek 2015)

This is not particularly helpful for approaching postphenomenological research. In part, it stems from the fact that postphenomenology can trace its origins to the philosophy of technology and does not encompass a research methodology. A pertinent question to ask, given this methodological free-for-all, is how one might distinguish between postphenomenological research with other kinds of research. Or even whether it can provide any form of (internal) validity and thus be considered research proper (cf. the demarcation problem). To avoid such criticism, I have used a variety of well-established research methods from HCI research and related fields such as anthropology and interaction design. In the coming sections, these are unfolded more fully. Postphenomenology for me thus becomes not a way of doing research, but a way of defining the objects of interest and a way to understand my findings as pertaining to the particular. In later chapters, these investigations of the particular will be tied to my broader research question and beyond, seeking not to present generalizable findings but to try and answer the research question from the concrete human-technology relationships that I have investigated.

I have worked in a variety of methodologies under the grander themes of qualitative research and constructive design research. In this section I will position my concrete choice of methods into larger methodologies as well as relating these methodologies to my epistemological basis. The methods discussed, e.g., observations and interviews, will not be unfolded in their concreteness. What this means is that concrete details such as the numbers of participants, format of interviews, and workshop prompts will be illuminated later in the manuscript when presenting the individual inquiries. The section, on the other hand, puts forth an argument for the appropriateness of the methodologies in regard to my research question. Any epistemological framing naturally affords certain methodologies and traditions while barring others. At the same time, I am inclined to follow Creswell's notion of the "pragmatic worldview" (Creswell 2014, p. 90), in that my focus is on answering the research question, and that concrete methods can be considered valid insofar as they help understand and answer the problem. The fundamental issue of HCI research being, in essence, interdisciplinary and drawing from other, more established fields is well-known and discussed (see, e.g., Mackay and Fayard 1997).

The dissertation is based on several research traditions, each of which have their own established methods, assumptions, and judgement criteria. In the following I present qualitative research and constructive design research methodologies and argue why my chosen methods are the best approaches to answer my research question.

2.2.1 Qualitative research

While phenomenology originated as a mainly philosophical tradition, it has since become a viable way to conduct qualitative research (Aagaard 2017). A postphenomenological approach to qualitative research thus brings the perspective that people and artifacts are closely related. Further, the phenomenological perspective acknowledges that people are not separate from the activities and contexts in which they engage. Therefore, to gain a deeper understanding of the relationship between concrete people and software as sociotechnical configurations, it is important to study them in-the-wild. To this end, qualitative research that takes seriously the context of practice is highly useful.

The concrete qualitative methods that I have employed in my work largely come from a set of established ethnographic methods. Qualitative research must be judged from a distinct set of criteria, one of which is the focus on, e.g., transferability rather than generalizability. This means that qualitative research should not be confined to the individual study, but rather resonate with a larger space beyond the immediate context. This criterion is highly commensurable with the postphenomenological focus on the particular. By starting from the particular people and artifacts under study, I can then broaden out and discuss the wider implications of my findings. My ethnographic work in the knitting workshop, for instance, cannot give any truth about a larger population of knitters. However, by looking at these particular people in a certain context, the findings are also suggestions that illuminate certain things to be explored further. They are both findings in their own right and prompts for future work.

Validity in qualitative research is judged by notions such as "trustworthiness, authenticity, and credibility" (Creswell 2014). This can be achieved through efforts like triangulation, self-reflection, interrater reliability and similar checks that I have sought to incorporate into my research. For instance, regarding my overall research question, the process of answering that has been in the form of a continuous, hermeneutic triangulation marked by transparency in methodology and challenges and a choice of research methods that match my epistemological stance (Tracy 2010).

2.2.1.1 Ethnographic methods

Ethnographic methods come from fields such as anthropology in which they came into being as long-term engagements with the people under study. A long-term engagement is generally not possible or desirable in HCI research. In this context, I have adapted them to the particular timescale of my research, in line with Hanington's view:

> These adapted methods serve to condense the extraordinary time devoted by formal ethnographers into more manageable and ultimately more relevant samples of information. (Hanington 2022)

For one inquiry in particular, though, I followed a group of scientists over a time period of more than a year in which my fellow researchers and I built a closer relationship with the participants. This allowed me to build trust and mutual engagement with the participants in question. According to Aagaard and Matthiesen (2016), qualitative methods such as observations and interviews, particularly in combination, are well-suited for postphenomenological investigations of materiality. Further, ethnographic methods are well-established in HCI research and have been so for several decades (Dourish 2014b). The use of ethnographic methods such as observations and interviews are therefore particularly useful in answering my research question.

As my research concerns the mediating aspects (or, mediating qualities) of computational media, I engaged in participant observations as a way to gain insight into the concrete interactions as they took place in both natural and artificial settings. By natural setting I refer to the contextual inquiry-based observations (Holtzblatt and Jones 2017) that I did in the case of the nanoscientists' own work practices. In contrast, I also did observations in the "lab"[1] of a different group of people, knitters.

[1]The word is used here to denote the opposite of in-the-wild

The purpose of investigating nanoscientists in context was to better understand the sociotechnical circumstances under which the scientists perform their daily work as well as inspect the interaction between scientists and their tools in action. My observations among the knitters had a related, but ultimately different goal. They were intended not to illuminate any existing work practice, but rather to investigate the meaning-making that took place among knitters who encountered programming and used relevant software for the first time.

Interviews are, like observations, a stable in qualitative research, and their validity as methods are well-established across a variety of disciplines including HCI. The purpose of using interviews to answer my research question is not to get to any *one truth*, but to get insights into participants' experiences and meaning-making processes. The pragmatic and phenomenological approach indeed argues that experiences of being-in-the-world are the main object of interest. I conducted interviews both as a supplement to observational studies and as a stand-alone engagement with university students. Interviews as a method allowed me to probe into the conceptions held by participants, for instance, in interviewing university students about their feelings of identity and self-concept in relation to programming. Here, group interviews were used as a method as the purpose was to encourage dialog between participants and foster mutual inquiry into the subject (Tracy 2013, p. 167). For my study of the nanoscientists, interviews were used to supplement observations as part of the contextual inquiries to substantiate and clarify their experiences with the prototype. In the case of the *KnitxCode* workshop, impromptu interviews (also called ethnographic interviews (ibid., p. 140)) were used to inquire about particular situations in situ as they took place.

2.2.1.2 Technographic methods

This term, technography, denotes a set of methods (and, arguably, a methodology in itself (Jansen and Vellema 2011)) that allows for investigating the relationships between people and artifacts (Kien 2008). The goal of using technographic methods was to gain a deeper understand of the actual artifacts, their affordances, and their mediating qualities. Verbeek provides an important starting point to avoid perceived pitfalls such as essentialism: "[A]n approach to technology in terms of concrete technological artifacts is essential in the philosophy of technology." (Verbeek 2005)

We must, in other words, start from the concrete being-in-the-world of the technologies. This need to study artifacts in use is supported by, e.g., Bannon, Bødker, et al. (1991). Aagaard (2017) more concretely presents two suggestions for how to approach such a study of the concrete artifacts: An in-depth exploration of the typical use of a given technology or a critical comparison of multiple versions of a technology. Through the various inquiries and engagement with the technologies under study, I have used the latter of these as well as supplementary methods. For example, my study of literate computing environments represents a critical comparison of multiple versions of a technology. In doing so, the otherwise subtle differences between systems become emphasized, allowing me to address these as manifestations of broader themes such as metaphor, script, and user community. Engaging technographically, the software takes on the role of a participant. A main difference is, as should be evident, that artifacts do not speak by itself. Considering that mediating qualities of artifacts is a central part of my research question, technographic methods are a way of interrogating the artifacts and their qualities, not in terms of their concrete effects on people but in the ways that values, preconceptions, and scripts for interaction are inscribed into them.

Technographic methods have gained legitimacy, especially in research on databases as epistemological technologies. For instance, Schuurman (2008) draws on STS and ethnomethodology to inquire into databases as they act on the world around them. Although not providing concrete methodologies, this approach constructs the artifact as an object of study beyond a simple tool perspective. A similar approach can be found in Burns and Wark (2020) for whom the technology acts as basis for subsequent ethnography. As their objects of study are databases, their focus is on the epistemological and ontological characteristics of the artifact. However, their approach lends a very useful perspective that,

> digital objects should be understood as events—with questions around what they do—rather than as static objects—with questions around what they are (ibid.)

My approach to the investigation of the mediating qualities of software benefits from this perspective.

To understand how computational literacy is shaped by material conditions, the focus of what technologies *do* is central. This aligns well with a postphenomenological view of artifacts as more than simple tools for use. Finally, the research field of software studies has a rich tradition of technography. As supported by Dourish (2014a), starting from the concrete technologies as contingent manifestations of multiple historical alternatives is central. By providing a historical perspective of programming modalities (see section 5.2), I make the argument that programming could be different and that various modalities bring about different levels of abstraction, opacity, and mediation.

Finally, software studies, just like database studies, emphasizes the ways that software can "constrain, shape, guide, and resist patterns of engagement and use" (ibid.). Thus, database studies and software studies provide both an ontological perspective of software as objects that act on the world and concrete methodological considerations of starting at the concrete materiality. The retrospective view of the *-STRATES software family[2] and the historical overview of programming modalities draw heavily on these kinds of technographic methods. Finally, my technographic investigations of computational media have been heavily informed by Fuchsberger, Murer, and Tscheligi (2013) who argue for a media perspective on digital technologies that draws on actor-network theory. Using this media analysis approach has allowed me to better understand the ways that computational media and programming environments are not solitary objects but inscribed artifacts that constitute people and practice in particular ways. The analyses of programming mediation in section 5.2 is a clear example of the possibilities of such an approach.

2.2.1.3 First-person methods

The final type of qualitative research methods that I employed in my studies is what can be termed a retrospective sociotechnical autoethnography. Using this method provides a way of opening the established research narrative in which research results are often presented as successful endeavors even if this ultimately might result in, e.g., publication bias (Song et al. 2010). This is equally the case in HCI, and likely even to a higher degree when the research contribution also encompasses a technological artifact.

My approach to this is similar to, e.g., "retrospective trioethnography", an approach that addresses constructive design research previously published as successful contributions but readdresses them as learning experiences (Howell, Desjardins, and Fox 2021). By doing a first-person inquiry into the computational media it was possible for me to critically trace the genealogy of the *-STRATES family of computational and reflect on the decisions that formed their design. In doing so, the computational artifacts can become destabilized; no longer conceived of as any "natural" or obvious evolution but rather as a series of contingent and contestable materialities, all of them the results of particular engagements with the world (Bevir 2008). As my research question pertains to software for programming, a retrospective approach to technography allowed me to view the different manifestations of computational media and reflect on how their mediating qualities were shaped by a particular set of values, principles, and practical circumstances. As such, it also helps qualify the question of materiality in terms of computational literacy.

2.2.2 Constructive design research

Some of my inquiries into my research question have been in the form of a constructive engagement with the digital material. The discussion as to the type and extent of the knowledge that can be generated through such an interventionist or designerly practice has been an issue in fields such as interaction design and HCI. In interaction design research, for instance, the notion of research-through-design has been defined (Frayling 1993), explored (Dalsgaard 2010; Gaver 2012), and critiqued (Zimmerman, Stolterman, and Forlizzi 2010), making it an established methodology tied to certain traditions and epistemological assumptions. Another field of study, educational research, has similarly used a methodological approach called design-based research for many years (T. Anderson and Shattuck 2012). Despite these similarities in nomenclature, their epistemologies and methodologies are quite different. Even within the individual disciplines, there seems to be a lack of agreement on, e.g., what kind of knowledge is generated (Höök et al. 2015).

[2]The *-STRATES software family is my collective term for the group of computational media that are built on the WEBSTRATES platform (see more in section 5.3.2)

Instead of using these terms, my way of making inquiries into the world by intervening and designing is to be seen as *constructive design research* (Koskinen et al. 2011). While the concrete activities might still be the same—constructive design research is not *a method*, after all—the name serves to discard the methodological baggage of the contested terms and start with a blank slate. Under the umbrella term of constructive design research, my inquiries are understood as knowledge-generating activities that align closely with other disciplines. While constructive design research is not tied into one particular epistemology, the field is built upon and grew from pragmatist and phenomenological philosophies.

Koskinen et al. (ibid.) present three 'spaces' in which design research might take place. Using their conceptual terminology, these are considered the *lab*, the *field*, and the *showroom*. These terms provide a way to position my constructive design inquiries, e.g., the computational lab book, into a certain space such as the *field*. Investigating the computational lab book in the field is well-aligned with the other kinds of knowledge-producing activities that I have employed such as contextual inquiries, observations, and in situ interviews. The constructive design research approach that I have employed likewise draws from participatory design practices and, to a lesser degree, action research (Hayes 2014). However, action research is fundamentally about creating sustainable change through intervention. In the case of the computational lab book, the nanoscientists were left with only a functioning prototype after the conclusion of the study without properly "ensuring that the technologies can be left behind and if left behind can be maintained" (Hayes 2011). This is an ethical challenge that is addressed in chapter 6.

A particular aspect of participatory design has informed my work. While some axioms of the first wave of participatory design, like the strong political agenda and the assumption of conflicting interests between workers and employers, were not central to my work, participatory design's focus on workers and their skills has nonetheless been influential in my approach to, for instance, the nanoscientists under study. For the purpose of this dissertation I will stick with the concept of *constructive design research* as the type of engagement that I have done. However, an important point of using this term is that more specific approaches, such as participatory design, can be understood within the 'spaces' of constructive design research (Zimmerman and Forlizzi 2014).

I chose to engage in constructive design research as my research question explicitly treats the digital artifacts as the object of study. Using the qualitative methods allowed me to probe into the state of Things[3] as they are. In contrast, by engaging in constructive design research, I was able to explore what might be. In the case of the nanoscientists I was thus able to not only investigate their current use of digital tools, but also to create a possible future for the participants (Salovaara, Oulasvirta, and Jacucci 2017). The concrete computational lab book, while interesting in itself as a design exemplar, became something else in the subsequent deployment and evaluation among the very same people. It allowed me to talk to participants *through* the prototype and bring to light the taken-for-granted and invisible aspects of their current work practices. The prototype therefore both represents a concrete artifact in itself, open to analysis and interpretation, and acts as a vessel into which assumptions can be questioned. However, as demonstrated by Salovaara, Oulasvirta, and Jacucci (ibid.), the use of prototypes as means for investigating both the present and possible futures brings its own set of epistemological issues regarding how to evaluate the prototype. While I, of course, need to be reflective about the in-betweenness of the prototype (part artifact, part vision), my main use of the prototype in the context of this dissertation is to investigate current practices. A final remark concerns the type of knowledge to be gained from constructive design research. There is an inherent contradiction between the knowledge production of design and of research (Fallman 2007; Bødker, Dindler, et al. 2022). Research seeks to produce generalizable (or transferable) knowledge, while design produces localized knowledge. Through the constructive design process of working with the computational lab book, we generated knowledge about a particular artifact in a particular group of people. However, I do not so much aim to evaluate the prototype itself, as I aim to use it as a springboard (Engeström 2015) for a better understanding of the current situation.

[3]I use the concept of Thing in the sense of "matters of concern" put forth by Latour (2004)

2.3 Research design and strategy

To answer my research question I have investigated three threads of inquiry. One thread concerns the material conditions (i.e, the software). A second thread concerns the relationships between people and their digital artifacts; that is, the mediating qualities that shape particular kinds of engagement. The final thread is the people themselves and how they think and act through and on technology. This separation into threads is artificial and does not neatly line up with the divisions of the dissertation. In practice, they might be represented as a series of inquiries in a continuum between the purely technical and the purely human. Further, the distinction between threads is mine for the purpose of characterizing my research. Each individual inquiry in my research takes into account both the human and the technical aspects. In my analysis of literate computing environments, for instance, even if the target of my work is the digital artifacts, I also keep a firm view of the (potential) people using them.

Considering the individual inquiries as a larger pattern of investigation into the research matter at hand, this larger pattern is both triangular and hermeneutic in nature. Approaching my research question using a variety of methods, approaches, and perspectives has allowed me to gain a broader, deeper, and more nuanced understanding of the mediating qualities of software and the people using them (Thurmond 2001; Tracy 2010). My approach, as a series of continuous inquiries that inform each other, further benefits from the hermeneutically connected insights. It is therefore not only the case that I have investigated computational media from two perspectives, for example. Instead, the fact is that each inquiry has led to new knowledge which has since fed into the subsequent inquiries in a hermeneutic phenomenological fashion (Kafle 2013). There is, therefore, both a parallel and a serial relationship between them.

The scholarly work upon which the present dissertation is written can be found in part II. However, it is not simply the case that each research output, for instance a conference contribution, equals exactly one research activity. Each engagement with the research "object"[4] I have termed an inquiry. Through my research design and strategy I have continually moved between the parts and the whole, each inquiry drawing from the ones preceding it and informing the subsequent ones. To show this process in the context of my research work, the map in figure 2.1 illustrates my various inquiries:

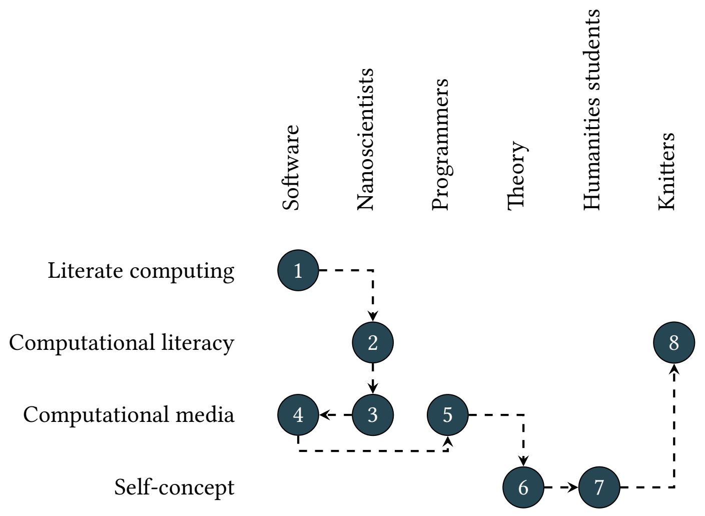


FIGURE 2.1: A map showing the development of inquiries. The y-axis shows the various themes or strands of inquiry, while the x-axis are the central phenomena (i.e., objects of study) of those inquiries.

The rest of the chapter presents the individual inquiries that form the basis of this dissertation. For each inquiry is described what decisions led to it, the inquiry in brief, the methods and context, and finally how the outcome of the inquiry led my research forward. The research question has framed a certain direction for my studies that the inquiries follow in a hermeneutic way. Table 2.1 on page 16 gives an overview of all inquiries.

[4]I use the term object in this dissertation to mean the matter under study

#	Name	Methods	Empirical foundation	Research output
1	Literate computing environments	Artifact analysis; affordance analysis	12 software artifacts	Fog and Klokmose (2019)
2	Study of nanoscientists	Observations; interviews	12 participants	Nouwens, Borowski, et al. (2020)
3	Design and evaluation of computational lab book	Participatory design; in situ interviews	4 participants	Nouwens, Borowski, et al. (2020)
4	Reflections on the *-STRATES family genealogy	Technography; retrospective trioethnography	*-STRATES family of software	Borowski, Fog, et al. (2022)
5	Game design challenge and interviews	Interviews	12 participants	Borowski, Fog, et al. (2022)
6	A theory of computational self-concept	Theoretical transfer	-	Fog, Pálfi, et al. (2023)
7	Student self-concept interviews	Group interviews	5 participants	Fog, Pálfi, et al. (2023)
8	KnitxCode workshop	Workshop; observations; interviews	12 participants	Sørensen et al. (2022)

Table 2.1: An overview of my research inquiries

2.3.1 ① Literate computing environments and characteristics

As a starting point for my research, I approached the concept of literate computing, a computing activity especially popular in the data sciences. Literate computing is a relevant phenomenon to investigate in relation to computational media as literate computing activities largely produce computational media as their output. As my research question specifically addresses the mediating qualities of software, I found that the literate computing paradigm presents a different kind of human-computer interaction that combines computing with textual production and multimedia in the same conceptual space. The original vision of literate computing was to bring use and development closer together by enabling the creation of interactive media and computational narratives. Literate computing therefore represented a promising programming paradigm in the larger scope of the development of computational literacy. Simultaneously, I was involved with the early stages of inquiry 2 and 3, and I wished to learn more about literate computing as a phenomenon, since inquiry 3 was guided by the development of a computational lab book based on literate computing principles.

To investigate literate computing software, I followed the postphenomenological suggestion from Aagaard (2017). I selected twelve software environments that either directly allowed for literate computing activities or exhibited characteristics of the paradigm. Each system was analyzed on the basis of a number of themes. To guide the analysis, I took inspiration from Davis and Chouinard (2016) who provide a list of verbs for understanding artifact affordances. While the analysis is not an affordance analysis in itself, the approach helped me to articulate my findings in more nuance, for instance using qualifying verbs (encourages, discourages, enables, demands, etc.). In a grounded theory approach, the final themes were inductively created based on my engagement with the tools. Through this recursive movement between emergence of themes and analysis using themes, I ended up with a series of findings related to, e.g., users, system metaphor, collaboration, malleability, and intentionality.

This study led to my first publication (Fog and Klokmose 2019). It made it clear to me that the artifact inscriptions matter deeply, for instance, how the software metaphor shapes how a user sees and interacts with the system. It also highlighted how there might be a relationship between the interactional qualities of software for programming and the computational literacy necessary to not only use them but become fluid in them. In particular, computational media stood out as a way to bridge this perceived gap between developing and using software more broadly.

2.3.2 ② Study of nanoscientists

I wished to learn more about the possibility of supporting particular types of knowledge work through computational media. Further, engaging with a group of people deeply dependent on computational tools who struggle with not having been trained to use them provides me with a case where computational literacy is lacking. More specifically, with a few colleagues, I gained access to a group of scientists at the interdisciplinary research center iNANO who do research on RNA origami. This particular group was relevant for my work for several reasons. First, they work on the breaking edge of their research fields, so there are no applications or ready-made software packages that allow them to do their work. Rather, they scrape together assorted scripts and web services for their work. Second, the scientists have no formal computing education despite the fact that their work is deeply dependent on computational competencies. Third, the group under study had already implemented digital tools for research such as electronic laboratory notebooks for research documentation and collaboration and were thus willing to experiment and invest resources into digital tools. And finally, the scientists' existing tools such as scripts and web services were quite suitable for translating into a computational lab book.

The computational lab book was developed in the subsequent inquiry, but I first wished to better understand the sociotechnical context of the nanoscientists under study. While the initial engagement with the group had already begun before the start of my research process (i.e., a large portion of the ethnographic fieldwork and a future workshop), I took part in subsequent analysis of the data resulting from these studies. Therefore, while I did not create the research data, I was intimately engaged with the data and the analysis of it. I further took part in later ethnographic fieldwork during the participatory design process in inquiry 3. These two inquiries (2 and 3) are thus highly connected.

Throughout the almost three-year long engagement with the nanoscientists, my peers and I used a variety of ethnographically inspired methods to understand their current sociotechnical circumstances as well as a future workshop to gauge participants' needs and wants for the prototype. I again wish to clarify that I did not take part in conducting the initial interviews and the future workshop, but I did work with the original PI in analyzing and (re)interpreting the data after I joined the project. An overview of the timeline and involvement of researchers and participants can be found in figure 2.2. A total of 12 participants, all connected with the lab in a research capacity, took part in the study at various points.

In short, the scientists struggled with some level of computational illiteracy, and the notions of the more capable peer, computational disempowerment, and computational culture came to be guiding principles for my future work which also fed into subsequent inquiries. The results from inquiry 2 can be found in Nouwens, Borowski, et al. (2020).

2.3.3 ③ Design and evaluation of a computational lab book

Based on our findings, we believed that a computational medium might solve some of these issues related to the computational conditions. As my research question focuses on the mediating qualities of software for programming, it was relevant for me to take part in the construction and evaluation of a computational lab book for the scientists as a way to probe into their issues with computational illiteracy. Further, it allowed me to explore how a domain-specific computational medium might be realized as opposed to the more domain-agnostic exemplars such as Boxer or Webstrates.

The design process was heavily informed by a participatory design approach in which the goal was to co-create a computational notebook prototype that was "empirically researchable in the present world" (Salovaara, Oulasvirta, and Jacucci 2017). To start it off, participants were engaged in a full-day future workshop to collectively reimagine how computationally supported biomolecular nanoscience might look. Based on these suggestions, a design idea for a prototype was chosen based on its level of significance to their research and the feasibility of implementing it technically. The workshop was conducted before my participation in the project, but I took part in the rest of the study. Through the participatory design process, we were in continuous dialogue with the participating nanoscientists. More concretely, we conducted four meetings with participants and—based on these and the empirical findings from inquiry 2—we constructed a prototype which was presented to the scientists in a meeting and iterated upon along the way. An overview of the participatory design process can be seen in figure 2.2. For a more thorough presentation of the methods, the reader is referred to Nouwens, Borowski, et al. (2020). The final outcome of the participatory design process was a computational lab book, i.e. a laboratory notebook based on the principles of contemporary computational media: malleability, shareability, distributability, and computability[5]. With this computational notebook, we made in situ interviews with participants using the artifact for their actual work, utilizing the artifact as a springboard for broader discussions of computational media and participants' computational conditions.

The participatory design process and the creation of the computational lab book provided a lot of new insights for my research. For instance, it became clear how computational media is not a one-size-fits-all in contrast with the visions from Kay and diSessa. The computational medium was, in practice, a malleable and computational media for us as researchers, not for the scientists. To them, it largely appeared as a bespoke web application, not a reconfigurable medium. This study with Codestrates, among other studies not included in this dissertation, made it clear to me that the literate computing paradigm that Codestrates was built upon was not necessarily the ideal expression of a contemporary computation medium. This insight also led to the subsequent iteration of the software into Codestrates v2. It further prompted me to investigate how the mediating qualities of these genealogical iterations had changed over time which resulted in inquiry 4.

2.3.4 ④ Reflections on the *-strates family genealogy

Although I did not take part in the development of neither Webstrates, Codestrates, nor Codestrates v2, I was closely involved with those that did. My findings from the previous inquiries on computational media and literate computing made it interesting for me to reflect back on how these different materialities afforded various interactions and how these were expressions of particular

[5]These principles are explained in section 5.3.2

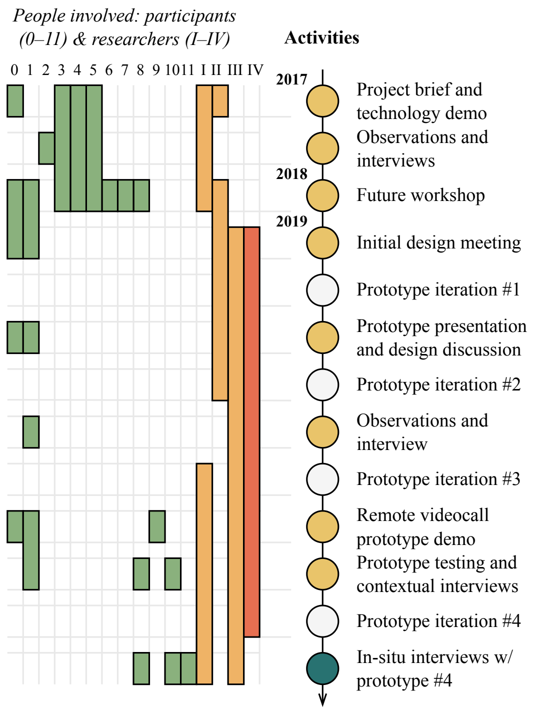


Figure 2.2: An overview of inquiry 2 & 3, leading to the development of the computational lab book. I am researcher *IV* in the model (in red). While I did not perform the actual interviews, I was still involved with subsequent data analysis.

tensions between the ideals of computational media and the practicalities of software, code, digital infrastructures, and computational idiosyncrasies. I did this with fellow researchers also affiliated with the *-strates family and interested in computational media.

For the study, I sought to trace the genealogy of the *-strates software family. We included the major platforms, Webstrates, Codestrates, and Codestrates v2, as well as auxiliary software such as Videostrates and Vistrates. To do this, we employed a first-person method inspired by, e.g., retrospective trioethnography (Howell, Desjardins, and Fox 2021) in which previously published success stories of design exemplars are reevaluated to identify the cracks in the narratives. In other words, how these outputs of constructive design research also embody imperfect design decisions. This was coupled with case studies of other software artifacts created using *-strates such as the computational lab book (inquiry 3) and collaborative software for public libraries. Based on our genealogical analysis, we identified eight tensions related to computational media built on Webstrates. Most of these are concerned with mediating qualities such as directness in editing and malleability versus stability. These tensions combined with the results from inquiry 5 led to us pointing out six broader "lessons learned" that other researchers in computational media should take away.

An important insight for me was the realization that the issues with computational media are not only specific to interactive qualities, but also point to broader questions of how to bridge the idealistic vision of computational media and the practicalities of technical limitations and people's existing life-worlds. This inquiry provided a few realizations regarding concrete manifestations of computational media. Through the retrospection, it became clear that computational media is a promising software paradigm for computational literacy, yet difficult to realize in practice. In the constructive design process of interactive systems, interaction cannot be predicted, only suggested. This is even more true in the case of reconfigurable, computational media that do not provide one predefined user interface. I wished to examine how these mediating qualities might actually play out in practice which led me to the subsequent inquiry (5), which would allow me to investigate the links between these qualities and the computational literacy of user-programmers.

2.3.5 ⑤ Game design challenge and interviews

To investigate how the mediating qualities of computational media can influence people's skills and self-perceptions, we decided to conduct a study of the newly developed Codestrates v2. This would allow me to understand better how the particular implementation of a computational medium, Codestrates v2, was experienced by programmers, thereby probing into how these people and the artifact would co-create each other in the mediating experience. Further, by exploring the multiplicity of roles embodied by Codestrates v2 in the interaction, I would be able to learn more about its capability for influencing computational literacy.

The study was designed as what we called a "game design challenge". We recruited people through an existing Slack space for Webstrates, Twitter, and through direct engagement with potential participants, aiming for a group of participants with varying degrees of experience with both Webstrates and Codestrates v1. Participants were required to have experience with web programming and JavaScript. 23 people signed up for the study, and 12 of these participated in the full game design challenge and the subsequent interviews. An overview of their previous experience and skills can be found in table 2.2. For more information about participants, see Borowski, Fog, et al. (2022).

Participants	P1	P2	P4	P5	P6	P7	P8	P9	P10
Programming	8	2	8	8	6	8	7	8	10
Web development	6	3	7	5	6	7	8	6	5
JavaScript	8	2	9	6	6	7	9	7	8
Game development	5	4	5	1	3	1	2	2	6
Webstrates	8	3	7	7	2	9	4	5	1
Codestrates v1	4	3	5	6	2	1	2	5	3
Developed with Webstrates	yes	no	yes	yes	no	yes	no	no	no
Developed with Codestrates v1	no	no	yes	no	no	no	no	yes	yes

Table 2.2: Overview of the participants' self-assessed programming knowledge on a scale from 1 (no knowledge) to 10 (expert), and whether they developed with Webstrates and Codestrates v1 before. (Only participants that filled out the demographic questionnaire are listed.)

Participants' ages ranged from 24 to 37. Most were PhD students, postdocs, or researchers, while one was a software engineer. Due to the ongoing COVID-19 pandemic, the whole study took place online. Participants were sent a design brief and given three weeks for their projects. The design brief contained a task description, software documentation, a few inspirational game projects, and a tiered list of goals. The goals were to make a small game that, ideally, would make its rules (collaboratively) editable by players. Further, participants were given a set of constraints, most importantly to *only* use the built-in editor Cauldron for the project (for more details about Cauldron and Codestrates v2, see section 5.3.6). After developing their projects, participants were invited to a demonstration day in which they shared their creations with each other and us. The participants were subsequently interviewed in eight semi-structured interviews (some had worked in groups and were interviewed together). Themes covered were, among other, the games themselves, how their ideas had formed, and their experiences with using the Codestrates v2+Cauldron development platform. Further, those who worked in groups were asked about their experiences with collaboration in Cauldron. Most interviews lasted between 30 and 75 minutes and were recorded and transcribed verbatim. For the analysis, we employed a two-stage reflective thematic analysis (Braun and Clarke 2006). In the first stage, we performed deductive coding of the interviews, guided by the eight design tensions identified in inquiry 4. The second stage was a combination of deductive coding and inductive coding to identify nuances and subthemes. For this purpose, all co-authors (at that stage) went over every interview together. The purpose of these interviews were not to establish agreement, but to use disagreements and differences as discussion drivers (McDonald, Schoenebeck, and Forte 2019). We mainly focused on issues related to breakdowns, frustrations, and confusions and did not address the things that worked well.

In regard to my overall research question, this inquiry in particular highlighted the intimate relationship between software and computational literacy. This made me realize that research on computational media—and other types of software for development—must take seriously their mediating role in the programming activity. People rely on what they know, and their computational literacies are intimately connected with the software and its interface. Finally, I came to realize how important people's habits and conventions are for not only their skills but also their feelings of empowerment. While the tools matter, they are not enough.

2.3.6 ⑥ A theory of computational self-concept

Based on the findings from, among other, inquiries 2, 3, and 5, I came to realize the importance of understanding *how* and *why* some people develop computational literacy and some do not, as well as the particular processes in which this development happens. In my studies with the nanoscientists and the game design challenge with Codestrates v2, it became clear to me that the feelings of frustration, the breakdowns, and the helplessness experienced were not just attributable to the tools themselves or even the skill sets of the people using them. In fact, these feelings of computational disempowerment were expressed by highly educated and otherwise computationally fluent people such as natural science researchers and experienced software developers. This led me to the insight that supporting computational literacy requires also a focus on the psychosocial conditions of the people involved. Further, as argued by, e.g., G. L. Nelson and Ko (2018), the use of theory from the more general fields of, for instance, learning science and psychology can contribute to a better understanding of these domain-specific psychosocial conditions. Malmi et al. (2019) similarly highlight the importance of using and creating theory to better understand the development of computational skills and literacies.

By looking towards computing education, a field in which these issues are widely investigated, popular theoretical constructs such as *self-efficacy* and *identity* seemed to provide at least part of the answer. My main issue with many of these was that they largely lack explanatory power for understanding the processes leading to the individual development of computational literacy. Some, like the widely referenced construct of *situated learning* (Lave and Wenger 1991), are sociocultural constructs that explain the processes of learning in terms of participation and practices, while others are oriented towards psychological phenomena such as self-efficacy and attitudes. By taking inspiration from Hattie's theoretical framework of self-concept (Hattie 1992), I sought to provide a domain-specific theory of *computational self-concept.* This framework was combined with the concepts presented by Barnett (2009) to provide a more nuanced division between selected subcomponents of the framework.

Although self-concept as a theoretical construct has already been used in computing education research, the concept is, much like identity, a potentially empty signifier in the sense that it means multiple things in different scholarly traditions. Therefore, although I use the same terminology as other theories of self-concept, my aim was to present a version of the construct coming from a particular school of thought. Self-concept, as stated, is difficult to define universally. Like other generalizable theoretical constructs from psychology, self-concept comes out of multiple traditions, each one drawing on a particular body of thought. My purpose was therefore not to provide one unifying theory, but to provide a useful theory to understand the process of computational literacy development. For the same reason, I aimed for the provision of a domain-specific self-concept, the relevance and importance of which is acknowledged in psychology (see, e.g., Shavelson, Marsh, and Byrne 1992; Yeung, McInerney, and Russell-Bowie 2001).

While the development of the theoretical framework did not yield any empirical findings in contrast with the rest of my inquiries, it nonetheless suggested the feasibility of introducing a domain-specific computational self-concept and making it operationalizable. In particular, the importance of confirmation and disconfirmation for the formation of self-images is an important addition for understanding the development of computational literacy. Regarding my overall research question, the inquiry therefore served to better understand the processes that lead to the formation of computational literacy. This can help explain my findings from previous inquiries in which otherwise competent people experience computational crises and disempowerment. More broadly, computational self-concept can illuminate the hindrances to the development of computational literacy. To explore the framework's viability as an explanatory theoretical construct, I conducted a series of interviews (inquiry 7).

2.3.7 ⑦ Student self-concept interviews

As stated, I wanted to examine the practical usefulness of the operationalized framework. Further, I wished to better understand how a particular group of people—humanities students—experienced the process of learning to program. The purpose was therefore two-fold. First, I sought to investigate the usefulness of the framework to guide qualitative research on the formation of computational skills and identities. This is not to be considered a *validation* of the framework, as validation requires a formal and systematic testing of each individual component and their internal coherence and correlation. Rather, I wished to understand its explanatory power, i.e., how well it fares as a number of interlinked, theoretical concepts that highlight particular elements of an individual's process of engagement with computational artifacts. Second, I also aimed to investigate a particular group of students and how they experienced an introductory programming course in a bachelor's degree in the humanities. The goal was thus a phenomenological inquiry into the experiences of the students and how these shaped their computational self-concepts.

For the inquiry, I put out a call for participants among a first semester programming class that I was co-teaching (as teaching assistant) at the time. The degree is an IT bachelor in the humanities and largely equally distributed between genders. To ensure a broad range of representation, we randomly selected ten students from the class list and reached out to these via email. Five students agreed to take part, of which three were self-identified women and two were self-identified men. Participants were between the ages of 20 and 22 at the time of the study. To avoid students feeling put "on the spot" and to foster a more conversational interview style, I decided to do group interviews. Participants were split into two groups, one with two and one with three participants. Each group had both genders represented. As computational self-concept is largely oriented towards experiences and processual development, we conducted two interviews with each group. One in the beginning of the semester and one after the final lecture (but before the summative oral examination). All interviews were conducted by me and a student helper in the same context. The student helper was a former student of the same course, brought in to take notes and prompt participant dialog by sharing her own experiences as conversation starters. To provide additional context for the interviews, each participant was asked to provide information about their background and previous programming experience (before the course). Finally, to examine if there were any links between their experiences as they came up in the interviews and their feelings of computational literacy, each participant was asked three times during the study (before, between, and after the interviews) to self-report their general computer competencies and their programming competencies on a scale from 1 to 100.

			Background	Tendency to ...	
Group	**ID**	**Gender**	**Previous experience**	**Underestimate abilities**	**Overestimate abilities**
A	P1	M	Yes, a little	Slightly agree	Slightly disagree
	P2	F	No	Agree	Disagree
B	P3	M	No	Slightly agree	Slightly agree
	P4	F	Yes, a little	Agree	Disagree
	P5	F	No	-	-

Table 2.3: Students' background information

The semi-structured interviews were centered around themes of breakdowns, feelings of helplessness, and breakthroughs. More information about participants, interview prompts, and methodology can be found in appendix 5. All interviews were subsequently transcribed verbatim. For the purpose of research dissemination, the quotes used in the dissertation have been translated from Danish to English, aiming towards *equivalence* in meaning rather than transliteration (Regmi, Naidoo, and Pilkington 2010). To analyze the data, we conducted a two-cycle data analysis based on Tracy (2013). In the first cycle, the co-interviewer and I coded all interviews individually. This first cycle was largely deductive in nature. This was followed by discussions of the emergent findings as recommended by Tracy (ibid., p. 189). These discussions led us to revise some codes for agreement or disbandment

	Early		Mid		Late	
ID	**Computer**	**Prog.**	**Computer**	**Prog.**	**Computer**	**Prog.**
P1	75	65	72	71	77	76
P2	65	25	75	69	75	65
P3	-	-	-	-	-	-
P4	60	27	69	26	69	30
P5	75	15	50	50	70	50

Table 2.4: Participants' self-reported computer competencies (1–100) and programming competencies (1–100)

of codes where no agreement could be found. To provide a measurement of inter-rater reliability, a third researcher was given the developed codebook and coded two of the four interviews. In comparing both the coded sentences and the specific codes applied, we found a Cohen's unweighted kappa of 0.95, signifying an "almost perfect match" (McHugh 2012). For more details, see appendix 5. In the second cycle, we sought to inductively bring out subthemes that were either prevalent among the participants' expressions or stood out as interesting edge cases. In doing so, we would be sensitive to both general patterns and individual idiosyncrasies.

The student interviews led me to the insight that the interplay between material conditions, skills, and self-images are complex, and that to foster computational literacy, the people involved must also come to see themselves as computationally literate.

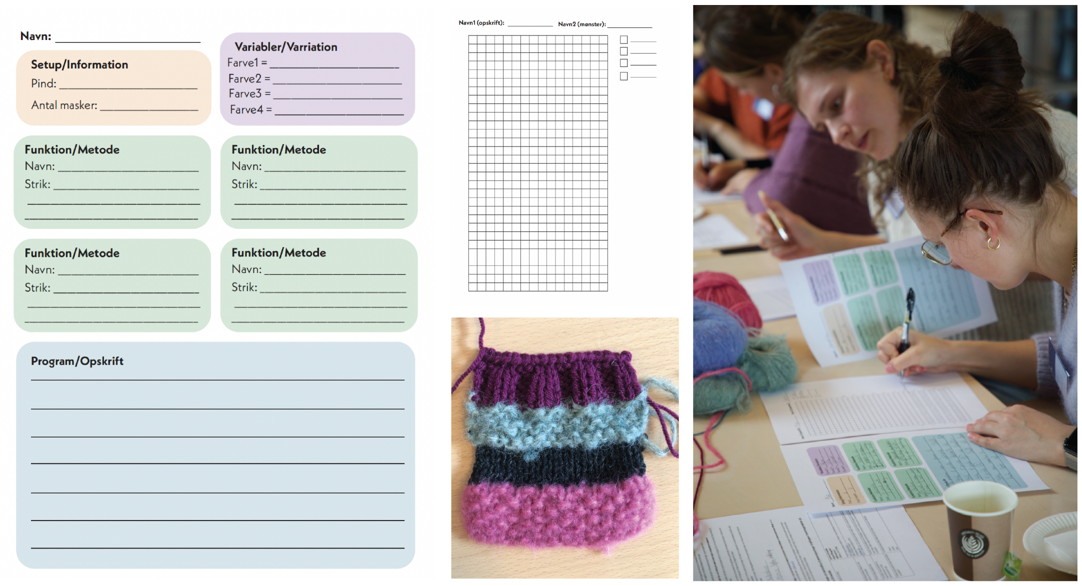
2.3.8 ⑧ KnitxCode workshop

One of my takeaways from inquiry 7 was the complex relationship between materiality, identity, skills, and competencies, and how the development of computational literacy likely depends on all of these factors. This was also made clear in my studies on computational media (i.e., the computational lab book and Codestrates v2). All of my previous empirical work, however, dealt with people for whom computation and programming was already a part of their daily lives. To investigate a different group, a fellow researcher and I designed a workshop in which we recruited knitters without programming experience in order to examine a craftsmanship-based approach to computing that takes advantage of people's existing material intelligence.

The foundation of the study was the perceived similarities between computational thinking and knitting. The study specifically investigated computational thinking, based in particular on the framework from Brennan and Resnick (2012) which deals not just with computational concepts and activities, but also with broader *perspectives*, leaning towards notions of computational literacy. After conducting and redesigning a pilot workshop, we recruited 12 participants for the full workshop (11 female, one non-binary). Participants were between 21 and 46 years old and had 2–25 years of knitting experience ($M = 6$). One participant had limited programming experience while the rest had none. Participants were recruited from a Facebook group for people interested in participating in research. When signing up, participants filled out a questionnaire about knitting and programming experience and their motivation for taking part. Motivations were generally the desire to learn something new and to become acquainted with programming. The workshop lasted around two and a half hours including breaks and consisted of the following activities:

1. Introduction to the workshop
2. Activity: *Decode and recreate*
3. Activity: *KnitxCode*
4. Evaluation

Decode and recreate aimed at letting participants explore the computational nature of knitting recipes. To do this, participants were paired up and given a knitted square. The were then asked to decode the square as a series of programmatic instructions and concepts (see figure 2.3). Afterwards, participants swapped instructions between each other and were tasked with drawing the other person's square in a pattern template, based solely on the instructions. The second activity, *KnitxCode*, aimed to investigate the transfer of skills and competencies to computing. In this activity, we provided participants with a MICRO:BIT and gave them access to pre-made programming templates in the block-based MICRO:BIT editor (see figure 2.4). For the activity, participants were asked to complete four programming exercises, each one building on the concepts and principles from the previous one. Before each activity, the concepts (e.g., loops) were introduced and related to knitting patterns, and participants were shown a video of what the MICRO:BIT should look like at the end of the exercise. To encourage inclusion, a new exercise was only introduced after everyone had completed the previous one. Finally, we conducted a semi-structured group interview with all participants to evaluate the workshop and the participants' experiences with it. Participants further filled out a post-hoc questionnaire about their perception of computational thinking.


Navn: ____________

Setup/Information
Pind: ____________
Antal masker: ____________

Variabler/Varriation
Farve1 = ____________
Farve2 = ____________
Farve3 = ____________
Farve4 = ____________

Funktion/Metode
Navn: ____________
Strik: ____________

Funktion/Metode
Navn: ____________
Strik: ____________

Funktion/Metode
Navn: ____________
Strik: ____________

Funktion/Metode
Navn: ____________
Strik: ____________

Program/Opskrift

Navn1 (opskrift): ____________ Navn2 (mønster): ____________

FIGURE 2.3: Activity: *Decode and recreate*. Template (left), pattern (top middle), knitted square (bottom middle), and participants (right).

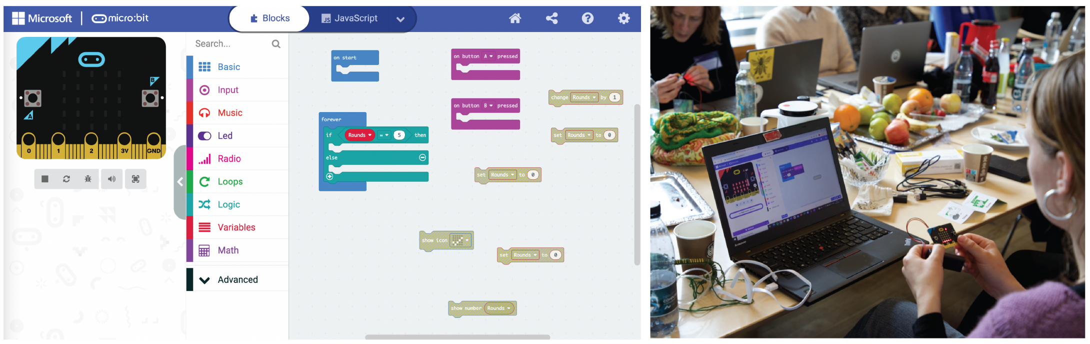


FIGURE 2.4: Activity: *KnitxCode*. Block-based interface (left) and participant (right).

To gather data during the workshop, we mainly relied on participant observations and ongoing note-taking during the workshop. To support our notes, we also took photographs and videos of participants during the activities. Further, to better understand particular situations of interest, we conducted spontaneous, informal interviews when such a situation arose. Finally, all statements made during the evaluation that informed the existing observations and interviews were written down verbatim.

Concerning my overall research question, the workshop showed the importance of material familiarity for the development of computational literacy. As was made clear in my studies on computational media, programmers draw on their repertoire of human-computer interfaces to make sense of an unknown medium. Here, the knitters' development show that this transfer is also possible from non-computing domains. Importantly, it seems as if the knitters were able to not only transfer skills, but also positive self-concepts by drawing on their existing material competencies. The workshop in particular made me realize that the materiality of computing matters more than we might think. This is investigated further in section 5.2. The inquiry also points towards a notion of computational literacy that emphasize craftsmanship and aesthetics which could hopefully also increase gender diversity in computing education.

My research activities thus form a series of ongoing engagements, each of which is an inquiry that draws from previous inquiries and feeds into the following. Moving on, the following chapter is a theoretically grounded exploration of computational literacy and the relationship of the concept with other related theoretical constructs.

Computational literacy and how it is formed 3

Most people find the concept of programming obvious, but the doing impossible.

Perlis (1982)

To be able to relate the mediating qualities of software for programming to any kind of literacy, I use this chapter to unfold the concept of computational literacy and the prerequisites for the formation of such a literacy. This allows me to position the contributions from the dissertation in the historical and current conceptualization of literacy in computing. At the core of literacy is the meeting of three basic human states: knowing, being, and doing. On top of that is the larger questions of *how* and *why* we ought to know, be, and do something. An early proponent of a literacy of computing was Danish computer scientist Peter Naur, who argued that computing (or, more specifically, *datalogy*[1]) ought to be considered part of a general education for everyone (Naur 1967). Returning to my research question for the dissertation, this chapter concerns one part of the relationship between the elements: Computational literacy.

Research question How do the mediating qualities of software for programming contribute to the development of computational literacy?

The concept of literacy is nontrivial. There is a variety of viewpoints of what literacy is or might be, each of which takes into account a different cultural imaginary and ideological backgrounds. The image is further complicated by the modifier *computational.* This chapter will therefore serve two main purposes. First, it provides a background section on computational literacy that culminates in a working definition of the term. Second, based on inquiries 6 and 7, I introduce computational self-concept as a complementary construct for understanding literacy formation.

The first part of the chapter presents various accounts of literacy, ultimately culminating in a working definition for the purpose of this dissertation. In this process, I compare computational literacy to other established concepts such as computational thinking, Bildung, material intelligence, and various new literacies to illustrate how computational literacy designates a particular human-computer relationship. Afterwards, I introduce *computational self-concept* as a way of understanding the processes that lead to identity formation and the development of computational literacy.

3.1 Tracing literacy broadly

Before presenting *computational* literacy, it is necessary to understand the broader concept of *literacy*, as literacy is closely related to sociocultural norms, societal values and governance, and individual competencies. We must therefore have a basic understanding of literacy and its multiple connotations. Etymologically, the word ultimately derives from Latin *littera*, a letter of the alphabet or, in the plural, letters, books, and similar written documents. To be *literatus* meant not just being able to read and write, but also to be learned, scholarly, and cultured. From the very beginning, literacy has been deeply connected to both the concrete activity of reading and writing through a textual medium as well as a broader quality of being learned or educated.

The Age of Enlightenment in Europe brought, among other ideas, the conception of educational ideals about equipping the emerging public with the capacity for engagement in the newly emerging

[1]Computer science was and is still called *datalogi* in Danish universities

nation-states, in opposition to their previous roles as subjects to an absolute monarch. This was not just a matter of enabling a population of citizens to read and write, but to be able to actively take part in society. Literacy is therefore deeply related to educational goals which, particularly in a Central and Northern European context, go beyond educating people to join the workforce. Scandinavian and German educational ideals also focus on Bildung which concerns the individual's *becoming*:

> Learning - as defined in opposition to Bildung - is merely equipping the individual with knowledge and the ability to do something, whereas Bildung is the innate development of one's own capacities (C. Schulte and Budde 2018)

> *(...) it's not public Bildung, because [programming] is not something that the general [public] have to do with (...) but I do think that it could be necessary that people did that, I mean, in line with the general development or whatever you call it* (S2 (post), inq. 7).

As also argued by Dirckinck-Holmfeld, Nielsen, and Webb (1988), literacy is therefore not just about the ability to read and write. Rather, it can be seen as part of a larger movement of literacy as emancipation, that is, Bildung, of the ability to participate on equal foot in a democratic society. C. Schulte and Budde (2018) argue that any serious undertaking of computing education should start from the didactic ideals of Bildung. In doing so, they pave the way for a dual purpose of computing education, of both learning and Bildung. Related to literacy and Bildung is also the notion of *empowerment*:

> (...) obtaining digital technology related skills and competencies as part of basic education is not enough, but in addition to that, agency and empowerment of people to use those skills are also significant (Kinnula et al. 2017)

Literacy is also intimately connected to *competencies*, defined by Rychen and Salganik (2003) as not just knowledge and skills, but also values and attitudes. This model of competence is, however, largely a sociocultural view of human being and doing. In this dissertation I argue that it is necessary to take into account the non-human actors, i.e., the technology, and the embodied and material intelligence that people develop. Literacy is therefore a more suitable term for addressing those competencies that involve a material "other" such as computational media or other types of software.

Having presented literacy in relation to related concepts, I argue for the following distinction:

1. **Competencies** are skills, knowledge, values, and attitudes (ibid.)
2. **Literacy** is a combination of competencies, fluency (Kay 1984), and societal relevance (Vee 2013)
3. **Bildung** is the development of literacy with a particular focus on the realization of an individual's capacities and the development of an informed public (C. Schulte and Budde 2018)
4. **Empowerment** is a combination of literacy and personal agency, reflection, and participation in democratic societies (Kinnula et al. 2017; Iversen, Smith, and Dindler 2018)

This hierarchy is not set in stone, and the difference between concepts is more relative than absolute. The reality is also that people sometimes use the terms interchangeably and with varying consistency. For instance, Kinnula et al. (2017) talk about "literacy skills". Similarly, in areas such as the US without an educational tradition of Bildung, empowerment or literacy is often substituted in meaning. I present a more in-depth definition of literacy in section 3.3.4. Before discussing why I scope my research as literacy, I wish to briefly present various modes of literacy.

3.1.1 Degrees and modes of literacy

Besides the various academic disagreements on the nature of literacy, there seems to be consensus that there are multiple modes, degrees, or levels of literacy. *Formal* literacy is the ability to, e.g., read and write without being able to apply it towards any real-world purpose (Dirckinck-Holmfeld, Nielsen, and Webb 1988). In contrast, *functional* literacy is proficiency in use of tools (Hoffman and Vance 2005). More specifically, we can understand functional literacy in relation to the ideals of Bildung and empowerment. For instance, Dirckinck-Holmfeld, Nielsen, and Webb (1988) call this level of literacy an *adaptive* functional literacy and emphasize how these competencies are used to "support and solidify the existing roles and institutionalized patterns" [my translation] that are part of everyday life. In comparison, *emancipatory* functional literacy is the ability to, inter alia, critically reflect on those roles and patterns (ibid.). In more recent literature, parts of this emancipatory view of literacy can be found in the concept of critical literacy. *Critical* literacy is, essentially, the ability to incorporate a medium or textuality in support of, e.g., critical thinking (Hoffman and Vance 2005) or a critical pedagogy (Tissenbaum, Sheldon, Seop, et al. 2017).

> *I mean, isn't there like a difference between being competent and having a deeper understanding of it? Because I guess you can be a very competent user of technology without having a deeper understanding of what's going on behind the scenes, I think.* (S4 (pre), inq. 7).

These modes of literacy in many ways mirror the five views of empowerment presented by Kinnula et al. (2017). This again illustrates the close coupling between literacy and empowerment. Further, they are a reminder that literacy is one of those terms that easily run the risk of becoming an "empty signifier" (Brown 2016). An empty signifier means, in Laclau's terminology, a signifier without the signified. Concretely, it means that the word *literacy* does not point to an enduring concept of literacy across time and place, but rather is an empty term to be filled depending on the particular context in which it is used. I also wish to remind the reader that other modes of literacy can likely be distinguished, but these concepts serve to frame what is meant by literacy in this dissertation.

3.2 Why computational *literacy*?

In my specific choice of literacy as the focal point (rather than, say, empowerment, competence, thinking, or knowledge) lies a certain conception of the human-computer relationship. Computational literacy is on a general level related to Bildung, competencies, and empowerment. The concept is further related to material intelligence, *computational* thinking, and *computational* empowerment. In this section, I trace how these constructs are related and why the concept of literacy is better suited for my research.

3.2.1 Material intelligence

diSessa has advocated for the use of *material intelligence* as a concept, specifically exploring how to foster the development of said concept through the Boxer environment (diSessa 2001). Material intelligence signifies an "intelligence achieved cooperatively with external materials" (ibid.). There are many similarities between material intelligence and computational literacy, and my understanding of literacy draws heavily from diSessa. However, as pointed out by Vee, there is one crucial difference between these positions: "While people *benefit* from material intelligences, they *need* literacies to negotiate their world" (Vee 2013). As will become apparent in chapter 6, the challenges and opportunities experienced by the students, researchers, and programmers in my research are not external to but an intricate part of their daily lives. Further, this understanding of literacy does not necessitate a common societal literacy; "their world" relates to participants' life-worlds:

> *(...) but I just knew I had to have something with IT, because it's really the future. Then this was just a perfect match, and I thought that this kind of programming, I need that (...)* (S3 (pre), inq. 7).

From material intelligence I have also drawn the importance of mediation. In diSessa's view, the three pillars of material intelligence are the material, the cognitive, and the social. While diSessa comes from an educational computing field, this tripartite division is also mirrored in (parts of) HCI research such as the concept of embodied interaction (Dourish 1999). The central tenet of these positions is that material intelligence (and, by extension, computational literacy) does not reside in either the person or the material but emerges as a property of the relationship between them (diSessa 2001, p. 8). These are the mediating qualities that I refer to in my overall research question which are central in the following chapters.

Finally, in addressing literacy rather than material intelligence, I wish to emphasize the emancipatory and, conversely, hampering aspects of mediation. It is not just a matter of benefit and inconvenience for my research participants but a central part of their life-world, particularly as computational literacy becomes an inseparable part of their studies or work.

3.2.2 Computational thinking

Computational literacy is also not equal to computational thinking. Since Jeanette Wing's call for teaching computational thinking to the broader populace (Wing 2006), the term has been on the agenda in both computing education and in broader educational policymaking in Denmark (e.g., Lund-Larsen 2017) and beyond (Heintz, Mannila, and Farnqvist 2016; Hsu, Irie, and Ching 2019). Computational thinking is, according to Wing, a generalizable way of thinking that stems from computer science and involves abstraction, modularity, et cetera. In this sense, computational thinking is seen as a general problem-solving skill that everyone will benefit from learning.

Different schools of computational thinking can be identified. Traditional computational thinking inherits from the constructionist pedagogy of Papert and focuses on programming practices, while the newer schools consider computational thinking a set of generalizable problem-solving abilities through concept learning (Spangsberg and Brynskov 2017; Denning 2017). The nature of computational thinking is still deeply contested (see, e.g., Tedre and Denning 2016; Denning 2017). I will not unfold the entire research discourse on computational thinking here but merely emphasize that "it was Wing's use of the term, not Papert's, which led to the concept being widely adopted" (Curzon et al. 2019). For a systematic overview of the field, the reader is referred to Li et al. (2020b).

The distinction between computational thinking as programming practice and computational thinking as concept learning (Li et al. 2020a) can be seen as a distinction between competencies and knowledge. The following quote by S1 illustrates just how big this gap can be (similar statements were given by S3, S4, and S5 in the same study):

> *And the hardest part [about the course] is that you don't really know what you– I mean, you know the concepts. I just don't really understand how to use them.* (S1 (pre), inq. 7).

Aho's oft-cited contribution on the nature of computational thinking defines it as "thought processes" involved in formulating problems and solutions (Aho 2011). Computational thinking as a cognitive process was also advocated by Guzdial (2008). However, this version of computational thinking, much like Wing's, strips away the mediating role of the sociomaterial conditions, an integral part of computational literacy. At play is, in essence, an ontological distinction between cognition as an individual, isolated phenomenon and cognition as a distributed, embodied, and material phenomenon that is more than the individual itself (Hollan, Hutchins, and Kirsh 2000; Bjørndahl et al. 2014).

It is my argument throughout this dissertation that we cannot separate thinking from doing. Computational literacy is an embodied and distributed phenomenon that involves thinking with and through media. Early proponents of computational thinking seem to also have realized the importance of the medium (see, e.g., Guzdial 2008; Guzdial 2019) Finally, the relationship between computational thinking and computational literacy is neatly summarized by Li et al. (2020a), who argue that computational literacy is important to every student but not as the new school of computational thinking: "computation is not the special province of computer scientists, and everyone does not need to think like a computer scientist". If my research contributes to a better understanding of computational thinking, then it is in the form of a revival of the practice-oriented and materially grounded school advocated by Papert, diSessa, Kay, and Nelson.

3.2.3 Computational empowerment

Despite its age, Wing's new school of computational thinking is still cited regularly and has come to dominate the discourse (Curzon et al. 2019); however, a number of computing education researchers have pointed out some issues with the terminology. For instance, as argued by Iversen, Smith, and Dindler (2018), computational thinking is closely tied with American STEM education, and the activities encompassed by the term are ill-defined. The authors suggest computational empowerment as a different term that not only includes computational thinking skills, but also broader abilities for reflection and participation in a digitized, democratic society. This position is partly inspired by Brennan and Resnick (2012), who posited that computational thinking is not just problem-solving but rather a broader set of dimensions: computational concepts, computational practices, and computational perspectives. Recently, though, it seems as if computational empowerment is gaining a larger foothold in American computing education research (Tissenbaum, Sheldon, and Abelson 2019)

I agree with computational empowerment as a worthwhile goal, but it is outside the scope of my research. The main reason being my conviction that empowerment is, in essence, *processual* capacity-building and develops from sustained feelings of being literate and competent (through ongoing mediation). This position is supported by Tissenbaum, Sheldon, Seop, et al. (2017), Schneider et al. (2018), and Iversen, Smith, and Dindler (2018). Of course, inquiring into empowerment also depends on an agreed upon definition of empowerment which is far from the case (see, e.g., Kinnula et al. 2017; Abbate 2018)

3.3 What is *computational* about computational literacy?

While the close ties between literacy and Bildung have been acknowledged for many years, a comparatively new development is the proliferation of domain-specific literacies, examples of which are economic literacy (Stigler 1970), design literacy (Christensen et al. 2016), and political literacy (Osler 2000). More specifically, since the spread of digital technologies in both private and public spheres, the so-called "new" literacies have appeared. Exactly what the term new literacies encompasses seems in flux; the *Handbook of research on new literacies*, for instance, largely sees new literacies as those afforded by the Internet and multimedia, e.g., *information literacy* and *media literacy* (Coiro 2008). For a more recent view, we might turn to Knobel and Lankshear (2014), who address two fundamental characteristics of new literacies: The transition from physical to a digital material and the change in ethos towards participation and democratization. However, they still largely see the difference between *old* and *new* literacies as a matter of modality. In their view, the digital medium is a tool through which new textual forms and social configurations might emerge (e.g., fan fiction, social media) that are the target of new literacies. New literacies, as an umbrella term, therefore does not succinctly capture the fundamental change that the computational medium *as* a computational medium brings. Similarly, the literacies touted under names such as computer literacy and digital literacy are not *computational* literacies despite their related names.

3.3.1 Computational literacy is not computer literacy

One perspective on computer literacy can be found in Hoffman and Vance (2005). The authors distinguish between different computer literacies such as functional computer literacy, information literacy, and critical computer literacy. These three levels, respectively, correspond to proficiency with the computer, how to use and understand the computer as a means for communication and information, and, finally, the ability to "incorporate computing technology in support of critical thinking" (ibid.). While these perspectives are admirable, they all represent the computer as a tool and literacy as increasingly sophisticated skills in tool use (Vee 2013). This position is supported in early work by Dirckinck-Holmfeld, Nielsen, and Webb (1988) who argue that computer literacy is, at best, an adaptive functional literacy based on technical skills for using computing systems. diSessa compares this kind of computer literacy to being able to decode "typical words" in place of a full textual literacy (diSessa 2001, p. 5).

3.3.2 Computational literacy is not digital literacy, either

While digital literacy and computational literacy might imply the same phenomenon—after all, computers are digital—I consider them two different things. Digital literacy is the competencies involved with using and creating with digital technologies. The term (in its Danish translation *digital dannelse*) is often conceptualized as being able to use applications for media creation, find information online and evaluate information, and use social media in an informed and responsible way (see, e.g., Bundsgaard 2017). While these absolutely are worthwhile endeavors, they largely treat the computer (or smart device) as a black boxed tool through which young people must learn to navigate, use, create, and reflect.

Tissenbaum, Sheldon, Seop, et al. (2017) similarly define digital literacy as the "ability to share ideas through digital mediums". They draw on an understanding of literacy as the equivalent of reading and writing, much like Pérez-Escoda and Rodríguez-Conde (2015) who argue that digital literacy concerns managing information, communicating through digital technologies, and understanding digital media (albeit digitized versions of traditional mass media), although they do acknowledge the cultural and scientific viewpoints involved in such a definition. All of these perspectives paint visions of digital literacy as, basically, *adaptive* functional literacies enabling people to fill roles and follow existing patterns. A more current perspective can be found in Tuhkala et al. (2019) who tie digital literacy in with notions of empowerment, making, and digital fabrication, arguing that digital literacy is necessary to meaningfully take part in an increasingly digitized world. Still, the digital literacy presented by Tuhkala et al. is less concerned with the computational capabilities *of* the medium than with empowerment *through* the medium.

3.3.3 Computational literacy is not (just) programming

Finally, before presenting a definition of computational literacy, I wish to emphasize that computational literacy does not equate programming. At least not programming as a purely textual practice as presented by Vista (2020). As elaborated further in chapter 5, there is an inherent duality in software in the sense that the textual representation is not the software. While the textual (or graphical) representation of executable code represents the actual code, the mediation happens between the programmer and the code as *potential*. Programming is an activity that may feed into a computational literacy, but it does not do so by necessity. Kay and diSessa focus less on the concrete forms of interaction and more on the characteristics of this mediation. For instance, Kay defined literacy as fluency:

> Literacy means fluency (...) Computer literacy is not even learning to program (...) Computer literacy is a contact with the activity of computing deep enough to make the computational equivalent of reading and writing fluent and enjoyable" (Kay 1984).

This is implicitly echoed by diSessa who does not discuss programming in his argument for computational literacy (diSessa 2001, pp. 1–28). The dangers of conflating literacy and empowerment with "coding" are also examined by Abbate (2018).

Based on these positions, I consider programming to be just one form of human-computer interaction, albeit a very powerful one. What is more important than the interactional form (e.g., programming) is the fluency, the mediating experience, and the emancipatory effects of computing. Finally, other literacies such as data literacy (Carrington 2018) and information literacy (Dirckinck-Holmfeld, Nielsen, and Webb 1988; Bruce 2004) are mainly concerned with computers insofar as they make it possible to work with data and information. That is, computation as a means for something else. One final literacy worthy of mention, however, is Mateas' procedural literacy which builds on the tradition of work by Papert, Kay, and Nelson (Mateas 2005). Procedural literacy is defined in terms of reading and writing processes, aesthetics, and the relationship between "culturally-embedded practices of human meaning-making and technically-mediated processes" (ibid.). This literacy perspective is closely related to what my research has focused on: The mediating qualities of software and how people make sense of these in a reciprocal manner.

3.3.4 Computational literacy *is* ...— A definition

On the basis of the preceding pages, I argue that computational literacy has a particular character that is not equal or reducible to constructs such as programming, computational thinking, digital literacy, and material intelligence, even if there might be (sometimes considerable) overlap between them. I draw heavily from Vee (2013), Mateas (2005), diSessa (2001), and Kay (1984) for the following definition which I have utilized in my work. My position is therefore informed partly from a view of computational literacy as an emancipatory practice towards empowerment and Bildung and partly from a new media perspective that treats digital artifacts as mediating technologies.

Computational literacy ...

- ...is not just competencies, but is related to wider societal considerations
- ...cannot be reduced to a cognitive phenomenon, but is deeply connected to the material conditions in which it is formed
- ...is more than material intelligence in the sense that it foreshadows the influence of computational technologies in all aspects of daily life
- ...is often related to programming (but does not have to be)
- ...has a mediating and reciprocal character
- ...is more than tool use
- ...is experienced as fluency in a medium
- ...is the prerequisite for computational empowerment and Bildung
- ...is a sociocultural phenomenon and is shaped by the cultures and ideologies in which it exists
- ...relies on not just actual, but also perceived competencies and self-images
- ...leads to identity development and demands the availability of a fitting identity
- ...is developed in an ongoing, unpredictable fashion through experiences

That computational literacy has these characteristics does not imply that everyone exhibits or experiences computational literacy in the same way. We can, however, draw out a few general ways that a computationally literate person might act in and understand the world. Most important are the actual competencies, i.e., the skills and knowledge of software and data along with an appraisal of the importance of these competencies. In practice, this can manifest as an understanding of computers beyond the user interface, of something that is qualitatively different from analog materials. For instance, a computationally literate person could write a script to automate manual or repetitive tasks. Ideally, this person also experiences computing and software fluently. This does not mean that they know everything, but they would have the competencies to navigate, learn, and act fluently. If the operating system breaks down, for example, a computationally literate person would be able to reason their way through the errors. And while they do not necessarily know how to or is able to program, they would be able to learn based on their existing competencies and appraisals. Finally, a computationally literate person sees themselves as such. If they encounter an unfamiliar programming language or computing environment, they should be able to draw on an existing belief in their own competencies.

Trying to *a priori* define what a computationally literate person is and does is comparable to asking what makes a great handball player (compared to an average one). They can both play handball, yet the way they do it is vastly different. What is, then, perhaps more interesting than this hypothetical person is to see how (the lack of) computational literacy is realized and experienced by the actual scientists, students, and programmers in my research. This empirically grounded investigation is elaborated in chapter 6. The latter three points in the definition of literacy have not yet been discussed. One of diSessa's arguments is that literacy consists of three pillars: The social, the cognitive, and the material (diSessa 2001, pp. 6–8). The rest of this chapter introduces computational self-concept as a framework for understanding the processes in which these pillars are shaped by and shape each other. This framework is a key contribution of this chapter and in particular provides insights into the way experience shapes the pillars of literacy.

3.4 Self-concept: How computational literacy is formed

This section is primarily informed by my work with students (Fog, Pálfi, et al. 2023). While I found inspiration in existing research on the nature of computational literacy, I was highly motivated by an interest in *how* such a literacy comes to be. Although my research is not specifically about fostering computational literacy in formal education, the field of computing education research is nonetheless a central venue in which research is done on how to support the development of computational literacy and associated skills and competencies. Particularly, personal-cognitive constructs like identity and self-efficacy as well as sociocultural constructs such as sense of belonging and communities of practice provide valuable insights into an understanding of computational literacy. Yet none of these deal explicitly with a model of the concrete experiences that feed into one's image of self.

From psychological research we get the notion of *self-concept* which has not yet seen the same popularity in computing education research as the aforementioned constructs. However, self-concept seems to be able to provide a promising model for the ongoing development of literacy. Even though several scholars have used the construct in computing education research, it is often vaguely defined, conflated with identity, and not living up to its full potential for exploring the wider social and discursive practices of identity development (Große-Bölting et al. 2021).

Self-concept, especially in the form presented by Hattie (1992), aims to provide a holistic model of how a person's self-image is influenced by and constructed from a series of valued statements applied to oneself and the world in an ongoing manner. More specifically, self-concepts consist of descriptions, prescriptions, and expectations of oneself and the world. Self-concepts are influenced by confirmations and disconfirmations of these beliefs.

My use of self-concept as a theoretical model is informed by two main considerations. First, self-concept does not replace or compete with other psychological concepts such as identity or self-efficacy. Rather, the construct can complement those perspectives by focusing on the ongoing development of these. Second, self-concept can be considered domain-specific (Shavelson, Marsh, and Byrne 1992; Yeung, McInerney, and Russell-Bowie 2001). That is, one can have multiple self-concepts that are dependent on the particular context in which one is engaged. As argued by Vee (2013) and Vee (2017), the development of computational literacy not only requires people to build competencies. They must also, in the process, come to find a place for them, a suitable role to inhabit. Self-concept as a psychological theory is, as stated, closely related to identity. In fact, the distinction between these constructs is muddied and still contested (Große-Bölting et al. 2021). For a more in-depth discussion of these terms, the reader is referred to this excellent paper.

3.4.1 The self-concept model

I was very inspired by the model of self-concept laid out by Hattie (1992). Although not directly operationalizable due to the way it is presented as a series of reduction sentences, it nonetheless quite clearly demarcates a few interesting subconstructs. Arguably, my framework is a poor substitute for the richness of Hattie's reduction sentences and the ambiguity and openness of them. The benefit, on the other hand, is a set of usable constructs to frame people's development of literacy: Descriptions, prescriptions, expectations, and confirmations and disconfirmations. Hattie's original reduction sentences are replicated here:

> Our [self-concepts / conceptions of our self] are cognitive appraisals, expressed in terms of [expectations / descriptions / prescriptions], integrated across various dimensions that we attribute to ourselves. The integration is primarily via [self-verification / self-consistency / self-complexity / self-enhancement]. These attributes may be [consistent / inconsistent] depending on the [type / amount] of [confirmation / disconfirmation] our appraisals received from [others / ourselves]. (ibid.)

As argued in section 3.3.4, computational literacy is, among other, related to perceived versus actual competencies and self-images, the development of identity and possible roles, and an ongoing engagement with the world through experiences. These aspects are explored in the subconstructs of computational self-concept as shown in the following sections. First, I wish to emphasize for the reader that self-concepts do not have to be true. They are value assessments and deeply subjective. A person's self-concept does not say whether they are, in fact, computationally literate, but it tells us whether they *feel* literate. That is, self-concepts also include the values and attitudes that people develop towards themselves and the materialities and activities associated with literacy.

Finally, I have been very inspired by Barnett's perspective in which he argues for a pedagogy of higher education that links scholarly disciplines with *being* and *becoming* (Barnett 2009). The purpose of the article is to reintroduce to higher education and their curricula the *being* that ought to complement the development of skills and knowledge in higher education. Barnett's project is emancipatory, emphasizing the relationship between knowing, being, and becoming. To allow students to become their full potential (i.e., the ethos of Bildung), it is, according to Barnett, necessary to develop their dispositions and qualities. Dispositions are those inherent characteristics of a person, fundamental ways of being and navigating in the world such as the will to learn, determination, and preparedness to explore (ibid.). The ways these dispositions actually show in the world are through associated qualities; for instance, the will to learn can show in many ways depending on a person and the circumstances. The qualities are the *character* of the person. I have integrated the concepts of dispositions and qualities in the descriptions component of the self-concept framework, as the relationship between dispositions and qualities are especially relevant for the formation of computational literacy.

For context, I will introduce the central elements of the model before returning to a more in-depth discussion of the opportunities that it brings to my own research as well as computational literacy research in general. An overview of the components and their descriptions can be seen in table 3.1. For a more in-depth description of the individual components, see Fog, Pálfi, et al. (2023). In the following section, I will show the usefulness of the computational self-concept framework by drawing on my empirical findings. The reader is reminded that my investigation of literacy among humanities student was done as part of these students' programming class; while programming does not equal computational literacy, diSessa and Guzdial make good arguments for their close connection (Vee 2017, p. 9).

	Description of the component	Examples: Self	Examples: World
Dispositions (desc.)	The nature of what I am; an innate quality	"I am logical"	"Programming is based on logic"
Qualities (desc.)	How my dispositions are acted out in the world	"I use my logic when approaching new tasks"	
Prescriptions	How the world or I ought to be	"I need good programming skills to impress my friends"	"It is important to know how to program in contemporary society"
Expectations	A future-oriented view of what will happen	"I expect that I will get a good job"	"I expect that AI automation will make programming skills unnecessary"
(Dis-)confirmations	The experiences that inform and change descriptions, prescriptions, and expectations	"I failed the exam"	"My teacher said that my work was good", "My program ran on the first try", "My boyfriend said I'm smart"

Table 3.1: An overview of self-concept components

3.4.1.1 Descriptions

Throughout my empirical research, descriptions of both self and the world turned up repeatedly when people made sense of their material intelligence. These descriptions can point to a variety of domains such as talent, skills, knowledge, and self-images. Self-descriptions can come up as identity, for instance. P2, who never made it through the game design challenge of inquiry 5, said,

> *I mean, yeah, as a non-computer scientist I think I really need very, very simple and basic kind of starting point* (P2, inq. 5).

In Borowski, Fog, et al. (2022), we framed this lack of success as a question of the medium itself and its characteristics. It is, however, likely that P2's self-description also hinders any substantial development of computational literacy, i.e., the cognitive pillar. While it is true that P2 is not a trained computer scientist, she possibly equates this not having the same "gene" for computational literacy, despite the fact that she had some existing programming experience. A similar statement came from a nanoscientist in inquiry 3 who called himself a "non-computer guy" even though he uses scripts as part of his daily work. In contrast, S1 and S2 in the first round of student interviews (inquiry 7) discuss how—after only three programming classes—they are now different from "normal people".

The conversation between S1 and S2 also contributes with a view of how the social pillar of computational literacy comes into play. In "stepping up" from other people, they are now building on the sociocultural conditions of literacy.

> *S2: "We sort of have a language that is for the initiated" S1: "Yes, you feel a little like part of the club"* (S1 and S2 (pre), inq. 7).

Interestingly, though, this "club" is never specified, and later in the same interview S1 discusses how he feels as if he stole someone else's spot in the class, while S2 says that she feels like she's "neither old enough, adult enough, nor good enough".

S2, at least partly, attributes this to her dispositions (i.e., talent): "*There are also those who just can do it, where it just makes sense*" (S2 (pre), inq. 7). Several times throughout all student interviews, the perceived importance of various personal dispositions and qualities come up, such as the necessity of being curious (S2, S4), driven (S1), persistent (S4), logical (S2), and social (all participants). In contrast, S1 many times describes himself as being too colorful and humanistic to be a programmer. This is to some degree echoed by S5 who acknowledges that she is more creative than logical, although she several times points out that she can probably find her "niche" by doing more creative and aesthetic programming projects. These examples of how people describe themselves and the world are beneficial to understand identity development and sense of belonging in relation to computational literacy.

3.4.1.2 Prescriptions

Prescriptions are those kinds of value statements that say something about how oneself or the world ought to be. While these are often presented in the positive, they can also appear in the negative. Importantly, they do not have to be true; they can appear in the form of imaginations. Reminiscing about the beginning of the degree, S1 discusses his (false) prescriptions about how programmers ought to be:

> *(...) but I thought I was too colorful, but then I kind of found out that it's possible to both be colorful and a programmer* (S1 (pre), inq. 7).

This kind of prescriptions about the *nature* of programming and programmers appear aplenty throughout my research. In general, they seem to fall into three categories: How a programmer ought to be (to be a real programmer), what programming ought to be (to be real programming), and the importance of learning programming (for all citizens). Particularly the latter point is reminiscent of the societal aspect of computational literacy. "*(...) a general knowledge that sort of becomes so relevant that you need to have it*" (S3 (pre), inq. 7).

In the game design challenge (inquiry 5), P2 shares her view that programmers should be computer scientists to fully grasp the competencies required. This is mirrored even more dramatically by S1:

> *(...) because you had this sort of idea that you had to be really nerdy to sit and program. But then all of a sudden you remember that you can sort of make it what you want to.* (S1 (pre), inq. 7).

One nanoscientist talks about the role of programming in their work: "*Yeah, we don't code much in this lab. We should, but...*" (N8, inq. 3). To be a good nanoscientist, one ought to be able to program. This is mirrored by another scientist who admits that he does not know how to fix a script, but that he should be able to. Here, the nanoscientists' prescriptions of programming influence their literacy development, ultimately leading to crises (more on crises in chapter 6).

Interestingly, across all participants who can be called *unintentional programmers* (inquiries 3 and 7), everyone seems to agree that programming is important and that everyone (including themselves) ought to learn it to do their jobs better, become better students, or become more educated and empowered citizens. Whether these views are true or not is less important than the fact that it shapes how those people engage with computing in their daily life. Arguably, this focus on values is already inherent in my definition of competencies, but the model emphasizes the close connection between such prescriptions and identity formation, sense of belonging, motivation, and computational literacy.

3.4.1.3 Expectations

Expectations are future-oriented statements that are directed towards oneself or the world. As I argue in the following chapter, the transformational aspects of mediation are at least partly of a temporal character, particularly as my research is oriented towards the conditions for computational literacy. Expectations are therefore a key element in the development of literacy, drawing on values such as motivation, societal relevance, future identity, empowerment, and employment. For instance, in the first round of student interviews, S5 has seemingly already formed expectations of her future computational literacy:

> *Okay, I just need an understanding of how things work (...) Sure, you might understand the advanced stuff, I understand some of it, and then I understand how it's supposed to be for the person who is to use it* (S5 (pre), inq. 7).

S5 distinguishes herself from those who understand the "advanced stuff", likely computer scientists and software engineers. This position is echoed by other students such as S1 and S3 as well. Where does this expectation come from? The identity of the study degree—a mix between humanities and IT—is likely a big contributor. It is very probable that such an identity expectation transforms into a specific computational culture that is, perhaps, unique to the type of degree. It could also be the case that each class develops their own computational culture. The extent and type of this culture is important, yes, but even more important is the way that students' views of computational literacy is colored by the culture in which they are embedded:

> *We are not supposed to sit and program, invent our own code, it's not what we're studying for. We're studying to understand other people's code* (S4 (pre), inq. 7).

A different kind of expectations are presented by another student, S2, in the first interview: "[I'm more like at the] start of competent. I mean, I have a feeling that I will become [competent]". This is a very interesting finding, as it indicates that computational literacy can exist in the potential. It likewise points to a difference between participant demographics as students (inquiry 7), nanoscientists (inquiry 2), and experienced programmers (inquiry 5). While the overall feeling expressed by the students is a trust in the process and a certainty that they will understand eventually, the nanoscientists arguably take another position that learning how to program is considered "a whole other career" rather than an integral part of their job. Finally, experienced programmers can draw on their existing computational literacy. For instance, when asked if anything but time would hinder them from completing their game challenge:

> *Oh no, not at all. We could have finished it, no problem. We'd probably also have redone it and made it properly [*laughs*]* (P11, inq. 5).

We see indications that the different groups are at different stages of computational literacy development. The students are in the process of literacy development, likely drawing on previous experiences that formal education eventually leads to competence. The programmers from the game design study have already developed some literacy, being able to draw on this literacy in their expectations. A third group is the nanoscientists who are outside formal education but never built a substantial literacy. When discussing their challenges, they mostly seem to exhibit resignation. These findings are, of course, slightly generalized and are further elaborated in section 6.3 on computational crises and disempowerment.

These findings indicate two things. First, computational literacy is processual and second, people are able to alleviate a lack of competencies by drawing on future potentials. This is in agreement with research from metacognitive psychology which argues that metacognition (i.e., the ability to reflect on one's learning) is an indicator of academic success (Veenman, Van Hout-Wolters, and Afflerbach 2006). Expectations are seemingly important for the development of computational literacy, since a future-oriented view provides a frame from which people can draw motivation, drive, resilience, and acceptance. In this sense, expectations are also related to self-efficacy, i.e., the belief in one's own abilities.

3.4.2 The missing link: Experiences, confirmations and disconfirmations

While descriptions, prescriptions, and expectations are value-laden appraisals of oneself and the world, they are not set in stone. After all, literacy is not a static phenomenon, but a dynamic quality that is shaped through interactions with the material and social world. As such, experiences in the form of confirmations and disconfirmations are most likely a key factor for the development of computational literacy. These (dis)confirmations can come from a variety of sources such as oneself, the social realm, and the material realm. Ihde's mediation concept of reflexive relations with the world (Ihde 1975, p. 270) likewise argues that learning and formation of self-understanding happens through engagement with a world that talks back, providing a phenomenological argument for the need for confirmations.

One such experience happened to a group (P6 and P8) in the game design challenge who had grand ideas for their game, but then were "faced with the realities that none of us were game developers". Despite this, they were able to remix the provided template code and produce a game, seemingly drawing on their existing literacy as computational scientists. An experience like theirs does not mean much for their self-concepts, and at no time in their respective interviews do they place doubts in their own competencies.

A student, S3, draws experiences from comparing himself to his classmates, albeit not negatively. Rather, by being able to figure things out on his own, preferably before his study group, he gains positive confirmation of his own competencies: "But it's just so awesome to figure it out on your own". Another student, S1, recounted in the second interview a story of how he was talking to some peers with more programming experience and had the realization that, "you sort of felt that you could, that you understood what they said. And then I was, like, 'hmm, maybe I am not that bad after all'". This confirming experience seemingly helps cement the student's sense of belonging and self-image, i.e., his descriptions of self. S5 reminisced about a similar experience from her high school years in which a male classmate asked her to help with a computer issue, and she ended up wondering: "*Am I technical? (...) I was probably also challenged on my perceptions of how competent I really am*" (S5 (post), inq. 7).

Another category of confirmations is the one coming from the human-computer interaction, for instance code running successfully. In the educational setting of the students, several of them speak highly of the automated grading system in the Codestrates[2]-based computational medium used for assignments: "*It's really a confirmation from the application [Codestrates exercises] with those checkmarks. I miss that so much*" (S3 (pre), inq. 7). Conversely, when not receiving these confirmations, this lack of feedback was a challenge to the same students:

> *I'm kind of, like, "I don't have any warning lights in any way, but I can see that the result that I'm getting is not what I want to have". Then now what?* (S5 (pre), inq. 7).

These findings are likely partly attributable to the role of confirmations in (computing) education which is widely established (see, e.g., Marwan et al. 2020). However, they are a reminder that the development of computational literacy depends on people receiving confirmation of their competen-

[2]See section 5.3 on page 62 for an explanation of Codestrates

cies whether from teachers, fellow students, or the medium itself. This latter point relates directly to the mediating qualities of software and is addressed throughout the dissertation.

Disconfirmations can similarly appear under a variety of guises such as failing an assignment, code not running, or being stuck and lost without the competencies to find a solution. P2 in the game design study was already struggling with her computational self-concept and the constant disconfirmations led to further feelings of disempowerment and resignation. Similar feelings of disempowerment were reported by several of the nanoscientists when faced with issues in their scripts. The dynamics of disempowerment and recovery strategies are unfolded further in chapter 6.

As noted by C. Schulte and Budde (2018), interactions with the world provide learning opportunities for an individual that can ultimately transform the person and their views of themselves and the world. While they discuss the role of these interactions in terms of Bildung, this transformative effect of experiences is central to understanding how computational literacy develops. The cognitive, the social, and the material pillars are built from the positive experiences (i.e., confirmations) that the individual has. In the coming chapter on mediation and interaction, I show how this transformation is a core aspect of the mediating qualities of the software for computational literacy.

3.5 Discussion

Being a theoretical construct from cognitive psychology, self-concept cannot in any way explain every single aspect of how computational literacy develops. It is, however, a promising model of the interaction between the cognitive, the social, and the material aspects of computational literacy. In particular, the model's focus on experiences and the associated confirmations and disconfirmations provides an important contribution for understanding people's concrete interactions with the material.

Further, my findings indicate the importance of alignment between the model's various components. For instance, S1 and S2 have—in the first interview—talked extensively about their prescriptions of programmers in the negative. Yet in the second interview, both of them seem to have changed those prejudices:

> *(...) at least question that kind of normal conception of a programmer (...) It's also regular people who need help who are programmers* (S2 (post), inq. 7).

It seems as if S2's self-image came into conflict with her prescriptions of what it means to be a programmer, and she had to resolve this internal conflict. Similarly, another student (S3) seems to have had a potential conflict between his view of himself, his prescriptions, and his future expectations:

> *I actually thought that I would be studying a software engineering degree, because I thought 'it's really cool and super relevant', and then I sort of had this period where I was, like 'you won't complete it'. I mean, I thought that when I was looking at those studying it and those who I know who were going to study it. And we were just very different (...) but I just knew I had to have something with IT, because it's really the future* (S3 (pre), inq. 7).

S3 seemingly resolved this conflict by adjusting the *kind* and *extent* of computational literacy appropriate for him. These internal conflicts were not purposefully addressed in inquiry 7, but the findings indicate the importance of finding a suitable role for oneself (Vee 2013). Interestingly, it seems as if the process of identity formation can also happen in reverse. One student (S1) in the first interview specifically talks about getting recognition for his competencies, and "then all of a sudden you also feel that programming is a little important, so you bring it in with you. And this also ends up defining a little who you are".

The conceptual distinction between descriptions (such as identity) and prescriptions about computing is a useful consideration for computational literacy and its foundations. This is unfolded further in the final chapter. Just as the disconnects between the internal components of self-concept seem to be important to resolve, it also seems to be necessary for the formation of computational literacy that one has (or is able to create) alignment between one's self-concept and one's competencies such as skills and knowledge. For instance one scientist discusses the conflict between his prescriptions and his actual competencies:

> *Yep, I should be able to access it myself. But I don't know how to do it. I have no clue (...) But it looks like a complicated way for a non-computer guy.* (N11, inq. 3).

This is a central contribution of this chapter: Computational literacy relies on competencies, but also demands a suitable self-image in relation to the world (as also hinted by diSessa (2001) and Vee (2017)). In section 6.3, I return to the crises that can appear when people are challenged on this alignment as well as the strategies they use to mitigate them.

3.6 Conclusion

In this chapter, I have provided a definition of computational literacy that builds on previous work from Kay (1984), diSessa (2001), and Vee (2013). By doing so, I illustrate how computational literacy is different from other literacies of the digital realm such as digital literacy and computer literacy. A central contribution from this is the emphasis on placing computational literacy in the larger context of Bildung and computational empowerment.

Another key contribution is the introduction of computational self-concept as a model of literacy development that emphasizes the relationship between self-images, prescriptions and values, and expectations for oneself and others. The model's strength comes from a deliberate focus on confirmations and disconfirmations as the positive and negative experiences that shape people's self-images, ultimately hindering or supporting computational literacy. By introducing the self-concept model into a computational literacy context, I have shown how identity and self-concept is not a tangentially related byproduct of literacy, but a central output of and foundation for literacy development.

Having explored computational literacy as a concept and provided an explanatory model for its cognitive pillar, in the next two chapters I take on the material pillar of computational literacy. First, through an investigation of the mediating qualities of software which is followed by an empirically founded inquiry into the mediation of software for programming.

Mediation and interaction in HCI and beyond 4

In man-machine symbiosis, it is man who must adjust: The machines can't.

Perlis (1982)

Since my research question explicitly tackles the mediating qualities of specific kinds of software, a substantiation of this concept, "mediating qualities", is needed. This chapter will therefore serve as a theoretically grounded account of mediation and how various research traditions fit into this account. My research question is, to reiterate:

Research question How do the mediating qualities of software for programming contribute to the development of computational literacy?

Through my research, I have been interested in a better understanding of how the mediating qualities of software might change the human being who uses it. Early visionaries of new media suggested, for instance, that the media are not interesting by themselves, but that they are interesting insofar as the people using them become something else in the process (Kay 2013b). For this reason, the mediating qualities of software become front and center if we wish to understand how the use of software might lead to computational competencies. In inquiry 2, for instance, it became evident that the nanoscientists struggled to maintain positive self-concepts because of the software that they used. This blame cannot be put solely on the software itself: After all, a different researcher might be able to overcome these issues. The blame cannot, on the other hand, be placed solely with the scientist: It is not just a matter of skills and familiarity of use. Rather, it is in the meeting between software and human, in the mediation, that something is *off*. If we want to understand this "offness", we must look towards the technological mediation.

Mediation and interaction are both diffuse concepts that are used, sometimes interchangeably, to denote various aspects of human-machine relations. In the present chapter, I therefore wish to contrast common conceptions of interaction and mediation, respectively, and posit a working definition of mediating qualities of software for the purpose of this dissertation.

My use of postphenomenology as a research approach is, ultimately, rooted in my interest in the mediating qualities of software for programming. Therefore, I will start the chapter by touching upon the broader implications of approaching my studies using such a lens. There are a few central tenets of postphenomenology that is of interest here. First, stemming from a philosophy of technology, the object of interest—the primary "target" of postphenomenology—is the mediation between human and the world through artifacts. Second, mediation is invisible, immaterial, and mutually enacted and therefore difficult to study empirically. Third, mediation is not neutral. Both parties (human and machine) become something else in the process of mediation. These points mean that the study of mediating qualities is relevant beyond the concrete user-computer interactions in my empirical studies and presents a number of challenges for empirical work. To counter this, the chapter serves as a theoretical unfolding of mediation and interaction and culminates in a working definition of mediating qualities with a set of theoretical concepts with which to grasp these. In this chapter, empirical findings will only be presented as examples. In chapters 5 and 6, my concrete empirical findings are presented, analyzed, and discussed more fully.

It is important to note that the purpose of this chapter is *not* to unfold the entire history and all variations of interaction and mediation as concepts. Rather, I present selected views of interaction and mediation as a stepping stone towards a working definition of how I understand *mediating qualities* in my studies.

I will start by referring back to the pragmatic worldview that I discussed with regard to methodology in chapter 2. Just like any research method might be appropriate insofar as it can help understand a particular question, so the different views of mediation and interaction can supplement rather than contradict each other. This position is reflected in key HCI research on the concept(s) of interaction:

> Whether or not [views of interaction] are true is less important than whether they are useful for understanding phenomena in human use of computing or interactive technology. (Hornbæk and Oulasvirta 2017)

Different research fields bring about particular sets of sensitizing concepts (Bowen 2006) that illuminate certain aspects of the human-computer relationship. This chapter therefore introduces key concepts that can be used to conceptualize and analyze digital artifacts as mediating technologies. Before that, I start by demarcating the concepts of interaction and mediation, as these concepts, while related, denote different perspectives on the human-computer relationship.

4.1 Interaction

A promising starting point is to look at HCI, human-computer interaction, a field whose very name embodies the concept. However, even here, *interaction* is often ill-defined. As laid out by Hornbæk and Oulasvirta (2017), interaction has been understood in at least seven different ways in HCI research: as dialogue, transmission, tool use, optimal behavior, embodiment, experience, and control. The two that come the closest to my approach are interaction as *tool use* and interaction as *embodiment*. Embodiment in particular seems to provide a promising understanding of human-computer interaction as not only being physically embodied but also socially embodied, always embedded in a particular cultural context and denoting a participative quality (Dourish 1999). In this sense, embodiment also addresses meaning-making in a concrete lifeworld.

However, as shown by Hornbæk, Mottelson, et al. (2019) in their overview of 35 years of interaction research at the CHI conference, even though the modalities and characteristics of interaction have increasingly become the topic of HCI research, the dominant interpretations of interaction are still rooted in concrete interactions and centered around “structure, feel, effectiveness, and efficiency” (ibid.). At least some part of this could likely be explained by the role of artifacts in HCI research. They are often characterized as problem-solving solutions evaluated in terms of significance, effectiveness, efficiency, transfer, and confidence (Oulasvirta and Hornbæk 2016). In many cases, HCI research specifically seeks to engage with empirically based theories of human behavior from other fields (e.g., psychology or sociology). In these cases, the purpose of these engagements is largely to *inform* the constructive process of artifact creation or to *evaluate* the constructed artifact on the basis of theory (Beaudouin-Lafon, Bødker, and Mackay 2021).

These conceptualizations of interaction from HCI are therefore not well-suited for understanding how computational competencies might grow out of human-computer interactions. In the field of interaction design, in contrast with HCI, we might find a more explicit focus on the use-oriented qualities beyond the concrete interaction. For instance, coming from the markedly political engagement of Scandinavian participatory design, Löwgren and Stolterman (2007, p. 5) argue that the user-oriented qualities also encompass ethical, aesthetic, political, and ideological aspects. Still, interaction design has as its focus how to design particular interactions, addressing the assumptions and worldviews that designers and participants bring into the process. Broadly, interaction design research therefore aims for knowledge about the designerly process itself (Fallman and Stolterman 2010). The field much less relates itself to the finished artifacts and the way they exist in the world after-the-fact.

In my work on literate computing environments (inquiry 1), I was inspired by the visions of computational media from diSessa and Kay that addressed the mutual changes in both human and artifact that follows from mediation. This media perspective is perhaps most clear in the following quote: “what is most important about a communications medium is ‘what we have to become’ in order to use it fluently.” (Kay 2013b) Engelbert’s conceptual exploration, drawing on Bush’s Memex, addressed this same notion of the transformative and emancipatory quality of the (computational) medium.

In HCI, we can find a conceptual exploration of this transformative aspect by going back to Wartofsky's notion of *tertiary artifacts*. As explained by Bertelsen (2004), tertiary artifacts stem from productive activities but are abstracted away from the concrete practice. They encompass poetic qualities and the transformational capabilities of tool use. Tertiary artifacts are, in brief, visions of alternatives to our existing tools and practices. Their transformational capabilities both concern the concrete tools and the humans using them. In the concept of the *tertiary artifact*, we thus find notions of interaction that are commensurable with mediation theory. Fuchsberger, Murer, and Tscheligi (2013) further argue that an important consideration for research on materiality and interaction is to "understand why certain media or materials led to specific ways of interaction" The use of a mediation framework provides a way to understand this.

4.2 Mediation

Mediation, as a broader and more abstract concept than interaction, is therefore a better frame for understanding the qualities of software for programming in the scope of this dissertation. A working definition of mediation is that designates the quality of the relationship between human and medium. Some branches of HCI do, however, engage with mediation as a concept. Some of the more influential research traditions in HCI that draw on mediation to explore and explain human-computer interaction use activity theory and affordances, respectively, as their theoretical foundations. Activity theory has as its basis that human being-in-the-world is mediated by artifacts (Bertelsen and Wartofsky 1999). However, this mediation perspective only encompasses one aspect of the full range of mediating qualities that I am interested in (see table 4.1). For instance, in Bødker and Bøgh Andersen's work on complex mediation, the mediation paradigm of activity theory is characterized as "material mediation" in which tools are placed between the goal-oriented subject and the object to be worked on (Bødker and Andersen 2005).

Where my work differs from activity theoretical HCI is that this position largely has a focus on work contexts and an implicit view of humans as acting towards a known goal. While the third wave of HCI marks an increasing diversion from a sole focus on work, there still seems to be both a need for—and confusion in—providing a definition of the difference between work and nonwork (Bødker 2006). I do not readily distinguish between work and nonwork in my studies. Further, activity theory has the theoretical underpinning that there exists a structural hierarchy of human doing as activities, actions, and operations (Bertelsen and Bødker 2003). A foundation of activity theory is an ontological separation between subject and object with activities being,

> the purposeful interaction of the subject with the world, a process in which mutual transformations between the poles of "subject-object" are accomplished (Kaptelinin and Nardi 2006, p. 31)

Activity theory, like mediation theory, emphasizes the transformative effects of interaction between subject and object. My main reason for adhering to a postphenomenological approach over an activity theoretical one is that the ontological foundations are different. Activity theory draws from a psychological tradition and emphasizes its close relationship with consciousness (ibid., pp. 36–37). This implies an ontological stability for both subject and object and prescribes the subject as a conscious, intentional actor who acts *through* the technology as tool. In contrast, postphenomenology emphasizes the multistability of artifacts and the co-creation of both human and artifact as products of the concrete relations (Verbeek 2005, p. 217).

While this certainly has its validity, in the scope of my work this division is only a small part of the larger understanding of tool use and interaction. Finally, as implied in their names, a mediation perspective takes mediation as the foundational element, whereas activity theory takes as its basis the activity. The two positions can be commensurable, but their theoretical object of interest is different. However, by drawing from a postphenomenological perspective, the activity theoretical view on humans and artifacts can be integrated within the larger scope of mediation as being-in-the-world. For example, consider the nanoscientists from inquiry 3. Viewing their work through an activity theoretical lens would conceptualize them as being motive-oriented scientists, aiming to perform their scientific work as part of the larger activity of doing research. Their more concrete actions are, among other, writing scripts, managing environments, interpreting textual and image results, and performing lab experiments. As such, the scientists are intentional, conscious actors whose work is grounded in their material surroundings as tools for work. In contrast, a mediation perspective starts at the mediating qualities of the artifacts and how these co-construct both the nanoscientists *as* nanoscientists and the scripts *as* mediators between scientists and their world. They are constituted as scientists precisely because of their use of these technologies. Similarly, their objects of work appear through the mediation. Without the mediation there would be no "object" of work. As previously stated, these two perspectives differ less on their objects of study and more on their ontological focuses. I could have approached my work through an activity theoretical approach, but the mediation perspective provides a framing for understanding other qualities of the human-computer relationship.

Another interesting theoretical perspective on interactions between people and artifacts can be found in the concept of *affordances* which stems from ecological psychology, particularly in the works of Gibson (Kaptelinin and Nardi 2012). In this theoretical perspective, humans are conceptualized as perceptive and embodied actors in-the-world, whose interactions with the environment are based on the ways that the environment "lends itself" to a particular engagement. A mediated action perspective on the basis of affordances is laid forward by Kaptelinin and Nardi (ibid.). The authors move away from a Gibsonian perspective of affordances as possibilities of action in a physical environment towards a view of affordances as "possibilities for human action in cultural environments" (ibid.). Still, both activity theoretical and affordance-oriented approaches in HCI has a series of foundational blind spots in common with other HCI research on interaction:

> All concepts we reviewed commit to some causal role of intentions (e.g., goals)—even the lower-level control and transmission views. However, intentions are taken-for-granted and precede interaction. Even Norman's "gulfs" say nothing about how they change via interaction. An exception is the interaction-as-embodiment view, which speaks about intentions and agency, but which still is silent about how intentions evolve. Thus, HCI, via its concepts, has had an overwhelming tendency to understand interaction as one-sided—as channeling and realization of human intentions through a computer, furthermore assuming that these intentions are outside the realm of interaction itself. (Hornbæk and Oulasvirta 2017)

If we accept the statement, there are a few ways to overcome this apparent limitation in interaction concepts. One is to focus largely on interaction as *embodiment* as a promising frame; another is to expand the notion beyond interaction and towards a broader concept of mediation. I take the latter approach. To supplement the views of interaction in HCI, I include writings from related fields such as the philosophy of technology, STS[1], and software studies, as each of these fields bring particular understandings of the human-computer relationship. By turning to Verbeek and the philosophy of technology, for instance, we can see interaction and meditation as being two distinct phenomena, with the concept of mediation being able to bridge the fields of interaction design, human-computer interaction, and philosophy of technology (Verbeek 2015).

4.2.1 Mediation beyond interaction

Mediation theory is fundamentally directed towards the broader role of technologies and their impact on the human condition. In the context of this dissertation, this scope is too broad. From the philosophy of technology we do, however, get a starting point for how to conceptualize human-technology relations. Verbeek argues that instead of understanding interaction as something that happens *between* a human actor and an artifact, we can consider the interaction the force in which artifacts and humans co-create one another (ibid.). To be more precise, this is particularly so in the type of relationship which Ihde calls the *alterity* relationship (Verbeek 2001, p. 131) in which the world "disappears" from the human-technology relationship. Postphenomenology as a philosophical tradition has as its objects of study the various technological mediations between people and the world. The alterity relation is one in which the world *is* the artifact. It is mainly this human-artifact relationship that I draw on in the present dissertation. However, in some cases, for instance the nanoscientists'

[1]Science, technology, and society

work on their virtual RNA structures, the relationship is instead embodied. When working with representations of RNA structures, the nanoscientists are in a *hermeneutic* relationship with the world. Here, the world and the technology blends together. The unintended shifts between mediation relations are experienced as breakdowns in which the world disappears and the mediating technology becomes present (Madsen 1988). For instance, when the scripts stop working or the medium acts in unexpected ways, the technology itself becomes foregrounded.

Understanding interaction as a manifestation of mediation alleviates the blind spot expressed by Hornbæk and Oulasvirta (2017). This is not to say that there does not exist an object or subject outside a relation, but this distinction brings to the fore how the mediation shapes a particular human and a particular artifact. Kiran (2015) identifies four dimensions of technological mediation that relate to different ways of seeing the mediation: the ontological, epistemological, ethical, and practical dimensions. As we shall see in the subsequent chapter on computational media, for instance, the media and the people using them give form and meaning to each other across all of these dimensions. I wish to stress here that activity theory, especially in the form laid out by, e.g., Engeström (2015), also has a perspective on the dialectical nature between humans and artifacts. My reasons for using a mediation perspective are partly that the terms *interaction* and *activity theory* carry significant connotations to their use. In other words, these concepts are already filled with meaning. Second, a mediation perspective emphasizes the multistability of technologies and how they carry implicit narratives and directionality.

Ihde calls the inherent directionality of artifacts a "technological intentionality" (Ihde 1990, p. 141). The technological intentionality, importantly, is not technologically determining. It is, after all, possible to program an implementation of Doom in Microsoft Excel[2] or subvert the governing capabilities of ICANN's DNS root[3]. Rather, technologies always carry particular inclinations for their use. This intentionality is termed a script in actor-network theory (Latour 1994). Similarly, Suchman uses the concept of plans to mean those prescribed user action list already written into the artifacts (Suchman 1987, p. 44), while Oudshoorn and Pinch (2008) speak of the intentionality of technologies as "configuring the user".

I choose a mediation perspective for the interactive qualities for several reasons. Most importantly, the concept of mediation is capable of addressing the transformative and transcendental aspects of interaction. Second, mediation emphasizes the non-neutral role of technology. And finally, mediation has a temporal character beyond the individual interaction. For the purpose of this dissertation, the central difference between mediation and interaction is that mediation involves a metaphysical element. This element might appear in the form of a *transcendence*. In this view, the mediating qualities that Kay expressed, such as "becoming something else" is the signs of a truly mediating experience, not just an interactive experience. The meeting between human and artifact as *experience* is rooted in a pragmatist outlook and quite elegantly expressed by, e.g., McCarthy and Wright (2004). Pragmatism is, however, not a philosophy of technology but a philosophy of being and experience. For that reason, a pragmatist perspective is not sufficient for my purpose.

4.3 A definition of mediating qualities

To be able to answer my research question, I here wish to present a view of mediating qualities as I have chosen to utilize it. This conceptualization consists of five aspects of the these qualities, drawing from HCI as well as the sociology and philosophy of technology. In short, the qualities related to human-computer mediation fall under the following aspects: Interactional, semiotic, distributional, ethical, and transformative aspects. Table 4.1 shows an overview of the aspects as well as concepts related to them. The different aspects are not mutually exclusive, but they each provide particular perspectives on mediation. This is precisely the point: These perspectives supplement each other to more fully understand a given human-artifact relationship.

While I am inspired by Kiran's four dimensions of technological mediation, these dimensions are too broad for my current purpose. I have therefore created the taxonomy of mediation presented in table 4.1 which synthesizes and brings together perspectives on mediation that I employ in my

[2] https://www.gamedeveloper.com/design/3d-engine-entirely-made-of-ms-excel-formulae-enjoy-this-doom-xls-file- (visited Jun. 8 2023)

[3] https://www.opennic.org/ (visited Jun. 8 2023)

Aspect	Description	Concepts
Interactional	These are the qualities that relate to use and breakdowns. In this aspect, the tool takes on the characteristics of a primary artifact.	Use Embodiment (Dourish 1999) Actions (Kuutti 1995) Primary artifactness (Bertelsen 2004) Breakdown (Madsen 1988)
Semiotic	These qualities go beyond the immediate use of the artifact and are concerned with both how people make sense of artifacts and how artifacts present themselves to humans.	Artifact intentionality (Ihde 1990, p. 141) User configuration (Oudshoorn and Pinch 2008) Script (Latour 1992) Plans (Suchman 1987) Secondary artifactness (Bertelsen 2004)
Distributional	These qualities concern the displacement of responsibility and action in the mediation. Here, the notion of enrollment and delegation from actor-network theory address how mediation can be temporally and physically distributed.	Complex mediation (Bødker and Andersen 2005) Enrollment and delegation (Callon 1984; Latour 1994) More capable peer (Vygotsky and Cole 1978)
Ethical	These qualities concern a broader value appraisal beyond immediate use. The qualities, as in the other aspects, do not address the artifact as such and are not appraisals of that. Rather, it is the mediation itself that carries ethical aspects, being crystallized into notions such as trust, alienation and involvement.	Trust (Kiran and Verbeek 2010) Alienation and involvement (Kiran 2015) Convivial computing (Kato and Shimakage 2020)
Transformative	The transformative qualities are those related to transcendence and becoming something else. In this aspect, mediation can address the development of, e.g., computational empowerment.	Hybrid (Latour 1994) Empowerment (Iversen, Smith, and Dindler 2018) Bildung (C. Schulte and Budde 2018) Literacy (Vee 2013) Tertiary artifactness (Bertelsen 2004)

Table 4.1: The aspects of technological mediation

dissertation. Importantly, the taxonomy is not aiming for a comprehensive overview of all possible interpretations of mediation. For instance, the semiotic perspective would be completely lacking without seminal work such as computer semiotics (e.g., Andersen (1991) and Andersen (2001)). And while I recognize that seminal work in HCI research addresses interaction beyond the concrete use (e.g., Bertelsen and Wartofsky (1999) and Bertelsen (2004)), I shy away from using interaction as a guiding concept because of its prevalent history in HCI research and its associated connotations. By providing this taxonomy, I draw together concepts from different research traditions that are able to supplement each other by relating to different *aspects* of mediation. The concepts presented in table 4.1 are selected concepts that embody the aspect, most of which I employ in my work. In particular, mediation as a concept is able to capture the transformative and metaphysical aspect of the human-computer relationship and can explain and expand upon the concept of computational literacy.

4.3.1 Interactional aspect

In this perspective, we can view the human-artifact relationship in terms of tool use. For instance, in the case of the nanoscientists (inquiry 3), we see how their engagement with their digital tools to a large degree can be seen as a use-breakdown-adapt loop. In one interview, a participant explained how they would find unsupported software online to do their work. When the software seemingly broke down (likely related to the scientist's lack of computational competencies), their existing workflow was disrupted. They would then go on to either get help from a more capable peer or find new tools to replace the old ones.

4.3.2 Semiotic aspect

The semiotic aspect addresses the ways that the artifacts are more than just tools for use. Rather, they embody particular inscriptions or plans that casts the user into a certain role. When interviewing students (inquiry 7) it became apparent that the switch from embedded web tools to WebStorm (an IDE) was much more than just a change in tools. The students were cast into new roles as "real programmers" rather than "experimenting with programming".

4.3.3 Distributional aspect

From the distributional perspective, mediation can be conceptualized in terms of the quality and degree of enrollment and delegation. I specifically use these terms from actor-network theory as they do not readily distinguish between human and non-human actors. This is not a case for the symmetrical ontology of actor-network theory as such, but the distributional aspects are relevant in terms of both artifacts and people. For instance, the computational lab book (inquiry 3) introduced the possibility of delegating code execution to a central server, thus enrolling another non-human actor. Interestingly, this actor is largely invisible to the scientists and takes on a different character altogether. Meanwhile, the self-hosted characteristics of the lab book accessible by a simple URL and the collaborative nature of the medium allowed the scientists to easily enrol a more capable peer.

4.3.4 Ethical aspect

The ethical aspect is concerned with qualities such as trust, conviviality, and estrangement. In an interview, a participant in the game design challenge expressed how the use of Codestrates v2 proved difficult for him:

> *I think I would have been like, this is too much for me, I just go back to my old editor and ... being sure things work the way I expect them to work* (P7, inq. 5).

Part of this can probably be explained in terms of tool use and familiarity, but a key point is that the breakdown in the interaction also carries ethical implications regarding trust in the software. In fact, P7 was never able to finish the game design challenge.

4.3.5 Transformative aspect

This final aspect addresses the ways that technological mediation can bring about "something new". This "something new" is difficult to define for all possible cases (owing in part to the multistability of technologies), but concepts such as hybrid, reflexivity, and literacy can be used to understand these qualities. In an interview with a programming student, she addressed how one particular engagement changed their self-image. The student explained how, during a web development module, she had an "eye-opening experience" of both the inner workings of webpages and her own abilities as a programmer. The student-IDE-browser hybrid thus transformed the student into a more computationally literate person.

It seems clear that a mediation perspective can meaningfully be used to investigate how software for programming may lead to computational literacy. In the schematic that I have put forth (table 4.1), the different aspects draw together a series of concepts that can supplement each other rather than compete. For instance, when observing knitters learning how to program during a workshop, the human-artifact relationships can simultaneously be viewed in terms of, e.g., use and breakdowns, in situ meaning-making, trust, and empowerment. An interaction perspective or an activity theoretical approach might in principle be able to do the same, but both approaches run the risk of drawing upon established traditions of meaning associated with the concepts involved. Therefore, I have employed a mediation perspective that is able to integrate with other approaches.

Part of the significance of this chapter is the operationalization of a theory of mediation. A post-phenomenological perspective is largely *philosophical*, but the schematic outlined in table 4.1 is an attempt to bring the philosophy closer to the empirically based traditions such as HCI. This should be a boon to all who wish to understand human-computer relationships in both HCI and beyond. In the following chapters, I will employ this mediation perspective to unfold my research question and investigate, based on empirical studies, how the mediating qualities of software for programming can be understood in terms of computational competencies, empowerment, and literacy.

Software and computational media 5

So many good ideas are never heard from again once they embark in a voyage on the semantic gulf.

Perlis (1982)

Having presented a vision of computational literacy and a model of technological mediation in the preceding chapters, the following chapter provides an investigation of software for programming as part of my research question:

Research question How do the mediating qualities of software for programming contribute to the development of computational literacy?

First, to provide a common ground and clarify concepts, I address the topic of what software is. Afterwards, to provide a perspective on *software for programming*, I give a historical overview of programming modalities and, more specifically, the visions of computational media. Computational media are touted as a particularly fruitful software paradigm for the development of computational literacy (diSessa 2001). A large part of my investigation concerns the computational media environments used in my own research. As the early iterations of our *-STRATES family were based on the literate computing paradigm, this programming modality is given special attention. Finally, I provide a set of empirically grounded findings about computational media and their mediating qualities in action.

Any type of inquiry into the computational material such as the present dissertation requires an elaboration of concepts pertaining to said material. As my research question tackles the mediation between material conditions and computational literacy, it is important to distinguish exactly what both of these terms connote. In this chapter, I therefore set out to present a vision of software for programming, actual and potential. This includes a review of computational media as a particular software paradigm and an in-depth discussion of literate computing environments.

5.1 A few conceptual clarifications

To start off, I wish to clarify some concepts relating to both the materiality and activity of programming, as the vernacular language often uses similar, but different words interchangeably. These words further have culturally contextual meanings.

Regarding the aspect of materiality, there are at least two concepts that need clarification: code and software. The act of creating these digital artifacts is similarly often conceptualized as either coding or programming. This section is not intended to provide a full overview of all possible permutations of meaning, but serves to lay down working definitions for the purpose of the dissertation. There are two reasons for this. First, the materiality of the digital is in itself a difficult entity as evidenced in multiple fields such as interaction design (Löwgren and Stolterman 2007), software studies (Berry 2011; Dourish 2022), HCI (Fuchsberger, Murer, and Tscheligi 2013), and literacy studies (Vee 2013). All the more reason to try and pinpoint a conceptualization of it. Second, to actually investigate the material conditions of computational literacy, it is important to understand what kinds of activities might be involved in such a literacy. Here, I also wish to point out to the reader that I prefer to use the term *user-programmer* when discussing software mediation rather than *user*:

> the very idea of "the user" reconfigures a multifaceted human being as an adjunct to a piece of hardware or software (Satchell and Dourish 2009).

And while there are other established terms such as end-user developers (Fischer and Giaccardi 2006; Borowski and Larsen-Ledet 2021), this discourse implicitly creates and sustains a particular relationship between people and artifacts. For instance, in the case of computational media, the output can be other computational media. So where is the end of this chain of transformation? For lack of a better word, I use the term user-programmer to signify the full human being in its emancipatory capabilities.

5.1.1 The act of doing

One of these conceptual confusions relates to the act of writing executable code. In the popular vernacular, particularly in a US context, the verb *to code* is often used. For instance, the popular platform Codecademy uses the subtitle "learn to code"[1]. The case is similar across a variety of other NGOs that advocate for and teach technical skills outside formal educational settings, e.g., freeCodeCamp.org[2] and Code.org[3]. Often, this is coupled with advocacy of computer science, creating a narrative in which computer science is the theory, and coding is the practice. Another school of thought categorizes the activity as programming, often lamenting the connotations of coding as a process of simple translation (Abbate 2018). Finally, the broader notion of computing is often used, especially in educational contexts.

For the purpose of my current work, I put forth the following definitions:

Coding Coding can be considered the act of applying codes to a given material. In the context of code authoring software, such a case might be the creation of a website by writing HTML. However, for the sake of simplicity and due to its popularity in vernacular English, I consider its usage equivalent to programming.

Programming Programming is the act of creating software using either physical or symbolic manipulation of a digital material. The type of artifact that programming produces can be one of many as will be unfolded below. One example of programming is software development.

Computing Computing is a broader term that not only encompasses concrete interactions like coding and programming but also auxiliary activities such as setting up a development environment and using versioning tools. As argued by Galey and Ruecker (2010) who in turn draw on McCarty, computing further signifies intellectual processes beyond the single artifact.

Other terms like *development* certainly also have their merits. My reason for using *programming* over *development* is mainly that the notion of (software) development is loaded with regard to the type of artifacts produced, namely interactive software. Programming can equally well describe the act of creating a one-off script or doing data science as writing a full-blown application. Further, while Vee argues for the terminology "coding literacy" (Vee 2013; Vee 2017), I find the use of coding as a term to potentially reduce computing and programming to a purely textual activity.

These activities all involve a direct interaction with the machine. There are other activities of a different character, e.g., computational thinking, that are related to these, but ultimately designate a different kind of (non-mediating) activity. The connection between these activities and computational literacy can be found in chapter 3. For now, the definitions serve as a background framing of the rest of this chapter which will investigate and elaborate on programming and software.

5.1.2 The artifact produced

The second conceptual confusion arises from the material produced through the constructive act of programming (or coding, depending on one's vocabulary). What is, indeed, the major difference(s)

[1]https://www.codecademy.com/ (visited Jun. 8 2023)
[2]"Learn to code"; https://www.freecodecamp.org/ (visited Jun. 8 2023)
[3]https://code.org/ (visited Jun. 8 2023)

between code, program, and software? Software is typically defined in contrast to the physical hardware. A humorous adage is that the hardware is what you kick, while the software is what you can only yell at. While likely hitting close to home for many people who have been frustrated with the digital technologies in their lives, it also points at a fundamental difficulty with grasping software as a concept: its materiality. Influential interaction design researchers have, for instance, called the digital the "material without qualities" (Löwgren and Stolterman 2007). However, despite the fact that the physical materiality of software is literally just seemingly arbitrary sequences of 0s and 1s encoded as electrical signals, there are structural, social, and cultural forces that shape what can and cannot be done with software. In turn, the mediating qualities of software are not completely arbitrary, but rather deeply embedded in existing communities, materialities, and practices.

For the purpose of the dissertation I put forth a few working definitions of software, application, program, and code. Just like coding and programming, these are terms that are often used more or less synonymously. Kay terms software the "materials of computing" (Kay 1984). Berry (2011), on the other hand, distinguishes between *ideal types* of code, two of which he calls "delegated code" and "prescriptive code". Delegated code is (textual) human-readable source code while prescriptive code is software, that is, machine-readable streams of 0s and 1s. The process of transformation from source code to software happens through translation, for instance via a compiler, an intermediate software artifact.

Software Software is processual. Software is not code, but the result of translating source code into machine-readable streams of binary. For the sake of simplicity in the scope of this dissertation, any binary representation in a digital system that is not data is software. Although Dourish, referencing von Neumann, argues that data and programs are essentially the same (Dourish 2022, p. 22), this does not provide for useable definitions in the present context. Importantly, software is not by necessity the outcome of programming, but can be both an output and an intermediate step towards it.

Application An application is a specific type of software; a "monolithic, nonmodifiable" (diSessa 2001) entity that is often tied to a particular work domain and conceptually separate from the data on which it operates. It is also the dominant mode of contemporary human-computer interaction (Nouwens and Klokmose 2018).

Program The word is often used synonymously with software and application. However, the word itself connotes regularity, order and sequence as evident in the use to mean, for instance, a political program or a schedule. It is also often used colloquially to mean application. For these reasons I choose to use the more general and computing-specific *software*.

Code Code is the basis for software. Through the act of programming, a programmer writes code (often textual) that is translated into software. Code is the textual or visual representation that requires a level of proficiency and literacy to manipulate (Vee 2017).

For the purpose of the dissertation I also want to distinguish between three *ideal types* of purposes for programming, as not all programming activities happen for the same reasons. The first of these I call programming for *system development*. The intended outcome of this type of interaction with the machine is to develop a system that can be executed. When software engineers work to create an application, the back-end of a social network, or embedded systems, this is the type of programming that they undertake. Another ideal type is programming for *computation* whose main purpose is to perform computations on an input to provide an output in the form of data. Data scientists are a group of programmers whose work centers on computing data, as are, for instance, statisticians working in R or Stata. Finally, programming for *interaction* seeks to provide interactive qualities to a system. The most common contemporary example of this can be found in front-end programming of websites, but historical systems like HyperCard similarly targeted this form of programming. It is important to note here that these are ideal types, and that real-world programming activities might have some overlap.

5.2 Programming and mediation

With the terminology settled for now, I want to first present a historical overview and material-interactional analysis of selected human-computer interactions for programming. This is a novel contribution that was not covered in any of my previous work. In the spirit of Petrick (2020) who merges media and interaction perspectives in analyses of various interfaces, the present contribution serves to show that interfaces do not tell the full story by themselves. The history of human-computer interaction is also the history of mediation, of ethics, distributions, translations, and transformations. Understanding the various historical configurations allows us to better reflect upon dominant software paradigms and alternative visions. It further cements the importance of the material conditions for the development of literacy, mastery, and identity among those who engage with digital technologies.

While the early days of programming consisted of manipulating physical switches and wires, along the way additional layers of mediation were introduced. By tracing the development of interaction for programming, I first and foremost seek to address the contingency of today's programming software. Popular, contemporary IDEs such as Visual Studio Code or WebStorm are not a necessary end goal of any continuous evolution and improvement of programming modalities. Rather, they represent a stabilization of a particular variation of technologies (Pinch and Bijker 1984), themselves products of the idiosyncrasies of those who invented them in the first place. By presenting other ways of interacting with the machine to create software, I aim to illustrate that things could be different and that varying types of mediation and delegation take place in these multiple configurations.

The earliest computers were imaginary: The principles behind the Babbage analytical engine did not come into physical existence until almost a century after their conception (Vee 2013). Similarly, the first formal program for this analytical engine was a physical representation, drawn using pen, ruler and paper by Ada Lovelace. This first conception of a computer was essentially a mechanical contraption, operated through the physical manipulation of levers and switches. Although fascinating, the broader history of computing as a whole is not covered in this dissertation (see, e.g., Fuegi and Francis 2003; Campbell-Kelly et al. 2018, for overviews). A later mode of operation was in the form of punch cards: Cardboard pieces containing literal holes in which current could travel to form the right connections for the computer to do its calculations. Once the graphical interface became a mode of interaction with the machine, ways of programming also changed.

In this section I present seven different human-machine configurations in the context of computer programming: ENIAC, punch cards, teleprinter/mainframe, programming languages and compilers, operating systems, Emacs and IDEs. Despite the seemingly evolutionary and chronological order, the list is not to be seen as subsequent developments that naturally build on top of previous ones. These various configurations have existed—and still exist—in complex, mediated layers entangled with other configurations. Infrastructural just as much as interactional configurations, these configurations build on each other and often co-exist in networks of dependency (Star 1999; Straube 2016). Nor is the purpose here to lay out an exhaustive list; multiple other ways to program have existed and still do. The main point is rather to make a technographic analysis of significant programming modalities and explore their interactive and mediating qualities. Each of the following configurations is considered a network of artifacts and human actors between the person that conceptualizes the program and the machine carrying out said program.

5.2.1 ENIAC

The ENIAC is presented here as a representative of the early electronic computers that were programmed through the physical manipulation of wires and plugboards. Programming this machine meant to literally manipulate the hardware by moving wires around. In this section, it is necessary to clarify a few concepts. The *programmers* of the ENIAC are the operators, the people physically programming the machine by moving wires and flicking switches. The person writing the programs to be executed is, in this terminology, the *engineer*. These are fluid definitions, and various sources use the term *programmer* to mean either role.

The act of programming was highly embodied. To program the ENIAC required weeks of manual work on behalf of the programmer. In turn, the programmers were deeply familiar with the inner workings of the machine, often being able to pinpoint the exact piece of faulty hardware responsi-

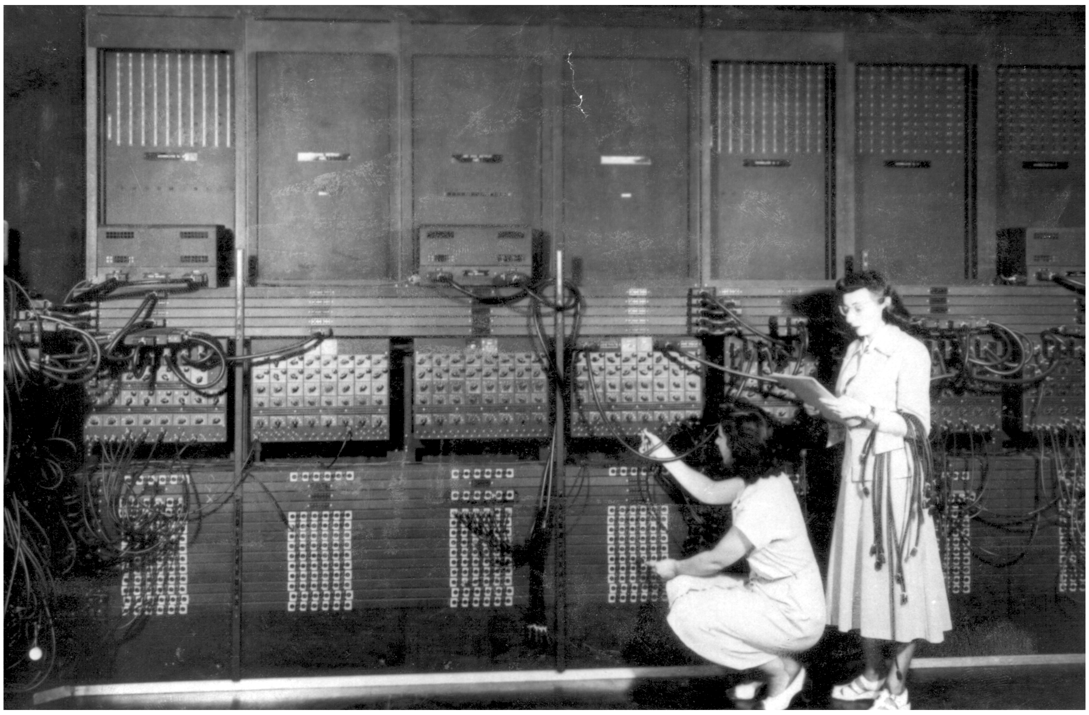


Figure 5.1: *Ruth Teitelbaum and Marlyn Meltzer wiring the ENIAC with a new program in 1946.* Image source: "Two women wiring the right side of the ENIAC" by Unidentified U.S. Army photographer. Retrieved from https://commons.wikimedia.org/wiki/File:Reprogramming_ENIAC.png. License: Creative Commons Public Domain

ble for errors (Fritz 1996). The embodied interaction afforded a different mode of engagement than what our software-dominated contemporary programming modality affords (Streeck, Goodwin, and LeBaron 2011). At the same time, the ENIAC and the complex, embodied forms of interaction that it afforded was also the cause of delegation of work. The vision of Babbage's analytical engine already set out to delegate processes of computation through the enrollment of a machine to perform complicated computations. In the case of the ENIAC, the small, highly specialized corps of programmers represents another degree of mediated action, as the human programmers came to be an obligatory passage point (Callon 1984) for interaction with the machine.

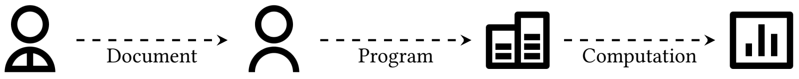


Figure 5.2: The engineer creates a symbolic representation of the program which is handed over to the programmer. The programmer engages with the machine in an embodied manner which leads to the machine computing results.

Programming the ENIAC thus depended on the successful enrollment of both human and non-human actors, creating a programmer-machine hybrid configuration. Despite the close relationship between the programmers and the machine due to the embodied nature of the interaction, there was considerable distance between the creator of the symbolic program and the machine carrying it out. The engineers would create symbolic representations of the program which would then be delegated to the programmers, relying on their embodied relationship to carry out the calculations through translational processes. Finally, programming the ENIAC had the explicit goal of programming for computation to arrive at a set of finished results such as ballistic tables. In the days of the ENIAC, there was no inherent separation between the two categories of *user* and *programmer*. In fact, one could not make use of the machine without programming it in some way. However, as demonstrated by, e.g., Light (1999), the (women) programmers were largely removed from computing history, in turn also obscuring the historical importance of the embodied nature of programming the ENIAC. The history of the ENIAC is, then, largely the history of the hardware, not of its interactional qualities. This material history of the ENIAC therefore does not tell the full story about the complex layers of mediation and delegated expertise. A final note on the ENIAC's materiality: As a physical phenomenon, it was literally larger than life, confined to a large room and representing an enormous sum of money and knowledge. Interacting with the machine implied a special ontological and epistemological status of the people who knew how to program it (Fritz 1996). This in turn inscribed the user-programmer as a developer-with-privileged-knowledge-and-status.

5.2.2 Punch cards

While the concept of punch cards already existed in manufacturing (most famously in the case of the Jacquard loom) and census counting, its use in computer programming lasted until well after the introduction of the first programming language. Programming via punch cards requires additional enrollment, translation, and delegation in comparison to the ENIAC: A programmer wrote the program in a symbolic form, for instance mathematical formulas or a list of instructions, not immediately parsable by the machine. To construct machine instructions, the instructions were passed along to the coder who would translate the symbolic into the physical by literally punching out the holes in the cards. Finally, the machine operator fed the cards into the machine. The exact nature of the symbolic instructions, whether text or formulas, is less important for the current example than the fact that some form of translational process was necessary to turn them into cardboard-based machine instructions.

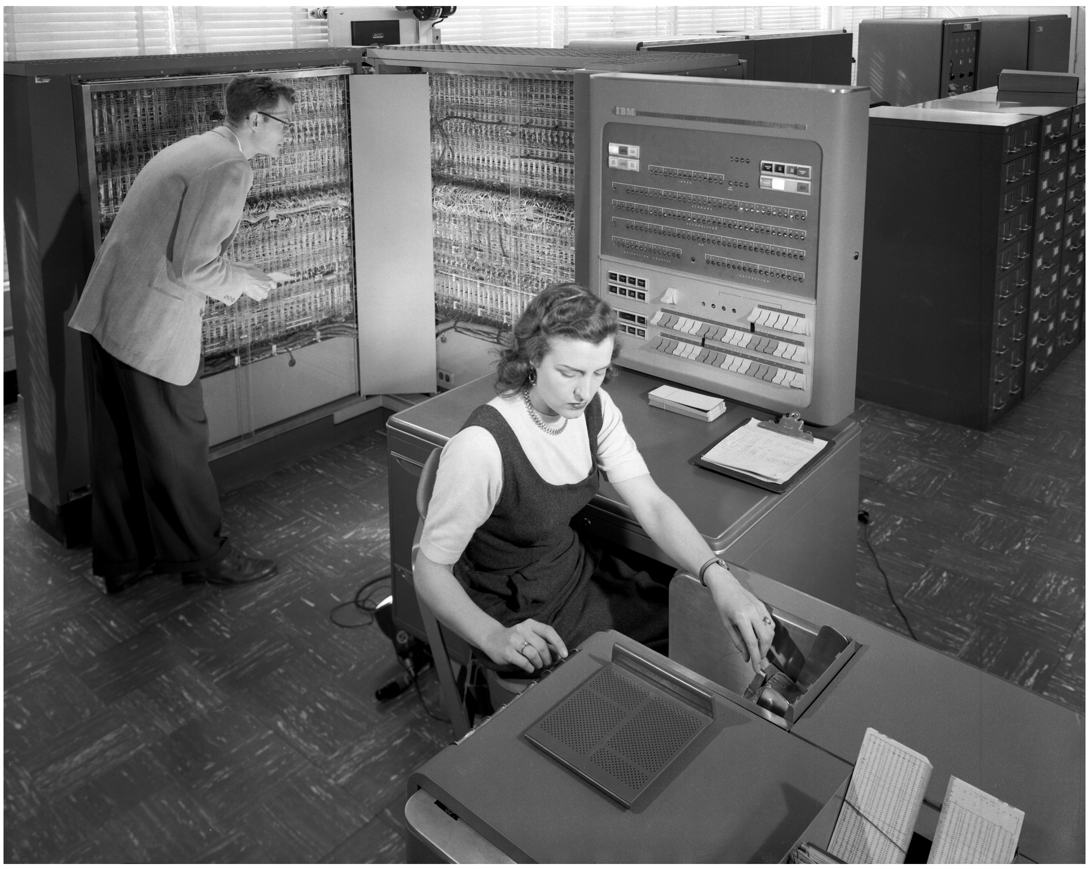


Figure 5.3: *Woman operating an IBM 711 card reader on an IBM 704 computer in 1957.* Image source: "IBM 704 computer at NACA" by NASA. Retreived from https://commons.wikimedia.org/wiki/File:IBM_Electronic_Data_Processing_Machine_-_GPN-2000-001881.jpg. License: Creative Commons Public Domain

Punch cards represent a different materiality than the ENIAC, one that has even become part of the cultural lexicon among certain social groups (Lubar 1992). In a way, their materiality is simultaneously more and less abstract than the ENIAC. On the one hand, the actual instructions (i.e., the program) are stored as executable code in a physical format. They have a form, and a program can be stored indefinitely, at least in theory. On the other hand, they represent an abstraction away from both machine and program. Punch cards, literally cards with punched holes, are semantically meaningless outside their context. It is only in their mediating roles in the network of people and artifacts that they have any meaning.

The interaction that takes place is therefore a complex entanglement of humans, artifacts, practices,

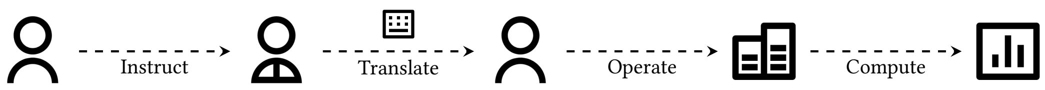


FIGURE 5.4: The programmer writes the program which is then translated by the coder into physical cards. These cards are fed to the card reader by the machine operator at which point computations are performed.

and conventions *in situ.* In fact, that is not different from most other programming modalities, but in contemporary programming practices the roles and responsibilities of people and artifacts are increasingly subsumed into the material conditions. It is also not unique to programming activities—Suchman's seminal work on photocopiers as bearers of situated plans examines just this inscription of interaction into the artifacts (Suchman 1987).

5.2.3 Teleprinters and mainframes

The teleprinter represents a mode of programming that still exists in contemporary life, even if it mostly looms in the background for most people. Modern operating systems implement the teleprinter as a possible way of interacting with the machine—one big difference between early text-based interactions and the current operating system terminals is that this mode is often optional today.

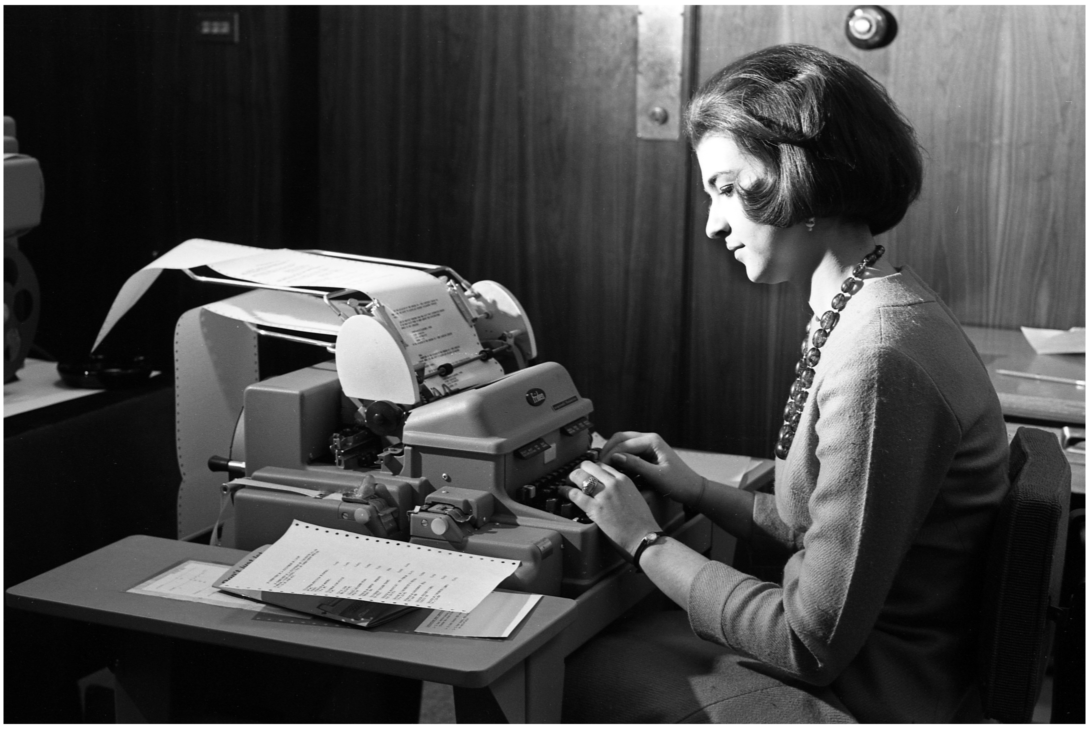


FIGURE 5.5: Woman interacting with a FRIDEN FLEXOWRITER in 1966. IMAGE SOURCE: "Flexowriter" by Yves Tessier. Retreived from https://commons.wikimedia.org/wiki/File:Flexowriter_192-1-004.jpg. License: CC BY-SA 4.0

In contrast with the contemporary operating system terminal, the interactional qualities of the teleprinter were characterized by both temporal and spatial shifts. Whereas the ENIAC and punch card-operated machines required co-location (or at least the enrollment of and delegation to a co-located human actor), the teleprinter allows for operation at a distance. Through the incorporation of existing infrastructural technologies such as telephone lines, a computer programmer could provide data and programs to a mainframe located in a different place. The interaction with a teleprinter can be considered a form of conversation: A symbolic rather than physical manipulation of a system in a call-and-response interaction modality. Further, the interaction is characterized by the teleprinter not being the ultimate target of interaction itself. The teleprinter becomes a medium through which one operates in order to effect a remote mainframe. The teleprinter is translational and mediating, transforming the program and communicating with the mainframe on behalf of the human operator.

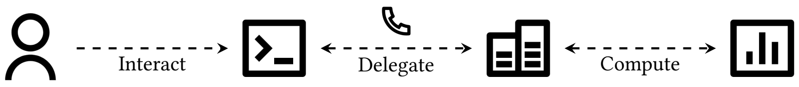
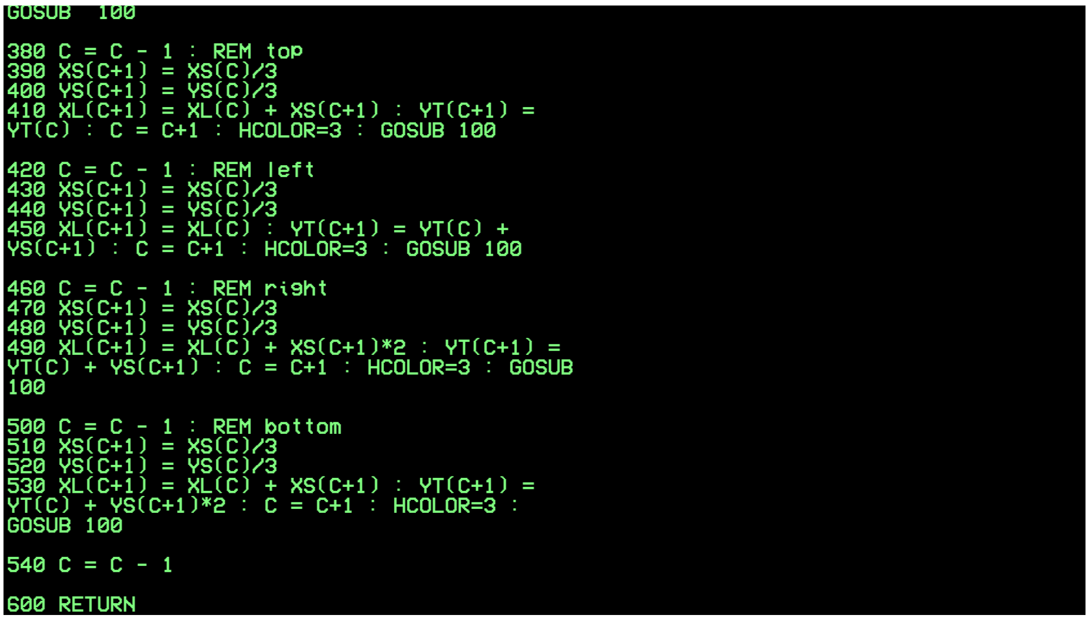


Figure 5.6: The programmer interacts symbolically with the machine which distributes and delegates computing at a distance. The mainframe is the locus of computation, black boxed and hidden from the programming practice, and returns only the finished computations.

Finally, the teleprinter and related technologies mark a change in the materiality of programs. Early computer interaction happened on the terms of the machine, plugging electrical wires or punching out holes through which electrical current could run. Interacting with a teleprinter involves communication in a human-friendly medium, text, but in the mediating process the text is transformed into a truly symbolic and ephemeral form rather than a material one. There is no program outside the text itself. This transition to semantically meaningful textual interaction took place already with the development of (high-level) programming languages.

5.2.4 Programming languages and compilers

Whereas the ENIAC, punch cards, and the teleprinter represent physical, in-the-world technologies, the software for programming is equally important. With the invention of the stored-program computers, programming took a different character: Rather than having to manually input the program every time, programs could be stored in electronic form in the machine in an ephemeral and intangible form.



```
GOSUB  100

380 C = C - 1 : REM top
390 XS(C+1) = XS(C)/3
400 YS(C+1) = YS(C)/3
410 XL(C+1) = XL(C) + XS(C+1) : YT(C+1) =
YT(C) : C = C+1 : HCOLOR=3 : GOSUB 100

420 C = C - 1 : REM left
430 XS(C+1) = XS(C)/3
440 YS(C+1) = YS(C)/3
450 XL(C+1) = XL(C) : YT(C+1) = YT(C) +
YS(C+1) : C = C+1 : HCOLOR=3 : GOSUB 100

460 C = C - 1 : REM right
470 XS(C+1) = XS(C)/3
480 YS(C+1) = YS(C)/3
490 XL(C+1) = XL(C) + XS(C+1)*2 : YT(C+1) =
YT(C) + YS(C+1) : C = C+1 : HCOLOR=3 : GOSUB
100

500 C = C - 1 : REM bottom
510 XS(C+1) = XS(C)/3
520 YS(C+1) = YS(C)/3
530 XL(C+1) = XL(C) + XS(C+1) : YT(C+1) =
YT(C) + YS(C+1)*2 : C = C+1 : HCOLOR=3 :
GOSUB 100

540 C = C - 1

600 RETURN
```

Figure 5.7: BASIC code. Image source: "Green Code on Black Background" by JOGOS Public Assets. Retreived from https://commons.wikimedia.org/wiki/File:Green_Code_on_Black_Background_2.png. License: CC BY-SA 4.0

A taxonomy of programming languages might separate different languages into levels such as low-level, high-level, and very high-level (see, e.g., Trois et al. 2016). I do not aim to discuss what the *right* taxonomy is. Rather, I wish to point to the mediating qualities of different levels of programming languages.

Programming languages and compilers are included in the list as they represent an embedded, translational interaction. Interacting with the computer through a low-level programming language such as Assembly shapes the user as user-playing-computer. With the introduction of higher-level programming languages, two interesting developments have happened. First, the textual interaction took on a different discursive character. Instead of writing esoteric, machine-friendly commands, the programmer could now interact with the machine by writing in an artificial language close to English (for a discussion about the implications of using English as the *de facto* language of programming, see, e.g., Guo (2018)). Second, there was now a need for yet another mediating entity to translate from these higher-level languages into machine code, i.e., the compiler. High-level programming languages therefore both represent a qualitatively different textuality and demand the enrollment of translating artifacts, transforming one symbolic representation into another and simultaneously black boxing said translational quality.

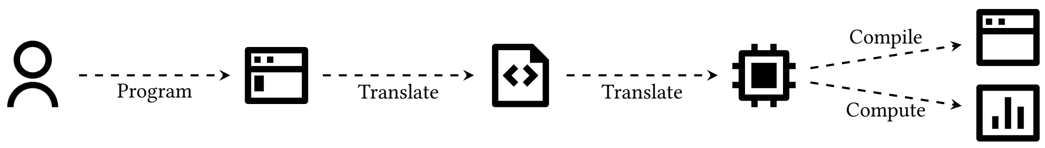


Figure 5.8: The programmer writes high-level code which is passed through layers of mediation and translation to become, ultimately, machine code which can perform the desired action whether computing or developing.

Programming in software rather than hardware also has a recursive character; it is development of software in software. One example is a compiler written in the same language that it compiles. The ultimate recursive software is, then, programming a system in itself from within, i.e., bootstrapping (this will be unfolded in section 5.3). Finally, programming in software marks the final departure from an embodied interaction. Software technologies do not allow for truly "bodily-sensory experiences" (Verbeek 2005, p. 228), although, as illustrated in chapter 6, keyboard shortcuts and finger dexterity still represent a remaining, albeit poor, form of embodiment in contemporary computing.

5.2.5 Personal computer operating systems

The operating system as an entity is, in essence, a catch-all term for software that allows one to use a machine. Operating systems were of course in use for decades before, but the release of MS-DOS in 1981 represented a transition in computational interaction. While systems such as the Commodore 64 brought the computer to the broader population, and the Xerox Star introduced many innovations that would come to influence computing (Johnson et al. 1989), MS-DOS in many ways represents the advent of the age of the personal computer.

MS-DOS, a commercial operating system, is emblematic of the growing division between user and programmer. The operating system provided an infrastructural ecosystem where people could install software that other people made. In the early days of personal computing, software was often published in hobby magazines as raw source code which you had to compile into working software on your own machine (N. Anderson 2018). MS-DOS, while not the only operating system, stands as a symbol for the transition into the commercialization of proprietary software, a move still contested by advocates for free software (see, e.g., Stallman 2015, p. 28). In fact, the source code was not made publically available until some thirty years later (Finley 2014), meaning that if the user of MS-DOS wanted to change something in the operating system before 2014—too bad.

It also marks a broader change in the relationship between the user and the programmer. Where computing previously belonged to the domain of the single user-programmer, multiple possible roles were now being created. Arguably, this transition was already well underway, such as with the introduction of the Xerox Star and its graphical user interface (Johnson et al. 1989). This graphical user interface presents a new degree of mediation and translation. In pointing and clicking, the responsibility of executing the precise commands to the system is delegated to the mediating qualities of the graphical user interface. The expertise of the system programmer had thus become materialized into the artifact, presupposing not only a division of roles and responsibilities, but also a temporal and spatial shift between use and programming. The operating system of the personal computer inscribes the user as just that—a user. In spite of this, there were and are alternatives to the user-programmer dichotomy of software in the age of the graphical operating system.

5.2.6 Emacs

One such type of software is Emacs, which is not a specific piece of software, but rather a family of related text editors that draw from a common ideological and technical heritage. The software was originally created as a series of macros[4] with the possibility of altering and extending this series.

[4]The name is a portmanteau of "Editing MACroS"

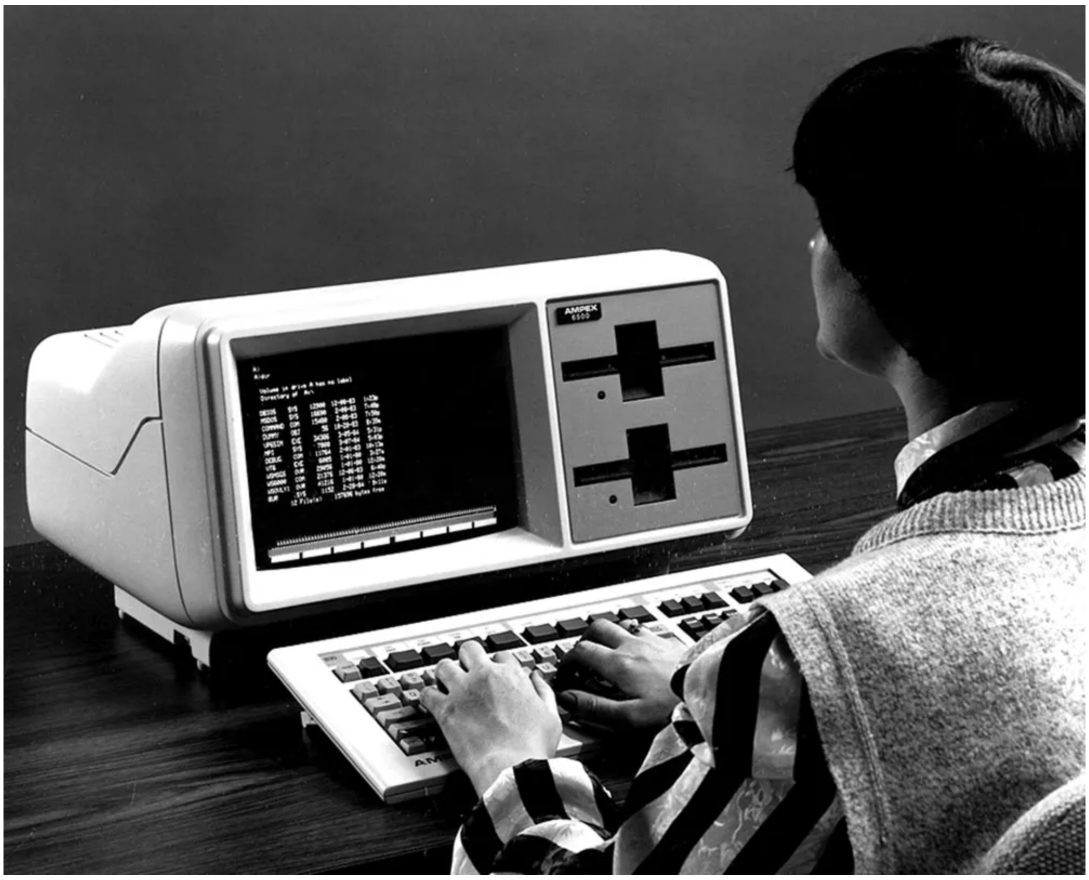


Figure 5.9: Woman interacting with DOS running on an Ampex computer. Image source: "MSDOS" by Rexhep-bunjaku. Retreived from https://www.wired.com/2014/03/msdos-source-code/. License: Fair use

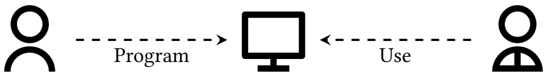
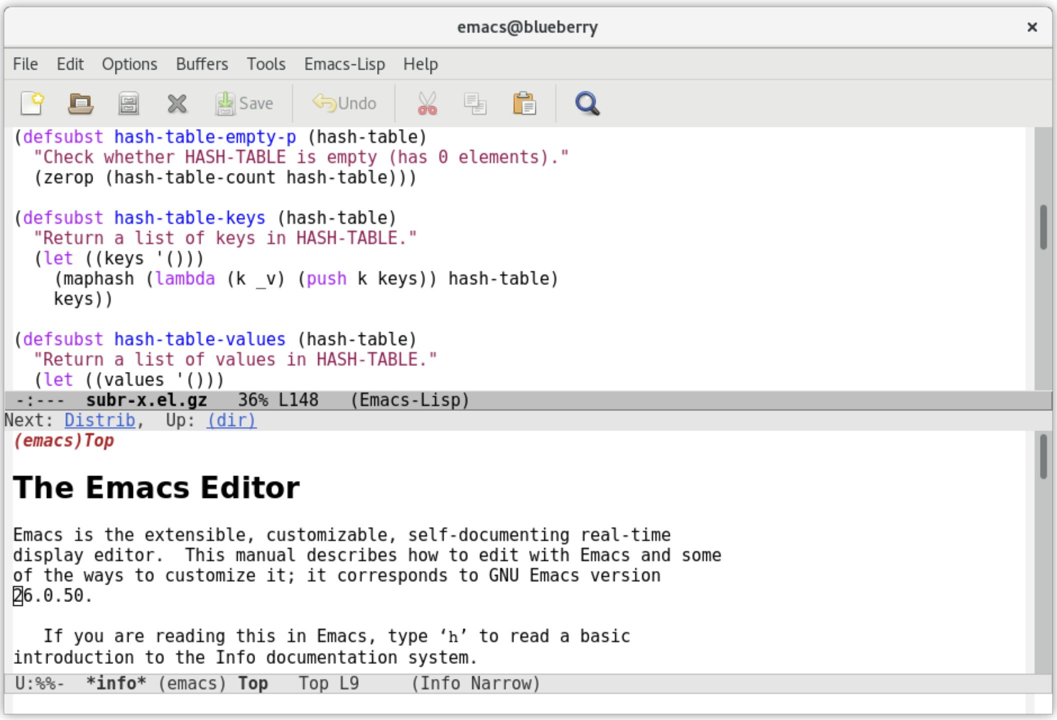


Figure 5.10: Operating systems conceptualize the user and the programmer of the machine as different people. Through black boxing "irrelevant" concerns and commercializing the software, the "user-as-consumer" category is created.

Emacs is included on this list due to its absolute insistence on dissolving the dichotomies of software. One of the first innovations of Emacs was the conflation of reading and writing text. Previously two separate modes of interaction, Emacs made it possible to do both in one "real-time display editor" (Stallman 1981). What is even more interesting with regard to this real-time editor is the absolute commitment to all text being treated as program code. Any interaction with Emacs is in the form of commands, i.e., interactive functions: The graphical user interface of modern Emacs distributions is simply a shell that calls Lisp commands under the hood. At the same time, Emacs provides an embodied delegation of commands into a multitude of keyboard commands in the form of chord-like combinations. A different mode is presented when pressing the `M-x` key combination: This brings up an interactive editor in which Lisp commands can be entered as text.

Even writing plaintext in the editor happens when letter characters trigger the `self-insert-command`. "These characters, which include letters, digits and punctuation, are normally all defined as commands to insert themselves into the text" (ibid.). The plans inscribed into the artifact are literally scripts. Of course, these commands can be altered as well, making it possible to re-script the semiotic aspects of the mediation. Emacs works through the so-called *modes* which define and frame the object of work. For instance, the HTML mode enables commands appropriate for writing HTML code. The Org mode[5] has Emacs itself as the object of work, allowing the system to get new, recursive qualities as a computational medium. I return to Org mode in section 5.3.

[5](The Org Mode Community n.d.)



```
(defsubst hash-table-empty-p (hash-table)
  "Check whether HASH-TABLE is empty (has 0 elements)."
  (zerop (hash-table-count hash-table)))

(defsubst hash-table-keys (hash-table)
  "Return a list of keys in HASH-TABLE."
  (let ((keys '()))
    (maphash (lambda (k _v) (push k keys)) hash-table)
    keys))

(defsubst hash-table-values (hash-table)
  "Return a list of values in HASH-TABLE."
  (let ((values '()))
-:---  subr-x.el.gz   36% L148   (Emacs-Lisp)
Next: Distrib,  Up: (dir)
(emacs)Top

The Emacs Editor

Emacs is the extensible, customizable, self-documenting real-time
display editor.  This manual describes how to edit with Emacs and some
of the ways to customize it; it corresponds to GNU Emacs version
26.0.50.

   If you are reading this in Emacs, type ‘h’ to read a basic
introduction to the Info documentation system.
U:%%-  *info* (emacs) Top    Top L9     (Info Narrow)
```

FIGURE 5.11: GNU EMACS 25 running on GNOME 3. IMAGE SOURCE: “GNU Emacs 25” by GNU Project. Retreived from https://commons.wikimedia.org/wiki/File:GNU_Emacs_25.png. License: CC BY-SA 4.0

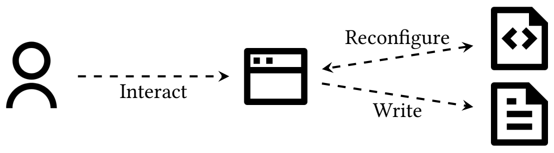


FIGURE 5.12: The programmer interacts with EMACS through writing text. All text is commands and the medium is inscribed with particular plans of action, rendering the dual options of working *through* and *on* the medium.

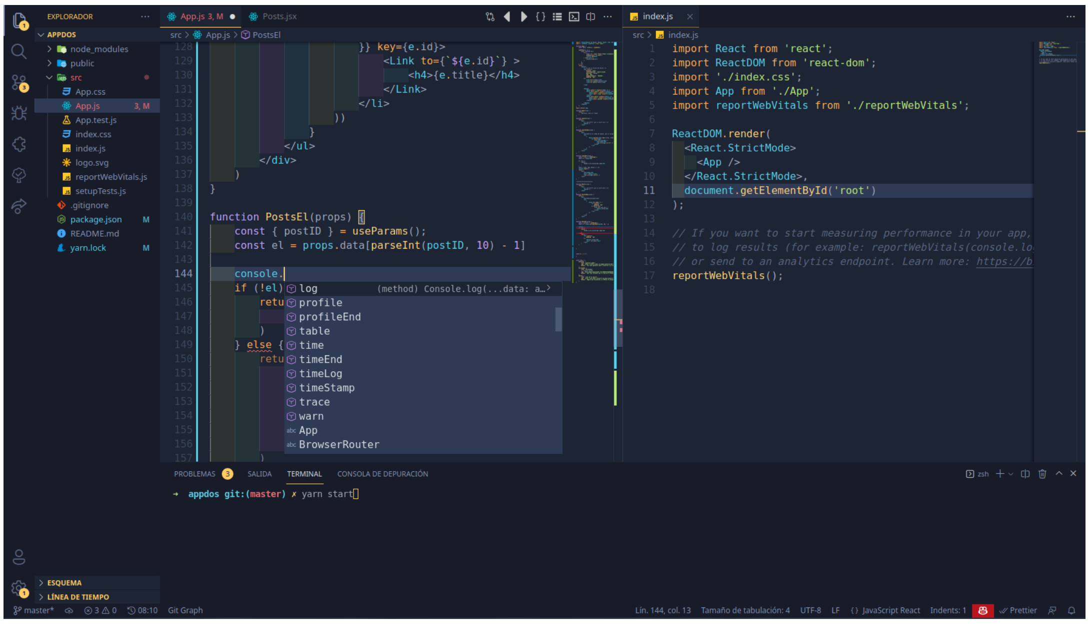
EMACS dissolves the ontological difference between text and code and the associated categories of tool, medium, and object of work. For instance, by editing EMACS’ textual configuration files or writing executable LISP code, it is possible to reprogram the system in real time. This has a few higher-order effects on the mediating qualities. First, the materiality of EMACS is code, the whole code, and nothing but the code. The LISP programming language is based on symbolic expressions that can be both executable code and data. Second, because everything is code, whether one is using or programming EMACS is an emergent property of the concrete interactions. The delegating qualities of the software are reduced by providing abstractions (e.g., commands) that can be reasoned on and with depending on one’s competencies: “Writing an extension is programming, but non-programmers can use it afterwards” (Free Software Foundation, Inc. n.d.). Third, the user configurations are flexible and not prescribed. One can take on various roles, depending on the type and extent of the activity undertaken.

EMACS is inherently distributional. It allows for remote access and can interface with other software such as web browsers, PDF readers, and LaTeX. Finally, EMACS is solidly founded in hacker culture and Stallman’s principles of the right to edit software. The ethical mediating qualities of the software reflect this commitment to what can be termed absolute transparency. Nothing is hidden away, and everything can be viewed, edited, and reasoned with. The downside to this commitment to extensibility and transparency is that the full complexity of the software is delegated to the user-programmer (Borenstein and Gosling 1988). An opposing movement can be found in the contemporary IDE which consolidates programming software as applications.

5.2.7 Contemporary IDEs

The current endpoint of programming modalities is the integrated development environment (IDE). The IDE is, in essence, nothing more than a collection of related tools for programming, typically a plain text editor with auxiliary tools such as syntax highlighting as well as tools for version management, a terminal window, and a file explorer. However, the IDE as a concept consolidates the various tools for programming into a neat package.

The concept of the IDE is not new, and seminal IDEs such as the Dartmouth Time Sharing System, the Maestro, and Smalltalk have thoroughly shaped the history of computing. What I have selected in this investigation are simply a few of many possible choices. Second, there is far more at stake in the history of the IDE than what is elaborated in the following, it is just not central to the present inquiry.



```
                        }} key={e.id}>
                            <Link to={`${e.id}`} >
                                <h4>{e.title}</h4>
                            </Link>
                        </li>
                    ))
                }
            </ul>
        </div>
    )
}

function PostsEl(props) {
    const { postID } = useParams();
    const el = props.data[parseInt(postID, 10) - 1]

    console.
    if (!el) log          (method) Console.log(...data: a…
        retu profile
             profileEnd
        )    table
    } else { time
        retu timeEnd
             timeLog
             timeStamp
             trace
             warn
             App
             BrowserRouter
        )
```

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);

// If you want to start measuring performance in your app,
// to log results (for example: reportWebVitals(console.lo
// or send to an analytics endpoint. Learn more: https://b
reportWebVitals();
```

```
→ appdos git:(master) ✗ yarn start
```

Figure 5.13: Visual Studio Code running in Ubuntu. Image source: "Visual Studio Code running in Ubuntu with custom complements" by Gátomo Dev. Retreived from https://commons.wikimedia.org/wiki/File:Visual_studio_code_updated.png. License: CC BY-SA 4.0

Many contemporary IDEs are commercially developed and sold, e.g., JetBrains' WebStorm, but even those that are not, like Eclipse, still largely operate under the application-centric paradigm. The Jargon File, a dictionary of computing-related terms developed among early hacker communities (Raymond 1999) has the following entry for the application:

app "*/ap/, n.* Short for 'application program', as opposed to a systems program. Apps are what systems vendors are forever chasing developers to create for their environments so they can sell more boxes. Hackers tend not to think of the things they themselves run as apps; thus, in hacker parlance the term excludes compilers, program editors, games, and messaging systems, though a user would consider all those to be apps. (Broadly, an app is often a self-contained environment for performing some well-defined task such as 'word processing'; hackers tend to prefer more general-purpose tools.) See *killer app*; oppose *tool*, *operating system*."[6]

The modern IDEs see the (professional) programmer as both conceptually, temporally, and spatially removed from what they develop. The black boxing of the translations and delegations into the application proper adds a complex layer of mediation. It is not easy to modify the IDE: Some, like Visual Studio Code, provide a rich interface for modifying settings and installing third-party extensions, while the software itself is not truly open. Despite the fact that its source code is released publically

[6] http://www.catb.org/~esr/jargon/html/A/app.html (visited Jun. 8 2023)

on GitHub[7], it is available only through the Microsoft Software License. The license further stipulates that, by using VISUAL STUDIO CODE, you agree that,

> The software may periodically check for updates and download and install them for you. You may obtain updates only from Microsoft or authorized sources. Microsoft may need to update your system to provide you with updates. You agree to receive these automatic updates without any additional notice. (Microsoft, Inc. n.d.)

Not only does the license affect the relationship between the programmer and the software, enforcing power dynamics of use and misuse, it likewise creeps out from the software's scope to the wider operating system. VISUAL STUDIO CODE, a hugely popular development environment, not only mediates between programmer and software, but also embodies and codifies Microsoft's corporate interests, delegating to the software duties of power and control.

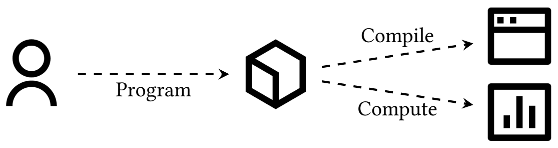


FIGURE 5.14: The programmer interacts with the contemporary IDE as a black box. It directs the action towards the production of artifacts or computation, yet the IDE itself is opaque and not immediately available as more than a consolidated set of tools.

Why this section on programming modalities? It provides context for the visions and realizations of computational media and serves as a springboard for the next chapter on the development of computational literacy in an age increasingly characterized by mediation and black boxing. From the historical overview we can gauge that a few core developments have taken place in the history of the (digital) computer.

First, there has been an ongoing substitution of human actors for technical actors. In the early days of the computer, humans acted as the translators between wish and reality in the process of programming. Second, the mediation between human and machine is increasingly complex. Translational processes and technical expertise have been subsumed into black boxes: Technical processes wrapped in neat application packages. While it may ease the use of computers, it also causes estrangement. Paradoxically, most of us have at our disposal vast capabilities for computing, but only a few possess the abilities for this. Third, as computer expertise has become technically embedded, computer users lose access to the expert. The programmer (in the sense from the early post-war days) no longer exists, having been absorbed into the software. Finally, and maybe most significantly, the user and the programmer are no longer the same. The activities of programming and use have increasingly become separated, if not physically then conceptually, for most people. The inscribed user—the role available to us in the interaction—has become something else than in the past.

These developments have, in effect, rendered many users of the computer impotent and incapable. Far from the early days of MIT's hacker spirit, the lack of computational literacy is a very real problem, despite the fact that we have more information available than ever before. It is therefore not just a matter of knowledge, although it is also that. It is a matter of the mediating qualities of the artifacts that we surround ourselves with. They inscribe particular users, not in a totalitarian way, but in a convincingly seductive way. Contemporary computing systems are built for *ease*. That the materiality of computers affects people is not a new insight, however. In the next section on computational media, I dive deeper into these visions of new media and the work that I have done on them. As a final note before that, however, I wish to address something that I did not explicitly go into detail with so far. The aforementioned characterizations of the human-computer interaction assumes an interactional chain between *one* person and *one* machine. As postphenomenology reminds us (Rosenberger and Verbeek 2015), human-technology mediation is never truly solitary—it is shaped by and in turn shapes the collective life around us in the form of cultural and social factors.

[7] https://github.com/microsoft/vscode (visited Jun. 8 2023)

5.3 Computational media

The following conceptualization of computational media draws on previously published work, primarily Nouwens, Borowski, et al. (2020) and Borowski, Fog, et al. (2022). This is supplemented by new work in which I revisited the empirical data from these previous studies.

The dominant mode of contemporary software is the application (see, e.g., Nouwens and Klokmose (2018) for a brilliant investigation of the application-centric paradigm). One of the consequences of this development is a clear separation between the document and the application. While the separation of data and program has been a foundation of computing since its early days, the application-centric paradigm not only emphasizes this division; it reinforces it and reconceptualizes the relationship between program and data. In the application-centric paradigm, software is hard to modify and often built from inaccessible or proprietary source code. However, even if the source code is freely available to access and modify (e.g., FLOSS[8]), there is still a large conceptual and practical gap from using to modifying one's software. Finally, the contemporary application, as it grew from the post-war North American office setting, is often oriented around a particular domain of work such as word processing or image editing and largely consolidated into the hands of a few corporate actors (Bergin 2006). Berry's notion of software as "prescriptive code" (Berry 2011) is quite fitting here; the application prescribes what can and cannot be done with it, shaping its interactive capabilities as well.

While some software ecologies offer a more interoperable and document-centric approach to, e.g., traditional office work (Microsoft Office Suite) or creative publishing (Affinity StudioLink), they are still not truly document-centric (see, e.g., Bier 1991) and fundamentally tied to a paradigm of (commercial) application *use*, not enabling the modification of the tools themselves. The domain of software for programming is likewise highly dominated by the application-centric software paradigm. Obviously, there are exceptions. For instance, there is a community of programmers who produce commercial software using modern implementations of Smalltalk such as Squeak and Pharo (Ducasse et al. 2016), while a (likely diminishing) group of developers still use the Emacs text editor—a remnant of the editor war of the 70s and forward (Auerbach 2014) that can be extended from within as a live software environment using the Org mode extension (E. Schulte et al. 2012). In fact, Emacs Org mode represents a computational medium that allows for literate computing-like interaction. I return to it in section 5.4.2.

The fact remains that modern programming often takes place in a bewildering array of applications. Some are for the programming activity proper, while others are auxiliary and infrastructural such as git for version control. Another interesting development is that for each new auxiliary task and associated tool, the IDE as application increasingly absorbs it into its interface. This is not to say that programming in an application is by definition bad. On the contrary, as this chapter shows, there are many benefits to using, e.g., a commercial IDE for programming. What the dissertation illustrates is that a common theme of contemporary software development is that, no matter the concrete tools employed, *development* of software is largely separated from the *use* of software. This is an important point. The inherent issues with such a separation have been acknowledged for several decades (see, e.g., Abbate 2018). Several serious efforts have been undertaken to address this conflict. Some proposed solutions for diminishing this perceived use-development gap have been concerned with creating educational initiatives and improving the opportunities for non-traditional users to grasp their software. Another branch of solutions concerns the imagination and development of new media for computing. The following section considers this latter category.

5.3.1 Visions of computational media

Throughout computing history, multiple visions of computational media have emerged. Some of them mainly existed in the minds of those who conceived them—whether for reasons of technical limitations or otherwise—while other were concrete manifestations of the visions behind them. Some of the more impactful and long-lasting visions come from people such as Alan Kay and Andrea diSessa. These present visions of the computational medium as a way of expanding one's capability for think-

[8]Free and Libre Open Source Software

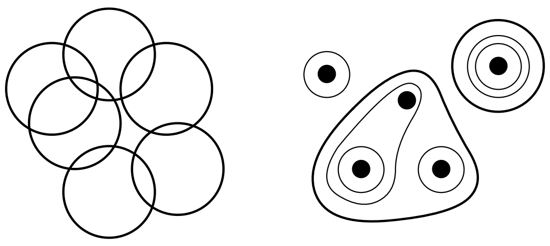


Figure 5.15: Applications (left) versus computational media (right)

ing and becoming. In essence, by using a computational medium, the medium-user hybrid turns both parties into something else.

Bush's idea of the Memex (Bush et al. 1945) still remains an oft-cited notion of the epistemological capabilities of computers to extend the human mind. With the vision of the personal dynamic medium (Kay and Goldberg 1977), however, the idea of the computational medium proper took hold, representing the mutual co-creation of both human and artifact in its moldable and interactive shape. Inspired by McLuhan, Kay posited that the most interesting aspect of a medium is what someone has to become to use it (Barnes 2007). In other words, the mediating or interactive qualities of the medium became a central point for Kay.

Kay's vision of the Dynabook represents just how such a personal dynamic medium might be realized (Kay 1972). This first vision was very much a technical imaginary, presented through drawings and "science fiction". Some years later, Kay and Goldberg realized their prototype in what they called "interim Dynabooks" built using the Smalltalk language and the Xerox Alto computer, both created for this purpose (Kay and Goldberg 1977). The vision of the Dynabook is interesting because of its insistence on context and learning as well as its suggestions of novel interaction forms. Simultaneously, its concrete implementation was "expensive, clumsy, slow, and—in practice—useless" (Löwgren and Stolterman 2007, p. vii).

Around a decade later, diSessa and Abelson presented their Boxer prototype, a "reconstructible computational medium" (diSessa and Abelson 1986). Boxer operates on the principle of naïve realism; everything is a box, whether text, multimedia, or executable code. Boxer never gained a wider adoption beyond the studies in which it was deployed. According to diSessa, this was in part due to the lack of resources and institutional support for implementing Boxer at a more cultural level (diSessa 2001, p. 236). For Boxer to become a viable alternative to other software paradigms, it needed to be approached in three levels: technical, small-world, and large-scale macroculture. The latter of these is similar to the postphenomenological perspective on people's macrocultural life-world. That is, those larger cultural contexts beyond the individual's immediate lifeworld.

While the concrete manifestations and visions vary, there are certain commonalities that allow us to provide a definition of computational media. First, they have a loose distinction between using the medium and (re)programming to the medium, both technically and conceptually. This is in part realized through a loose distinction between application and document. In computational media, whether the medium is an application or a document is a matter of context and use, not as an ontological definition inscribed into the medium. Finally, computational media work on the principles of seeding and extension. This means that, ideally, it should be able to seed the medium, for instance by providing basic editing features, and allowing people to extend upon these seeds. Figure 5.15 illustrates the difference between computational media and application-centric computing.

Both Kay and diSessa were inspired by Papert's work on LOGO (Papert 1993), particularly focusing on using computational media for teaching concepts from mathematics and physics. In my research on computational media using the *-strates family of software, I have explored the potential and challenges of computational media for the development of computational literacy outside formal education.

Even if much of the seminal work on computational media is decades old, the fundamental issues that they try to address have not been resolved. The mode of engagement in most contemporary software for programming is still highly symbolic and complex (Tedre 2020)[9]. Both Kay and diSessa have also entered the academic discussion about computational literacy and computational thinking as elaborated in chapter 3.

5.3.2 A contemporary anatomy of computational media

Since the introduction of Boxer and the Dynabook prototype, there has, of course, been tremendous developments of technological capacities in terms of both faster, more powerful hardware and the introduction and spread of new programming paradigms. So why hasn't computational media been realized beyond research yet? diSessa attributes the lack of success of computational media to a changing Zeitgeist in the '80s and '90s. A changing conception of the nature and purpose of programming combined with a focus on vocationalism meant that,

> [t]echnically, most people assumed the media that existed at the time were sufficient, with minor changes. As a consequence, few if any attempts emerged to improve the medium specifically as a literacy base (as opposed to commercial work to improve the "interface", speed, and prettiness) (diSessa 2001, p. 232)

In our work on a contemporary vision and manifestation of computational media, we have defined four characteristics as the basis for the new generation of computational media. These are shareability, distributability, malleability, and computability. *Shareability* means the ability to share the medium with other people. *Distributability* is the ability to distribute (parts of) the computational medium across devices in an artifact ecology (Bødker and Klokmose 2012). *Malleability* denotes the ability to change the medium from within, according to one's wishes and needs, and finally, *computability* is the ability to perform computations in the medium. Notably, all principles share the suffix *-ability*, which,

> Forms a noun from a verb or an adjective by changing from -able; ability, inclination or for a specified function or condition[10]

Therefore, I would argue, these words do not define what computational media *are*, but what people might *do* with such a medium, much in the spirit of Kay's original vision. They give shape to and influence the possibilities of interaction and mediation. They form the mediating aspects of the medium. The following is therefore not to be considered an analysis of any *essences* of computational media. Rather than being essential, these characteristics are affordances in the sense that they allow, shape, and hinder particular ways of interacting with the medium. These characteristics have proven useful as guiding principles for conceptualizing and imagining computational media as they might exist in a contemporary sociotechnical software ecology. At the same time, they represent ideals. The matter of fact is that any concrete implementation of a digital imaginary will have to come to terms with the practicalities of its context. Before moving on to these characteristics and the subsequent analysis of their mediating qualities, I wish to present the software that has been the basis for my studies.

5.3.3 Webstrates

This section present Webstrates, a research prototype developed in the research group that I have worked with. While Webstrates was developed before I started my PhD project, it has been the foundation for several of the inquiries that I made. In the following section I introduce Webstrates based on previous work (Klokmose et al. 2015). Part of this introduction is an overview of the four guiding principles that have influenced the subsequent computational media prototypes used in my studies, e.g., Codestrates. Similarly, while I was not part of the technical and conceptual development of Codestrates, I have nonetheless been intimately involved in subsequent use and reflections, particularly in Borowski, Fog, et al. (2022), where my peers and I have looked back on the history of

[9]The closing keynote was available here: https://drive.google.com/file/d/144DkaFHXhbYLd2pKe7PRP_DtNPKdR8-T/view (visited Dec. 20 2022; no longer online)
[10]https://en.wiktionary.org/wiki/-ability (visited Jun. 8 2023)

the software. Therefore, even though I have not developed Webstrates nor Codestrates, I have still conducted research based on these two systems. Figure 5.16 illustrates the timeline of the history of the Webstrates software family[11].

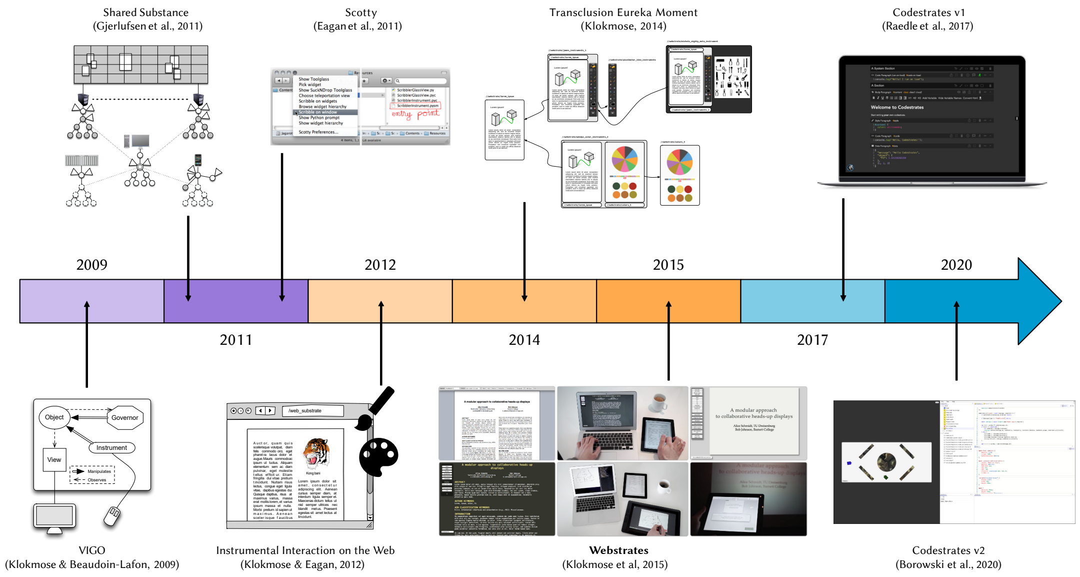


Figure 5.16: A timeline of the Webstrates software family

Webstrates[12] was initially developed as a prototype for exploring a contemporary vision and implementation of a "personal dynamic medium" (Kay and Goldberg 1977). Built on basic web technologies, Webstrates was presented as a "shareable dynamic medium" (Klokmose et al. 2015). Interestingly, Webstrates is both a piece of software in itself and a computational and extensible medium with infrastructural capacities. A Webstrates server serves web pages that are collaboratively editable. A single webstrate is a persistent, self-contained document. A document can contain markup (i.e., HTML and CSS) and program code (JavaScript). Therefore, a single webstrate contains both data and code, making it document and application in one. Any changes to the document is saved in the Document Object Model (DOM) of the webpage and synchronized across all instances of the same page using operational transformation for conflict resolution, much like Google Docs and similar collaborative systems. A fundamental characteristic of Webstrates is transclusion, the ability to include one webstrate in another through an `iframe` HTML element. If both webstrates share the same domain name, transclusion also allows for a shared JavaScript environment. The four principles of contemporary computational media that served to guide the development of Webstrates are, as stated, *shareability*, *malleability*, *distributability*, and *computability*.

Shareability denotes the ability to share a document (i.e., a webstrate) with other people. For a conventional file-based document, this often involves sending a copy of the file via email or a similar transfer technology. It further requires that the recipient has an appropriate application installed on their machine to open and edit the file. And finally, the process demands coordination between the involved parties for managing versions and merging files. Webstrates, in contrast, takes advantage of the infrastructural capacities of the web. When a new webstrate is created by the server, it is given a unique and persistent URL. Sharing a webstrate is as simple as sharing said URL. Further, by appending the URL with the query `?copy`, the server will create a self-contained copy of the webstrate at the URL and provide a new, unique URL for the copy. Webstrates also provides the ability to indirectly share by using the principle of transclusion. Through transclusion, one webstrate might be reused and shared between people working in different webstrates.

Distributability was achieved in Webstrates much in the same way as shareability. Due to the collaborative and web-based nature of Webstrates, distribution across devices could be done using URLs and the individual devices' web browsers. In plain Webstrates, distribution is either-or. Dis-

[11]For a brief overview, the reader is referred to the TOCHI presentation video https://youtu.be/VY9lX2obcHo (visited Jun. 8 2023)
[12]https://webstrates.net/ (visited Jun. 8 2023)

tributability is also not fully realized in Webstrates, as browsers on mobile devices are less likely to have tools for manipulating the DOM.

Malleability is the ability to change the system from within. As a webstrate is, in essence, just a persistent webpage, the native way to change this environment is through the browser's developer tools. The contemporary web browser is sadly far from Berners-Lee's vision of the combined web browser and editor (Berners-Lee 1996). However, by taking advantage of the HTML5 element attribute `contenteditable=true`, individual HTML elements in a webstrate can be made interactive by adding this attribute from the developer tools. A different approach was demonstrated by Klokmose et al. (2015) in which one webstrate was provided with the capability to alter another one through transclusion. Importantly, the malleability of Webstrates does not extend into its infrastructural aspects (i.e., the server and database), as only the individual webstrates can be changed.

Finally, *computability* designates the ability of the medium to run program code from within itself. Built from web technologies, Webstrates natively runs JavaScript in the browser's execution environment. Using the developer tools, it is possible to write `<script>` tags in HTML or use the interactive console. As a webstrate is persistent, any changes to the DOM remain over time, while the JavaScript runtime environment resets on reload. While these features theoretically allow for programming the shareable dynamic medium, using the browser's developer tool is inconvenient and not feasible for larger projects. Although the Webstrates File System (Antonsen et al. 2017) provides the possibility of cloning and synchronizing a webstrate with a locally stored copy, Webstrates presents itself more as dynamic media than as computational media.

In the following I present three environments based on the Webstrates platform that in different ways embody the mediating qualities of a computational medium: Codestrates, the computational lab book, and Codestrates v2. While I have not taken part in building the systems personally, I have been closely engaged with the research group behind them. Further, the development of the computational lab book and Codestrates v2 is the result of research that I have been conducting in collaboration with other members of the research group (see, e.g., inquiries 2, 3, and 4).

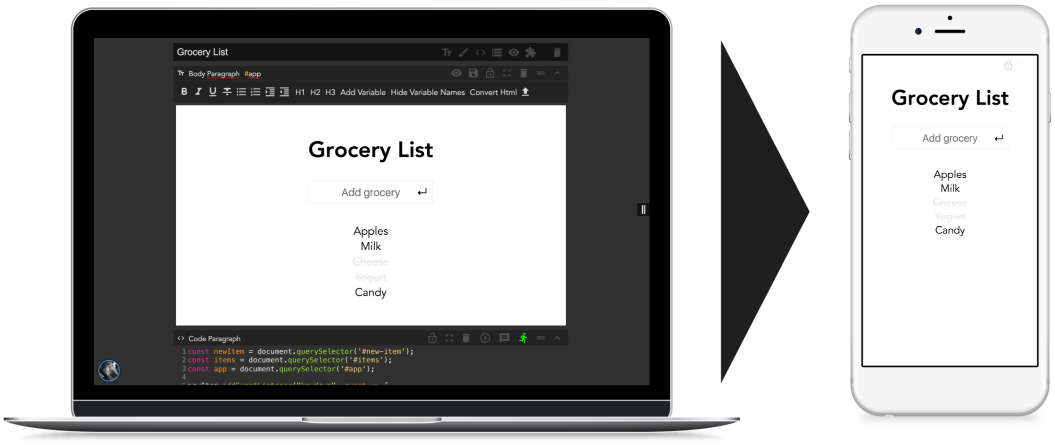
5.3.4 Codestrates

As Webstrates proved to provide a cumbersome user experience with poor capabilities for editing and programming, Codestrates was developed on top of Webstrates, providing an abstraction of code through the adherence to the literate computing paradigm (see section 5.4). This would ideally provide a structured and semantic interface for working with the computational and infrastructural possibilities of Webstrates. Codestrates[13] is therefore an implementation of a computational medium that allows for literate computing (Perez and Granger 2007). In contrast to popular literate computing software at the time such as Jupyter Notebook, Codestrates was intended not only to enable programming for interaction or computation, but to provide a computational medium for programming for software development. Section 5.4 explores Codestrates' literate computing characteristics further, including a fuller discussion of the benefits and drawbacks of the literate computing paradigm that ultimately led to the subsequent development of Codestrates into Codestrates v2. For the present purpose, I will briefly present Codestrates and how the principles were realized.

Codestrates, built on Webstrates, employs the same web technologies for its construction and extensibility. Drawing on a document metaphor, a user can create sections with paragraphs containing either data (JSON), content (HTML), or functionality (JavaScript). Figure 5.17 shows a grocery list being created in Codestrates.

The principles of computational media are realized in Codestrates in the following ways. Since Codestrates is built upon the infrastructural properties provided by Webstrates, they share many aspects of the principles. However, Codestrates expands on the user interface by providing new interactional qualities. For instance, while Webstrates is built around transclusion, Codestrates expands on and mediates this experience through *packages*, essentially reified, reusable webstrates containing functionality, data, or content (Borowski, Rädle, and Klokmose 2018). With the intro-

[13] https://webstrates.net/project/a-brief-overview/#codestrates (visited Jun. 8 2023)



```
const newItem = document.querySelector('#new-item');
const items = document.querySelector('#items');
const app = document.querySelector('#app');
```

FIGURE 5.17: CODESTRATES

duction of packages, CODESTRATES became modular in its construction, providing a more nuanced shareability. With CODESTRATES also came different views. Through a full screen "application" mode, the user-programmer can essentially switch between modes of development and use. This extends to distributability where one device can show the development view while another shows a use view, allowing for a different kind of distribution.

The functionality of Codestrates is, importantly, completely bootstrapped, meaning that it has been written in itself. This conflation of use and development provided an extreme degree of malleability, much like the one envisioned in the BOXER environment. However, as will become clear in section 5.5.3, this freedom to reprogram anything in the medium often led to unintended consequences. Finally, while individual paragraphs can be written containing JAVASCRIPT, the runtime is not truly live or reactive. If a user-programmer programs interactivity through the use of, e.g., event listeners, the runtime environment needs to be reloaded to prevent multiple listeners stacking.

5.3.5 Computational lab book

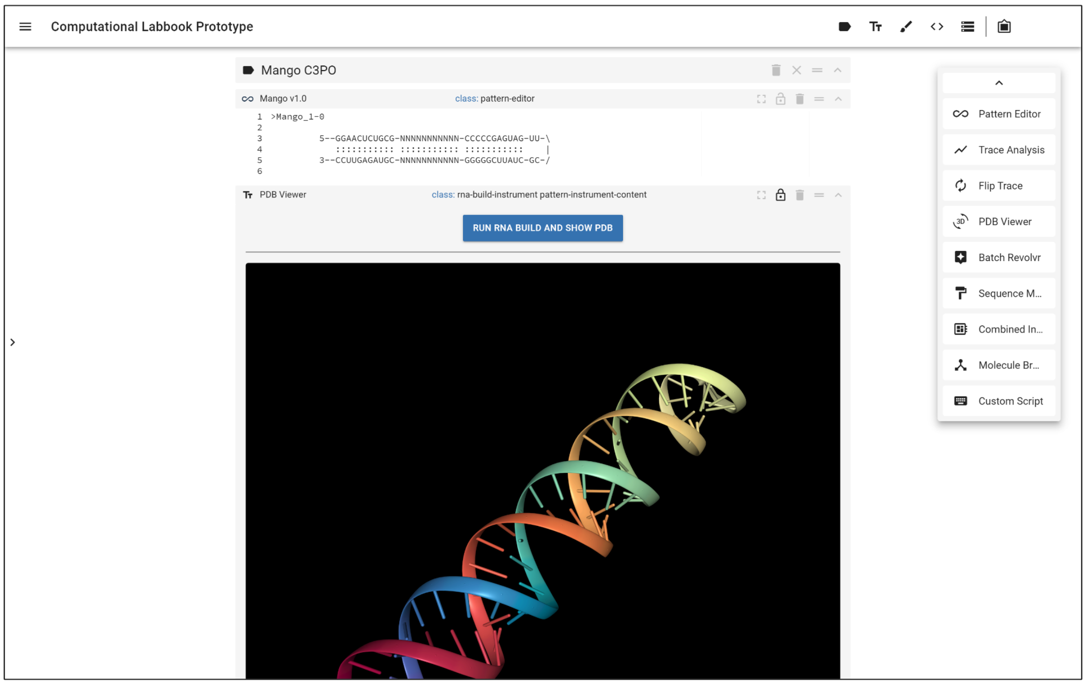
With peers, I used CODESTRATES as a malleable prototype to investigate how a computational lab book might be realized. Figure 5.18 shows the final prototype. The prototype was based on the idea of *instrumental interaction* (Beaudouin-Lafon 2000) and essentially builds upon the interface of CODESTRATES. Instead of structuring sections and paragraphs around web languages (HTML, CSS, JAVASCRIPT), the lab book is constructed as a series of domain-specific steps, where each step can be dragged and dropped from a toolbox of instruments. These instruments are, in practice, reified elements of the nanoscientists' existing pipeline of work. For instance, one of the instruments—the *pattern editor*—is nothing more than a plaintext editor. In the nanoscientists' previous practices, they would create and curate personalized libraries of patterns for reuse in loosely assembled plaintext files using a general-purpose plaintext editor (such as TEXTEDIT) on their "dry" machine[14]. By providing an instrument to store and retrieve these patterns, the computational lab book makes managing auxiliary files and data easier. Further, by providing a library of oft-used patterns, the practice of editing patterns was brought closer to the actual work domain of the scientists.

> *We didn't execute a whole bunch of codes in our previous lab. I was more using pretty well established software with a nice-ish GUI (...) I was still doing RNA work, but I wasn't doing any design of RNA sequence or anything like that. So depending on the research you'll have either really well-supported workflows ...or nothing ...or homebrew scripts.* (N10, inq. 3).

Each lab book built on CODESTRATES is web-based and completely self-contained. To accommodate the computationally heavy work that the scientists need to perform, and to allow for other programming languages than JAVASCRIPT, the lab book was designed to be able to delegate code to a more

[14]As the lab is sterile, the scientists each have a more powerful office computer for their "dry" work (i.e., planning and analysis) and a less powerful device in the lab for "wet" work

powerful server. The central Webstrates server was equipped with Docker containers to be able to execute other programming languages, e.g., Perl, and to return the results of these computations to the lab book requesting them. Through the enrollment of the central server, a different type of distribution comes into play. Whereas distributability in previous iterations of Webstrates concerned the distribution of software across devices, the type of distribution at stake in the lab book is also about delegation of computation.



```
>Mango_1-0

        5--GGAACUCUGCG-NNNNNNNNNNN-CCCCCGAGUAG-UU-\
           :::::::::::  :::::::::::  :::::::::::    |
        3--CCUUGAGAUGC-NNNNNNNNNNN-GGGGGCUUAUC-GC-/
```

Figure 5.18: The computational lab book built on Codestrates

The computational lab book is based on the principles of computational media. While e-Science and similar fields have for a long time experimented with enhancing the conventional lab book with digital capabilities, these existing examples are largely digitized versions of conventional lab books rather than truly computational software. For instance, the nanoscientists had already migrated their research activities to Confluence[15], an electronic lab notebook. While Confluence and similar electronic lab notebooks allow for wiki-like functionality and collaboration, they do not allow one to run computations directly inside. The computational lab book, embodying the scientists' scripts that were previously scattered and foraged, allows computations to be run. However, to the nanoscientists who are domain experts and "not concerned with the tool per se, but in doing their job" (Fischer and Giaccardi 2006), the computational lab book is probably seen as yet another tool. Good, but not qualitatively different from other tools and scripts.

> *Obviously, if you want to run a lot of* revolvr[16] *in your computer, it's going to collapse. You can just run it nice in the server and just forget about it, that's really nice.* (N11, inq. 3).

Another important aspect of the computational lab book is that the scientists work both *in* and *through* the artifact simultaneously. When creating, analyzing, and visualizing RNA structures, although still represented virtually in the lab book, these structures exist both in and beyond the system at the same time. The computational capabilities of the artifact are thus also used to operate on a world beyond the interface (at least conceptually).

> *Interviewer: "How crucial is the PDB viewer really?" N10: "It's not. It's just a check to see if there are any obvious problems like folding wrong. It's our best estimate of what a 3D structure would look like. And if the best estimate shows there is clashing going on, then it's probably not a good design (...) it's also pretty to look at."* (N10, inq. 3).

[15] https://www.atlassian.com/software/confluence (visited Jun. 8 2023)
[16] The name of a custom script

The mediating qualities of the lab book concern issues of translation and thinking through the medium in the spirit of Kay's vision. The computational lab book therefore embodies a certain "carrying capacity of ideas" (Kay 2013a). The lab book mediates both epistemologically, by magnifying particular aspects of the world (RNA structures), and ontologically (the RNA structures are purely representational and potential and do not exist outside the medium). Interestingly, it seems as if the mediation has aesthetic qualities as well, as evidenced by N10's statement above.

Finally, through the introduction of an external server for offloading computation, another degree of mediation is introduced. When offloading code to the server, a textual representation of potential code is sent along and a textual response is received back. There are processes of translation and black boxing at play. For the scientists, it appears that scripts are "magically" run faster than previously, while the fact is that the seemingly self-contained lab book now depends on networks of translation and mediation to function properly.

The lab book was built on Codestrates, which worked well as a prototyping tool for us. However, it turned out that the literate computing paradigm was not immediately well-suited for programming for software development, despite its enormous popularity in the data sciences. This issue is unfolded in section 5.4. As the literate computing paradigm proved lacking, a second iteration of Codestrates took place, leading to the aptly named Codestrates v2.

5.3.6 Codestrates v2

In contrast to Codestrates, Codestrates v2 was designed to be a computational meta-medium that serves as an authoring environment for other computational media. Codestrates v2[17] represents the newest iteration of the *-strates software family. As the literate computing paradigm and complete commitment to malleability of Codestrates v1 proved insufficient, the second iteration of the platform was developed. The unbound malleability was unfortunately negatively realized multiple times, for example when people accidentally overwrote or deleted the interface or the built-in functionalities. Codestrates v2 introduced a series of changes from the first version on both technical and conceptual levels. Some technical changes involved a new relationship between the textual representation of markup and their rendered counterparts as well as the introduction of a more restricted sharing model. This had consequences for how things are shared between clients, which was an unwelcome surprise to some people who were already familiar with the sharing model of Webstrates:

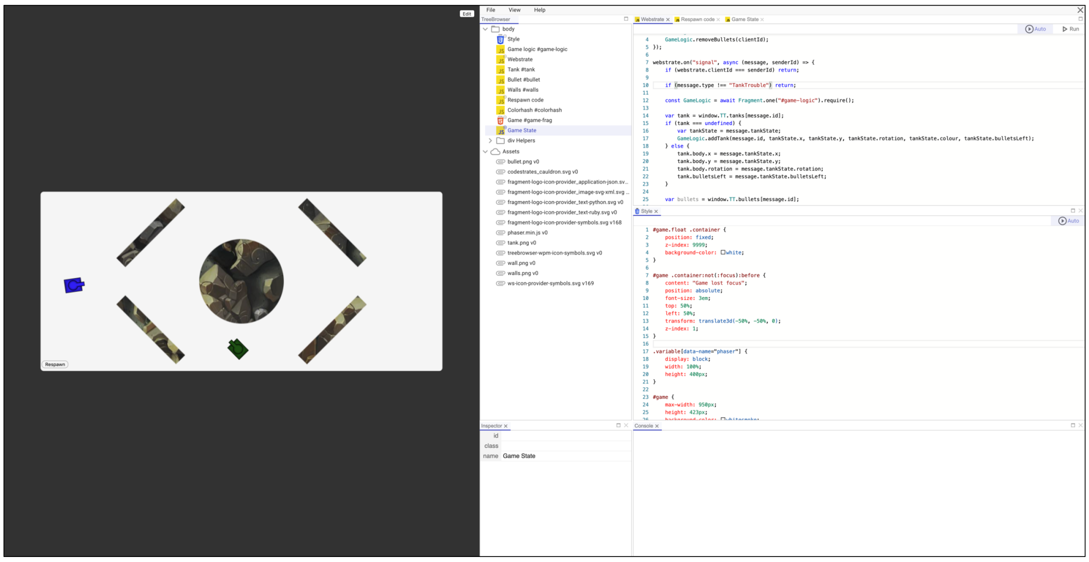
> *I also felt lightly betrayed by the tools because I felt like the very fact that you didn't have to do anything to persist [and share] stuff was the key value of Webstrates.* (P1, inq. 5).

Another change of a both technical and conceptual nature was the separation of editor and application. In Codestrates v1, the editor was part of the application on an equal level to other code, but in Codestrates v2 the editor was placed outside the document[18], preventing accidental breakage of the system. Further, the editor does not exist in the application until a button is pressed, at which point it is loaded into existence from an external server. Finally, the loose coupling of editor and application made it possible to have multiple editors that could be used depending on needs and wants. The default editor is named Cauldron (see figure 5.19). For a more in-depth explanation of the technical foundation of Codestrates v2, please see Borowski, Kristensen, et al. (2021).

On a more conceptual level, an important change is the metaphor. Whereas Codestrates v1 was built on the popular notebook format prevalent among literate computing environments, Codestrates v2 draws its inspiration from conventional IDEs: A separate editor representing all code fragments in a hierarchy with open fragments shown in a tabbed view (see figure 5.19). Finally, the concept of fragments was introduced as a catch-all term for textual representation of both program code and markup. However, as will be illustrated in section 5.5, it quite quickly became apparent that the idea of fragments was not well-received. Similarly, some user-programmers had conceptual issues with the relationship between HTML fragments and the DOM. In Codestrates v2, HTML

[17] https://codestrates.projects.cavi.au.dk/ (visited Jun. 8 2023)

[18] Technically, this was achieved by placing the editor outside the `<body>` of the DOM


Edit
Respawn
File
View
Help
TreeBrowser
body
Style
Game logic #game-logic
Webstrate
Tank #tank
Bullet #bullet
Walls #walls
Respawn code
Colorhash #colorhash
Game #game-frag
Game State
div Helpers
Assets
bullet.png v0
codestrates_cauldron.svg v0
fragment-logo-icon-provider_application-json.sv...
fragment-logo-icon-provider_image-svg-xml.svg ...
fragment-logo-icon-provider_text-python.svg v0
fragment-logo-icon-provider_text-ruby.svg v0
fragment-logo-icon-provider-symbols.svg v168
phaser.min.js v0
tank.png v0
treebrowser-wpm-icon-symbols.svg v0
wall.png v0
walls.png v0
ws-icon-provider-symbols.svg v169
Webstrate
Respawn code
Game State
Auto
Run
Style
Auto
Inspector
Console

```
    GameLogic.removeBullets(clientId);
});

webstrate.on("signal", async (message, senderId) => {
    if (webstrate.clientId === senderId) return;

    if (message.type !== "TankTrouble") return;

    const GameLogic = await Fragment.one("#game-logic").require();

    var tank = window.TT.tanks[message.id];
    if (tank === undefined) {
        var tankState = message.tankState;
        GameLogic.addTank(message.id, tankState.x, tankState.y, tankState.rotation, tankState.colour, tankState.bulletsLeft);
    } else {
        tank.body.x = message.tankState.x;
        tank.body.y = message.tankState.y;
        tank.body.rotation = message.tankState.rotation;
        tank.bulletsLeft = message.tankState.bulletsLeft;
    }

    var bullets = window.TT.bullets[message.id];
```

```
#game.float .container {
    position: fixed;
    z-index: 9999;
    background-color: white;
}

#game .container:not(:focus):before {
    content: "Game lost focus";
    position: absolute;
    font-size: 3em;
    top: 50%;
    left: 50%;
    transform: translate3d(-50%, -50%, 0);
    z-index: 1;
}

.variable[data-name="phaser"] {
    display: block;
    width: 100%;
    height: 400px;
}

#game {
    max-width: 950px;
    height: 423px;
```

id	
class	
name	Game State

Figure 5.19: A multiplayer tank game implemented using Codestrates v2 and edited through Cauldron

fragments are the primary entities, only rendered as DOM elements when running the software in the browser, but this was not clear to all user-programmers in inquiry 5.

In summary, Codestrates v2 has essentially stepped away from diSessa's vision of computational media as reconstructible and extensible. By implementing the Cauldron editor outside the document, the system has in practice been separated into hot and cold spots, respectively. In this terminology, a cold spot designates code that is frozen in place, not malleable, while hot spots are *alive* in the sense that they can become the objects of actions. This means that not everything is malleable in Codestrates v2 in contrast with the visions of Webstrates and the first iteration of Codestrates.

Interestingly, Codestrates v2 has shifted the locus of the computational medium. Codestrates v2 is a meta-medium in which one can create other computational media. By providing the ability to expose editable and executable code in the created computational media, user-programmers can provide hot spots for reconfiguring the codestrate. Less a computational medium itself and more akin to an authoring environment for other computational media, Codestrates v2 is reminiscent of the user interface software tools for creating other interfaces as discussed by, e.g., B. Myers, Hudson, and Pausch (2000).

The fact that not all parts of the medium is malleable means that Codestrates v2 is no longer a truly reconfigurable system. The conceptual and technical separation between application and editor provides stability and flexibility but has the side effect that the system takes on the uncanny role of looking like, but not being, an IDE. In the rest of the chapter I dig more deeply into the mediating qualities of computational media and how they shape and are shaped by the people using them. First, as the literate computing paradigm of Codestrates proved lacking, this programming modality is given special attention.

5.4 The unfulfilled promise of literate computing

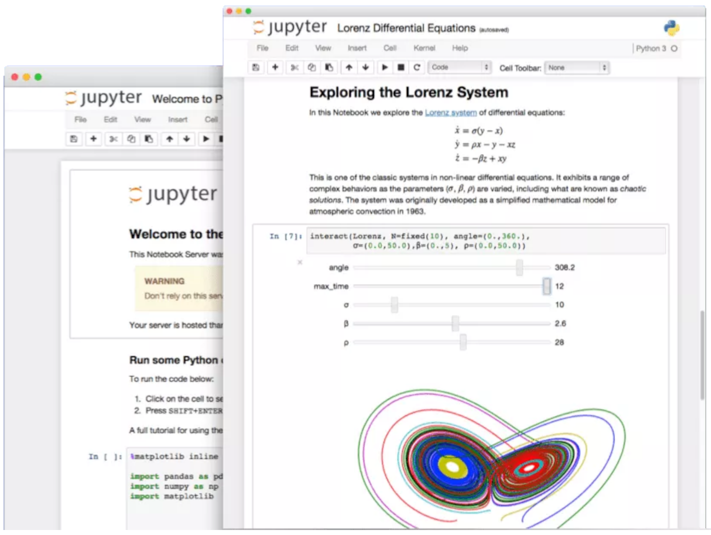
I previously hinted at how the shortcomings of Codestrates led to the subsequent development of the platform into Codestrates v2. While these shortcomings are probably attributable to several aspects of Codestrates, there are indications from my empirical work that the programming paradigm had a large role to play. In the following, I therefore address the literate computing paradigm theoretically and empirically by drawing on Fog and Klokmose (2019), Nouwens, Borowski, et al. (2020), and Borowski, Fog, et al. (2022) as well as preliminary findings from ongoing research.

This section first explores the broader issues and potentials of literate computing, a human-computer interaction paradigm made popular by the now omnipresent Jupyter software systems (Pérez and

Granger 2015). The literate computing paradigm presented a new way of authoring computational media and enabled alternative human-computer mediation (see figure 5.23) which could ideally support the development of computational literacy. By leveraging the familiar narrative structure of text and allowing for the co-production of executable code and other media forms, the literate computing paradigm should ideally inscribe the user-programmer as author more than developer (ibid.). This would in theory let people draw on their existing material and cultural competencies as also manifested in the consistent use of a notebook metaphor. Combined with the inherent opportunities for feedback (i.e., confirmations) from the "interactive computing" (Granger and Pérez 2021) environment, the literate computing paradigm seemed promising as a foundation for computational literacy development.

5.4.1 An overview of literate computing

To begin, I find it necessary to draw some lines. More specifically, a good starting point is to define what literate computing is in contrast to what it is not. The term was coined by Pérez (2013), who in turn drew upon Knuth's *literate programming* paradigm (Knuth 1984; Knuth 1992). However, the distinction between these two is not just a matter of semantics. While both paradigms are characterized by the coupling of two otherwise separate activities—programming and textual production—and a focus on writing programs as human-friendly explanations, they differ fundamentally on their ontologies. For instance how people are conceptualized, what materialities are involved, and what the purpose of the activities is. Concretely, literate computing is an activity in which programming and computation is mixed with the production and manipulation of rich media such as prose, images, video, and other digital media. The associated output from such an activity can be computational narratives, interactive documents, or even full-blown applications. The archetypical case can be found in the ubiquitous Jupyter Notebook (see figure 5.20).



```
In [ ]: %matplotlib inline

        import pandas as pd
        import numpy as np
        import matplotlib
```

Exploring the Lorenz System

In this Notebook we explore the Lorenz system of differential equations:

$$\dot{x} = \sigma(y - x)$$
$$\dot{y} = \rho x - y - xz$$
$$\dot{z} = -\beta z + xy$$

This is one of the classic systems in non-linear differential equations. It exhibits a range of complex behaviors as the parameters (σ, β, ρ) are varied, including what are known as *chaotic solutions*. The system was originally developed as a simplified mathematical model for atmospheric convection in 1963.

```
In [7]: interact(Lorenz, N=fixed(10), angle=(0.,360.),
                 σ=(0.0,50.0),β=(0.,5), ρ=(0.0,50.0))
```

angle 308.2
max_time 12
σ 10
β 2.6
ρ 28

Figure 5.20: Jupyter Notebook. Image source: "Jupyter Notebook" by Project Jupyter. Retrieved from https://jupyter.org/. License: Fair use

Literate computing is at its core a different human-computer interaction paradigm than what is afforded by most of the software presented so far in this chapter. More specifically, literate computing environments represent a type of software modality in which the boundaries between application and document blend together. In that sense, they have a lot in common with computational media. In one extreme, the software can even afford meta-programmatic capabilities, allowing for modifications to the software itself *in situ* and *in vivo*. If the history of modern software for programming has been characterized by an ongoing black boxing and translation of formerly human capacities, literate computing environments can be seen as a counterweight to this movement. The popularity of the particular *software genre* (Konzack 1999) called computational notebooks is a clear example of the potential afforded by these types of environments. It became apparent in Codestrates, however, that literate computing is not equally well-suited for all forms of programming. The type of programming activity matters deeply: For instance, computational notebooks, the main literate computing software genre, are mainly used to do programming for *computation*. One of the key findings from the retrospective autoethnography of computational media (Borowski, Fog, et al. 2022) was that the literate computing paradigm was ill-suited for programming for *system development*. It turned out that creating one system for multiple types of programming resulted in something less than ideal.

At the time of my study of literate computing (Fog and Klokmose 2019), the paradigm was still comparatively new as both phenomenon and research subject. Figure 5.21 shows an overview of both the number of articles published per year (blue line) and the types of systems investigated in each article (orange lines). As can be seen, the software genre was still being established at the time, although some fundamental work on literate computing had been done prior, e.g., Kery, Radensky, et al. (2018), Kery and B. A. Myers (2018), Rule, Tabard, and Hollan (2018), and Rule, Drosos, et al. (2018).

A key finding in my study was that popular literate computing environments largely draw on a notebook metaphor (e.g., Jupyter Notebook), and that these systems mainly provide means for programming for computation, not for system development. This was further cemented in the empirical findings from studies on Codestrates, which showed that the notebook format was not immediately suitable for software development. Further, the landscape of literate computing in 2019 was characterized by the dominance of software that did not allow for liveness and transparency. For instance, the hidden states of Jupyter-based notebooks are a source of divergence between live code and the textual representation of code (Singer 2020), due to the ability to run and re-run parts of the notebook in isolation from the rest. Another related issue is the clash between the non-linear execution model of the computer and the linear narrative presented in most literate computing systems. These issues are potential sources of distrust and alienation.

An important finding concerns malleability: Early visions of computational media such as Dynabook and Boxer emphasized the ability to alter one's software in-place. Even though the source code of popular environments like Jupyter Notebook and Live notebook is open source and freely accessible, the skills and tools needed to alter the software are beyond those required to use it. This issue is further addressed in section 6.1. These findings form the basis of the subsequent analyses of literate computing and the mediating qualities of computational media.

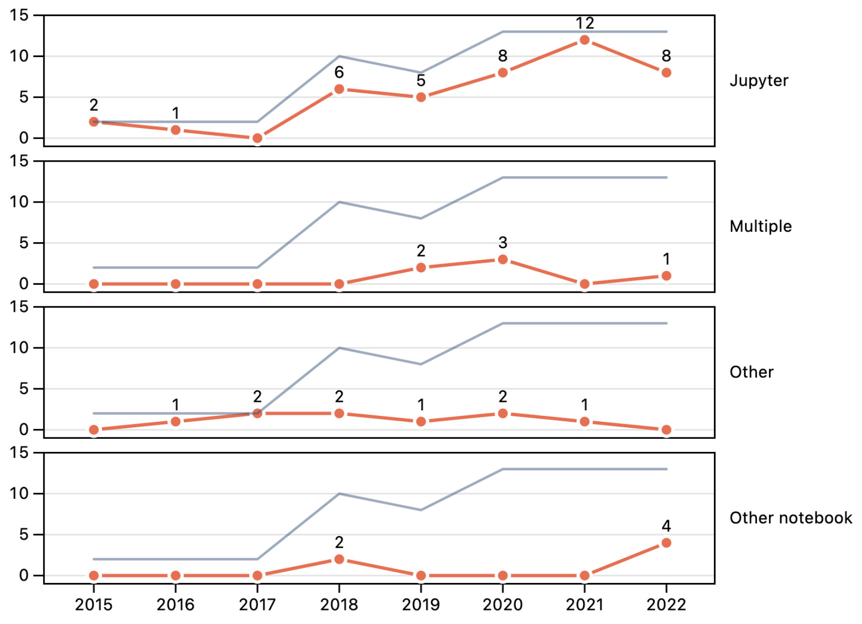


Figure 5.21: Types of systems in literate computing research. Orange lines represent research on the systems noted on the right, while the blue lines represent the total number of studies published on literate computing.

The terms *literate computing* and *literate programming* are often used interchangeably in research. This is less than ideal as the vision of literate computing presented by Pérez and Granger (2015) showed great promise in building upon prior explorations of computational media such as diSessa's Boxer environment. This, too, was the ideal goal of Codestrates. However, conflating the terms runs the risk of diluting the unique position of the medium and reduce it to a mere tool. Finally, literate programming specifically denotes *programming* as the key activity, configuring certain kinds of possible roles for the individual engaged in the activity, while possibly barring others in the process. One might argue that the terminology does not matter as long as we are in agreement on what we are talking about. However, I would argue that these are two different paradigms, and that we must treat them as such, lest we risk diluting a still promising computating paradigm. Table 5.1 shows selected differences between the paradigms.

	Literate programming	Literate computing
People	Experienced programmers	Non-system programmers
Materialities	Textual documents	Interactive media
Temporality	Serial	Recursive
Purpose	Programming for system development	Programming for computation

Table 5.1: Key differences between literate programming and literate computing

Literate computing as a unique activity is further complicated by the fact that the meaning of the term was never really stable from the outset. In my current research I have found clear signs that the discourse of not only the activity itself, but also of the associated people, media, and outputs is in a state of flux and has been for quite some time. This is not expanded further here, but will serve as a reminder that discourses matter. Literate computing and literate programming are different programming modalities, and their mediating qualities are similarly unequal (see figures 5.22 and 5.23). I make the argument, in accordance with the original vision by Pérez and Granger (ibid.), that the central aspect of literate computing is the *activity* itself. It is a mediating and “intellectual computing activity” (Galey and Ruecker 2010) that ties together the capabilities of the software itself, the person engaged in the activity, and the resulting product of the activity. As such, literate computing is a highly mediated activity that both depends on and recreates particular user-computer configurations. This is succinctly captured by O’hara, Blank, and J. Marshall (2015):

> The focus of literate programming is to document a program. In this manner, it is an inward-facing document, designed to explain itself. On the other hand, literate computing is meant to focus on the computational goals, rather than on the specific details of the program. The goal of literate computing is not to explain the workings of a program to programmers, but to explain a computational problem to a wide audience. Literate computing does not subsume literate programming—it is something altogether different.

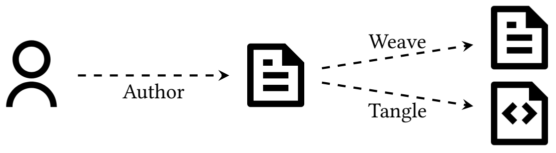


Figure 5.22: Literate programming is textual production that is translated into new conceptual spaces.

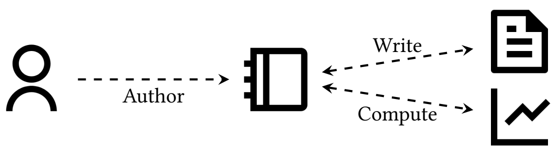


Figure 5.23: Literate computing where code, prose, and multimedia lives in the same conceptual space.

5.4.2 Literate computing and computational media

In brief, literate computing environments are those types of software that enable literate computing activities. This is not a binary true or false but rather points on a continuum. Even though both kinds of software seek to blend development and use, literate computing environments are not (necessarily) computational media. Computational media are, as defined in section 5.3, characterized by a close coupling between use and development and by blending the application and document. A salient feature of computational media is the ability to alter the software itself. Despite the Jupyter environment not being easily malleable, the resulting notebooks do exhibit computational media characteristics by allowing (some degree of) shareability, distributability, malleability, and computability. A distinction can further be made in the terminology: Computational media are descriptions of software artifacts, while literate computing describes a computing modality. It is a small, yet important difference.

To better understand the characteristics, variations, and potentials of literate computing environments, I selected and analyzed twelve systems that in some way embodied the literate computing paradigm. Three of these are historically important and were selected for their visions and the ways they (re)imagined the mediating qualities of software: Boxer, HyperCard, and Smalltalk. Another four are contemporary systems that are either commercial or otherwise officially supported and publically available: Mathematica, Jupyter Notebook, Observable, and Notion. Finally, the latter five are environments that are either experimental or results of ongoing research projects: iodide, Live Notebook, Codestrates, Eve, and Capstone. Importantly, not all of these environments are literate computing environments as such, but they each exhibit characteristics of the literate computing paradigm. Referencing my workshop paper, Lau et al. (2020) also included all of these in their large-scale study of literate computing software.

Based on this analysis, I have provided a conceptual space for both understanding literate computing systems and providing directions to other researchers. This consisted of a number of themes such as system metaphor, intention, threshold and ceiling, liveness, malleability, community, and collaboration. The themes are both relevant by themselves and in the broader question of the development of computational literacy through literate computing. For instance, the focus on threshold and ceiling is also captured in the design mantras of the Lifelong Kindergarten research group at MIT who further emphasize the “wide walls” (Resnick et al. 2009). That is, to support the development of computational literacy for novices, software environments must ideally be easy to use, capable of complex activities, and support multiple ways of using and interpreting them.

It is worth noticing that literate computing environments do not by necessity need to be for novices to be useful. The Emacs Org mode is, for example, clearly oriented towards people with computing experience, not least because of the possibility of reprogramming the system itself in real-time using Lisp commands. In its commitment to liveness, malleability, and transparency, this major editing mode for Emacs provides a literate computing environment that aims to erase the difference between use and development. Smalltalk and HyperCard similarly provided capabilities for being altered and extended (Justice 2021).

5.4.2.1 Literate computing in Codestrates

As stated in the introduction to Codestrates in section 5.3.4, the environment was introduced as a computational medium built around the literate computing paradigm. Drawing from Kay’s Dynabook, the system was presented as a “shareable, dynamic medium” (Rädle et al. 2017). In contrast to the file-based computing paradigm dominant in most contemporary software, including Jupyter, Codestrates employed a document-centric software approach. This means that the distinction between application and document was more a matter of activity than a matter of ontology. Even though I was not part of the development of Codestrates, I have been involved in the subsequent analysis and evaluation of the platform.

While computational notebook environments like Jupyter lend themselves mainly to programming for computation, the vision behind Codestrates was the provision of an authoring environment for software development. However, it still used the vertical metaphor from computational notebooks which proved challenging for some:

> *(...) the problem with [Codestrates] was that it sort of had that computational notebook format. Which is really practical when you need to be processing some data linearly down-*

> *wards, but as soon as it's sort of event-based and different functionalities, then it just becomes a hell to have a wall of code.* (P4, inq. 5).

Further, the design decisions behind Codestrates made it browser-based. Providing a computational medium in a browser environment unfortunately meant blending genres in an unfamiliar way:

> For users, the unconventional way of programming in a notebook environment could cause confusion: Being able to run JavaScript code inside an application that is already running is contrary to how web applications are usually developed. But also the introduction of the run-on-load feature of paragraphs caused confusion, e.g., when users were not sure how to execute something after the page was loaded. (Borowski, Fog, et al. 2022)

Related to the design choices of Codestrates and its infrastructural foundation, we also found a series of issues with employing the literate computing paradigm in a web-based, collaborative environment. One of the key properties of contemporary computational media is, as stated, their shareability. In Codestrates, being a literate computing environment in which there is no distinction between development and use, at least parts of the system state was inadvertently distributed across the network. This caused issues such as paragraphs and sections suddenly moving, much like in collaborative writing software such as Google Docs and Overleaf. Further, it turned out that Codestrates as a literate computing medium suffered from many of the same issues as those computational notebooks that it took inspiration from: Cell execution order, "Schrödinger's notebook" (i.e., the uncertainty of state), no concurrency, and no introspection (Singer 2020).

Using literate computing to create a computational lab book, on the other hand, turned out to be surprisingly well-received. The computational lab book was in essence a set of domain-specific abstractions built on Codestrates. The lab book was, by definition, a literate computing environment, but it allowed for multiple levels of abstraction through the provision of "instruments", i.e., reified functionality (Beaudouin-Lafon 2000). This was a deliberate design decision based on findings from inquiry 2 that programming was a necessary, but not primary, part of nanoscience. In doing so, the computational lab book can be seen as an *instrumental interaction literate computing environment* that was able to scaffold the scientists' computational activities. Even though they still took advantage of the same custom scripts for doing their work, these scripts could be abstracted away in the interface, further allowing collaborators with different skills to relate to the code in multiple ways depending on their needs and wants.

This suggests that literate computing was and is a viable programming paradigm for computational media, but it demands the right circumstances to be meaningful. The nanoscientists appreciated the linear notebook structure and the reified instruments which seemed to be familiar to them as natural scientists, which is not surprising, given the popularity of computational notebooks (Granger and Pérez 2021). The difference from generalized notebooks like Jupyter is that our computational lab book provided a domain-specific, instrument-based interface. It therefore seems as if the literate computing paradigm is well-suited for building prototypes and bespoke software. Second, literate computing environments demand the right competencies (i.e., literacy). This is addressed in the following section which, like this one, specifically discusses the first version of Codestrates unless otherwise stated.

5.4.3 Literate computing and computational literacy

"Beware of the Turing tar-pit in which everything is possible but nothing of interest is easy." (Perlis 1982) This quote describes one of the main issues with the Codestrates literate computing platform, i.e., Codestrates v1, which is just referred to as Codestrates throughout the dissertation. Computational notebooks like Jupyter are scaffolding devices, clearly demarcating possible user roles and the actions that they are allowed to take. Codestrates, on the other hand, suffered from an inbetweenness: Not quite notebook, not quite web application, and not quite computational medium.

One reason that Codestrates never seemed to hit the *right* mark as a computational medium can likely be chalked down to people having no way of drawing on their previous experiences. Even experienced programmers had not developed any substantial material intelligence with Codestrates, and so they struggled to make sense of their existing competencies in this new medium. In fact, as is made clear in chapter 6, people with less programming experience seemingly had a lower entry level into computational media. In the game design challenge, one participant noted: "*And then I didn't want to sacrifice the tool set I already knew*" (P3, inq. 5). Although not particular to literate computing, it does point to a significant issue when introducing people to new media and programming paradigms.

Another issue is the very loose coupling of process and product in literate computing. As laid out in chapter 4, an important aspect of mediation is trust. This is, however, one of the downsides to literate computing. There are two main issues at stake here: One is inherent to the malleability of some literate computing systems, e.g., Codestrates, while the other is related to the free-form way of authoring and combining data, executable code, and textual descriptions.

Knuth's vision of literate programming emphasized the construction of "comprehensible" programs by introducing concepts in "an order that is best for human understanding" (Knuth 1984, p. 99). This decree is suitable for literate programming as software development, where the authoring medium and its resultant files are separate from each other. The same could in principle be said about any kind of functional programming paradigm. In literate computing, however, such an imperative does not work. In those systems, the medium and the output is both temporally, spatially, and conceptually entangled.

Finally, most dominant literate computing environments do not support the wide walls and high ceiling advocated by computing education researchers. The floors are low—indeed, one of the benefits of computational notebooks, which was also discovered to be the case with the computational lab book, is the ease of entry. The infrastructural element of these systems provides a plug-and-play experience for most users. In computational notebooks, the wide walls and high ceilings mainly open up for increasingly sophisticated data work and visualizations. They are largely intended for computation and data processing. In contrast, Codestrates presented a vision of literate computing in which the walls were very wide and the ceiling tall. The medium could be interpreted and used in many ways and for a variety of complexities. Unfortunately, the floors were also too high.

These findings illustrate a few important aspects of the relationship between literate computing and computational literacy. Because of the required competencies and the material intelligence needed, computational notebooks and Codestrates demand different *types* or modes of computational literacy. They are tied to their ideologies. Computational notebooks are created as tools for work, and they are not to be reasoned with. They are black boxes that make it possible to work with data. In contrast, literate computational media are oriented towards software as executable, extensible code and are recursive in the sense that they make it possible to work on themselves.

Just as Codestrates moved on to Codestrates v2, it seems that other researchers of literate computing have similarly abandoned the idea that the programming paradigm has any bearing outside computational notebooks (see, e.g., figure 5.21 earlier this chapter). Despite this, there are still researchers who argue for opening up the field again and designing for diverse user groups outside computational data science (Lau et al. 2020). In the same vein, these authors suggest returning to the visions of Boxer, HyperCard, and Smalltalk which are computational media *par excellence*. This rest of this chapter is a mediation analysis of contemporary computational media in the *-strates family.

5.5 Findings, or how computational media mediates

This final part of the chapter presents findings and reflections on computational media based on my empirical work. I have sectioned these into a number of themes that are of particular importance or interest with regard to the mediating qualities of computational media. These do not correspond one-to-one with the aspects presented in table 4.1. For instance, the question of use and development in the following section is not *confined* to the interactional aspect of mediation alone. The use of an artifact is, naturally, deeply connected to other concepts such as trust and empowerment. To contextualize and show the richness of people's experiences, the following contains a substantial amount of quotes drawn both from my published work and the original data. These help illustrate exactly how the mediating qualities of software influence computational literacy in practice.

5.5.1 Use and development

As software for programming, computational media *by definition* seek to blend and diffuse the use and the development of an artifact. The visions of computational media speak highly of this effort (diSessa and Abelson 1986), but the case is not so clear-cut in practice. In fact, apart from the odd research prototype and small communities based around languages like SMALLTALK, computational media never seemed to have gained any proper traction. Why? One answer could, in fact, lie precisely in the conflation of use and development. An oft-cited software artifact in end-user programming research, MICROSOFT EXCEL, in which people can easily switch between use, computation, and programming[19], is still used by three-quarters of a billion people (Jamieson 2021). Similarly, literate computing environments such as JUPYTER are hugely popular (Granger and Pérez 2021).

There are probably multiple issues at play. One of them likely concerns the *role* of programming in the activity: In EXCEL and literate computing environments, the purpose of programming is mainly to compute, to transform data. This is unfolded in the next chapter. In the visions of computational media, programming concerns not only computation but also (re)programming the systems themselves. I return to this point in section 5.5.3 below.

Another is in terms of breakdowns. Breakdowns in computational media seem to be caused largely by context-switching between use and development. P12 explained how reloading the page threw him out of context:

> *(...) you're insanely dependent on that, this thing of jumping back into the development environment, well, then you're in the context that you were right before (...) You become completely confused when you're programming, sometimes. "Where am I? What the hell was it that I was doing?" I've experienced that more times in this than I normally do.* (P12, inq. 5).

In fact, this kind of workflow disruptions might more accurately be called breakasides. The ability to easily switch between activities of programming and use is supported in most popular IDEs for web development, but CODESTRATES v2, despite being web-based, presented new challenges:

> *(...) so usually when I'm working on the web, I have a live environment where whatever changes I make to the files, automatically propagated to the website and refreshes. So the experience was quite similar. However, when I worked with Cauldron, this meant that whenever I refresh the page, I would have to open the IDE again and go to the file that I was working at (...) it was annoying.* (P8, inq. 5).

Similar comments were made by other participants. It is not a full-on breakdown, but an "annoying" breakaside that hinders fluency. The cause of these kinds of disruptions can be found in the fact that our implementation of a contemporary computational medium lives purely on the web. Web-based computational media thus present their own challenges. A web browser embodies many different possible interpretations for its use. However, it seems as if programming is not readily a part of these possible interpretations, at least for some people:

> *(...) it's just as much the fact that it's in a browser, that then I kind of stop thinking in architecture (...) normally, you wouldn't sit and write an entire application or something inside the browser. (...) so I think "*now *I'm developing inside an IDE, and* now *I'm running it in the browser". And here it's been sort of weirdly mixed.* (P9, inq. 5).

Interestingly, in the original vision of the web presented by Berners-Lee, the web was just as much a space to create as a space to read (Bouvin and Klokmose 2016). Granted, the type of creation concerned writing documents rather than programming, but the idea of the web as a hypertext system for both reading *and* creating seemingly died off years ago. For some people, however, the multistability of CODESTRATES v2 is seen as beneficial and empowering, providing capabilities in the spirit of the original web vision:

[19]Using VBA (VISUAL BASIC FOR APPLICATIONS)

> *(...) because it's one thing to see that in particular, how it works, I mean just, like, the end result (...) But it also gives you something, the fact that you can just go in and see "okay, how is this implemented?"* (P4, inq. 5).

It is difficult to say exactly what causes these two different interpretations of the use-programming conflation. Likely, it is a result of the mediation itself in which both the technical capabilities of the software and the self-concept of the user-programmer shape the interpretation. This is explored further in chapter 6.

Computational media authoring software such as Codestrates v2 is not only a type of computational media itself. It is also a computational meta-medium as it can be used to produce other forms of web-based computational media. This further muddles the boundary between use and development as user-programmers must also take into consideration future user-programmers of the finished artifact:

> *(...) maybe for some parts I don't need to expose it, but then for the bird part, I can sort of expose it to the outside. Then the idea is to just make it simpler for non-programmers to, like, sort of understand what's going on* (P5, inq. 5).

By exposing code in the form of hot spots, P5 successfully completed the final tier of the game challenge. Interestingly, P5 was the only participant to use the meta-programmatic aspects of Codestrates v2 in this way: While others created interactive games with dynamic rules, P5 seemed to be the only one to see and appropriate the possibility of exposing fragments to future user-programmers. In fact, Codestrates v2 is itself structured around cold and hot zones. In the first version, Codestrates v1, everything was unified with no special privileges given to any part of the code, editor or otherwise. This meant that user-programmers could accidentally break the system by, e.g., accidentally removing DOM elements belonging to the editor. By moving the Cauldron editor outside the `body` of the DOM, Codestrates v2 would more clearly separate these conceptual areas. The separation between hot and cold zones seemingly solved some issues related to accidental breakage of the system. Still, at least one group in inquiry 5 ended up with a nonworking program state that they could not recover from. It also turns out that the close integration of use and programming presented some users with unintended consequences related to the instability of the artifact.

5.5.2 Semantic shift

As explained in section 2.1, one would be hard-pressed to define the *essence* of artifacts in general. Postphenomenological researchers might call this the multistability of artifacts: They mean different things to different people across different times and contexts. This is likely even more true in the case of software, particularly the kinds of software that belong in the category of computational media. Indeed, one could argue that a defining characteristic of computational media is their ability to be more-than-one-thing-at-once. Borrowing a term from Mol (2002), this *ontological multiplicity* of computational media mean that they at times are in flux, balancing between meaning. For instance, when interviewed about the use of Codestrates v2 and Cauldron for game development, one participant remarked,

> *So, we had this `WASD` mapping and that means when you press that your tank moves around. But that also meant that if you didn't click out of Cauldron, it would write into the text file that you were working on, the `W-A-A-A-A-S-D` and so on* (P8, inq. 5).

This is a kind of breakdown, but it is not a breakdown of workflow. Rather, it is a semantic breakdown in which the meaning of the software changes abruptly. I term this a semantic shift. It is different from the breakdowns and context-switching presented in section 5.5.1, as the breakdowns related to semantic shifts are caused by the unstable nature of the artifact, not by a purposeful context-switching initiated by the user-programmer. We can see how Codestrates v2 enacts a technological intentionality that is unpredictable. There seems to be an additional component to the instability of the artifact. For some participants in the study, it was difficult to reason about the state and behavior of the software: "*(...) there is no reason why [the editor] disappears if it had been open the last time*" (P11, inq. 5).

This kind of unintentional semantic shift likely influences both feelings of trust in the artifact and the individual's feelings of competence. Interestingly, the collaborative nature of contemporary computational media can even delegate semantic shifts across the network:

> *I think it was primarily the aspect of one person being in the process of developing something and two other who would like to run something, and then it fucked up quite a lot.* (P11, inq. 5).

While the person currently developing did not experience any shifts as they were intentionally switching activities, the two others were caught in an unanticipated and unwelcome disturbance to their current activity. Two design choices of Codestrates v2 seemed to be causing many of the semantic shifts experienced by user-programmers in the study. One is, as mentioned by P11, the fact that all changes to the textual code is immediately shared across all connected clients. Another is the fact that the web-based foundation of computational media by necessity shares program state between the editor and the application. Multiple participants reported issues in which the shared JavaScript runtime environment affected the editor:

> *(...) we were running something insanely heavy inside the browser which made it freeze up several times. And that then has the consequence that the entire development process freezes, too, because it's part of the application.* (P12, inq. 5).

This was mirrored by other user-programmers, too. One of these, P4, discussed how the editor would not break, but rather become incredibly slow, sometimes taking half a minute to register keystrokes. The inscribed plans of the artifact, whether intended or not, thus run counter to the notions of fluency and empowerment emphasized in visions of computational media.

5.5.3 Malleability is a double-edged sword

Malleability is one of the principles of contemporary computational media. It is, however, not purely a feature of the software, but is rather an opening that reveals itself in the meeting between user-programmer and software. It is a property of both the technical possibilities of the software and the user-programmer's perception of both the software and their own self-image and competencies.

One of the challenges in Codestrates v1 which led to the development of its second iteration was that the software was "one thing". By programming, one could edit, amplify, and break not only the document, but also the infrastructure that supports the editing itself. In effect, this could render the codestrate useless. diSessa's original vision of computational media emphasized the ability to reconfigure the medium, but in the case of Codestrates v1 this could have unintended consequences and breakdowns that user-programmers had a hard time recovering from.

One way to recover from this type of breakdown was to use the built-in versioning system. Codestrates v2, inheriting these capabilities from Webstrates, has a very granular versioning system down to each character input. While this theoretically provides for a very direct and engaging experience with the medium, the conflation between use and programming turned out to leak into the versioning:

> *We took the kind of bad choice that we also saved where every player['s camera feed] was, we also saved their position in a JSON fragment. And that had the effect that every time a player moved, a new version appeared. [...] We were up in like 30,000 or something [versions]. And that actually made it kind of useless to roll back.* (P4, inq. 5).

Since there is no inherent distinction between use and programming in the Codestrates v2 environment, at least from an infrastructural point of view, the versioning system proved useless for at least some participants in the study.

Although the editor and the application is both technically and conceptually separated in Codestrates v2, some issues still remain. First, since the technical capabilities for reprogramming and reconfiguring the computational medium have changed, it is no longer possible to, for instance, make desired changes to the editor itself. While this is arguably not an issue to most users in practice, it does represent a move away from the visions represented in diSessa's reconfigurable computational medium and the live reprogrammability of Smalltalk. The issue of discerning liveness of code in Codestrates v2 was experienced by a participant in the game design challenge who tried to dynamically alter the code on-the-fly but could not do so due to the execution model of JavaScript. Malleability thus leads to uncertain state and begs the question: What is being mediated? If both delegated code (the textual representation) and prescriptive code (the runtime) co-exists in the medium simultaneously, what exactly is the "object" of the mediation?

Unknown states can also be caused by other people acting on the artifact at a distance:

> *No, well, I think we would have had greater challenges if we had been working at the same time and not sitting together. I mean, because here we could at least, like "Are you running it now? Stop that!" [*laughs*] Sit and shout a little at each other or something like that. "Don't refresh, because it fucks up everything" [*laughs again*]* (P11, inq. 5).

Malleability as a principle marks an unsolved issue with regard to computational media. Unknown states represent ongoing acts of translation not immediately transparent to the user-programmer, while the collaborative nature of Codestrates v2 further erodes the flow of activities, demanding an adaptation of workflow and sociotechnical arrangements to support its idiosyncrasies.

There is, then, technological intentionality embedded in the artifact that forces or suggests particular ways of engaging with the medium. However, this intentionality is never absolute, as evidenced by P3 who managed to counteract the plans of the software.

> *For example, something that happened when we modified- we were doing some weird things. My UI was on top of the Cauldron UI, so the edit button wasn't reachable anymore. (...) So, I destroyed the whole ...and I knew, because I talked with some developers that you can go to the console and write `cauldronEditor`, and that will open it. But if was a normal user it's like, okay, I'm done* (P3, inq. 5).

P3 points to a central element of the relationship. By breaking the software, some level of trust is broken, too, and only by drawing on his existing literacy is he able to mend this relationship.

5.5.4 Roles and inscription

The technological intentionality inscribed into software clearly *does* something. For one thing, the intentionality creates a set of possible roles that an individual can fit into. This is not determined in advance but happens in the meeting between person and software in a complex manner, drawing also from existing self-perception: "A deep consequence of 'The medium is the message' is role and identity change." (Kay 2013a) This was discussed in chapter 3.

It is impossible to define all possible roles, but it is clear that computational media create a sort of hybrid between the role of user and the role of developer. An established branch of HCI research calls this hybrid relationship by various names such as "end-user developers" (Fischer, Giaccardi, et al. 2004) or "co-developers" (Mackay 1990), but these terms are not appropriate here. For one, they imply that the software is the final artifact that can be adapted for the individual's own wishes. Second, there is also the implication that the "user" is the primary category and "developer" a secondary one that can be appropriated as needed (or vice versa). While this might be true for some uses of computational media, I would argue that—for contemporary computational media—development *is* use and vice versa. I use the word *user-programmer* to illustrate that there exists a continuum of various roles between the activities of use and of programming.

Some people can easily inhabit different roles, switching between them fluently:

> *(...) one of the good things about this is that you can just put it out there and anyone can edit it. With basically no effort, or no other technical preparation.* (P5, inq. 5).

Although P5 emphasizes how effortless it is to edit the medium, others do not readily share this view. Another participant in the game design inquiry, despite already being familiar with programming, did not feel as comfortable doing so:

> *(...) this is the first time that I'm using that kind of complex programming tool. (...) The platform in itself wasn't that much self-explanatory. (...) But this platform for me is like, really, you have to know a little bit how to program* (P2, inq. 5).

This user-programmer did not readily experience a role for themselves in Codestrates v2:

> *As a non-programmer, let's say, as a non-computer science person (...) it [the artifact] should promise me that you don't have to be very much knowledgeable about programming.* (P2, inq. 5).

In fact, they are longing for the artifact to prescribe them a role that they feel comfortable with. The participant looked through the instructions, but ultimately did not finish the game design challenge, probably to a large degree due to Codestrates v2 configuring the user in a particular way:

> *And in the instructions, for instance, that PDF, some of the wordings, some of the terminology that you have used I wasn't even familiar with. (...) And then I realized that it's a very computer science terminology.* (P2, inq. 5).

This is somewhat in contrast with the nanoscientists from inquiry 3 who—despite not being computationally competent—were able to find a role for themselves in the use of the computational lab book. This is likely also caused by the extra "layer" of the interface that abstracts away the concrete code from their area of concern. For the nanoscientists, there seemed to be a different kind of role available that allowed them to use computational media and scripts without being able to program. The configuration of the user happens in the interface:

> *It's more complex than children's toys. But it's simpler than its equivalents in, like, for adult ones. But I am not knowledgeable about those examples for the adults (...) perhaps this is a little simpler. That's what I mean, the interface looks less ...intimidating.* (P2, inq. 5).

Despite the fact that P2 never managed to complete the challenge, her immediate impression of the software is that it casts a possible role between playful child and serious adult. As the computational medium is inscribed with particular intentionalities, some participants seemed to be quite deliberate in switching roles to adapt to the medium:

> *So you can say that it was quite important that we were sitting together [so we] could have these iterations between "now we do something," and "now we test something"* (P12, inq. 5).

The intentionality inscribed into an artifact further shapes the ethical aspects of its use, which goes beyond a discussion of an ethics of technology. Rather, it is an ethics of mediation and interaction. In line with other participants like P11 and P12, one participant in the game design study experienced this as the medium taking control, forcing their hand:

> *Why is it forcing me to think in HTML fragments? Why is it forcing me to think in code fragments? Why is it not simpler?* (P3, inq. 5).

5.5.5 Trust and alienation

The ethical qualities of mediation most clearly show themselves in terms of *trust* and *alienation* (or estrangement). We can distinguish between two modes of trust that come into view in my research on computational media. One of them we can call trust *in* the software, while the other can be termed trust *through* the software. In the case of the computational lab book, even if participants felt trust in the software itself, there were still issues regarding the mutability of both software *and* data. The conventional paper-based lab books (and to some degree their digital counterparts, e.g., Confluence) have certain properties that lend themselves well to research. Most prominent is probably their immutability. Several participants expressed concerns that since everything is editable in computational media, the conventional ways of resolving issues of, e.g., data provenance and intellectual ownership had potentially been eroded. While Webstrates' descendants all have a versioning system that could in theory solve this issue, not even experienced programmers trusted the versioning system. One group in inquiry 5 resorted to manually copying their code to a document in Google Docs to create a safe space for their code. Another participant in inquiry 5, P3, mentions how they missed git as a safeguard for their code.

These issues of trust are likely a consequence of the distribution and delegation inherent in contemporary computational media. Since the entire medium is persisted and self-contained at a distance, participants seemingly experienced a lack of control. N11 of inquiry 3 extended this trust issue to remote servers in general:

> *The thing is ...if one of these computers break down, maybe all the data is gonna ...if it's stored in some kind of server. That's a problem I can think of. Maybe it's stored somewhere else. Probably, yes, but I have no clue.* (N11, inq. 3).

This is supported by N10 who discussed the safety of having "local files". Probably, this is not a property of computational media *per se*, but a consequence of delegating data storage and computation to an (untrusted) actor, for instance the computational lab book. However, as computational media strive for the unification of data, computation, interaction, and interface, this problem clearly extends into their concrete work domain.

An interesting counterpoint came from another participant in the same study. It seems that trust in a person can be delegated into the material itself:

> *Interviewer: "(...) and the [script] is one of the scripts that the lab provides?" N10: "Yeah, it's one of [remote coworker's] scripts" Interviewer: "And you don't do PERL, I'm assuming, so they just gave it to you, and that's just what you're using?" N10: "Exactly."* (N10, inq. 3).

The scientist had previously explained how he does not understand the script, but the fact that it was written by a trusted coworker gives it legitimacy and fosters trust. Sometimes, the lack of trust is a product of past expectations not being fulfilled:

> *And that can be a little counter-intuitive because I think, like, well, my basic thought is "okay, I have this DOM that just synchronizes". And then all of a sudden, then it doesn't actually do that.* (P4, inq. 5).

This does not necessarily lead to breakdown and resignation, but it may subvert their expectations of the artifact based on previous experiences with Codestrates v1. Another user-programmer, in contrast, discussed this subversion of expectations in a more antagonistic way: "*Why to add this new thing? What is it giving me? It's like I'm fighting with the tool*" (P3, inq. 5).

P3 seemingly experienced some form of estrangement from the software. It is not just a matter of trust, it's a matter of fighting. The estrangement from the software and the subversion of expectations can potentially lead to negative self-concepts:

> *we lost a lot of time on the fact that it doesn't provide any help. I mean, if a variable is wrong or something. And also that we happened to create something stupid where we accidentally swallowed some of our errors (...) And any other editor would indeed catch that and say "hey", but there wasn't anything to help with that here. (...) I mean, we had error messages that didn't make sense at all.* (P4, inq. 5).

P4 mentions that they did something “stupid”. It is not clear if they extend this to mean that they *are* stupid, but they clearly expected an editor to be able to catch a syntax error. In this sense, the computational medium enters the uncanny valley of being an-IDE-but-not-really. In the more extreme cases, the alienation of software might even lead to resignation:

> *So now I’m following this tutorial on how to use this other script, and it doesn’t work and that’s it. I have no idea what to do. I don’t understand how it works at all.* (N11, inq. 3).

While this quote does not address computational media but scripts, it nonetheless perfectly illustrates the ultimate consequence of alienation from software: a dead end. This kind of computational disempowerment are elaborated in the next chapter. One way of overcoming resignation in the design of computational media is to allow for different levels of engagement.

5.5.6 Levels of abstraction

The lab book introduced a level of black boxing to the computational medium. By providing layers of abstraction and mediation, different modes of interaction was made possible. For instance, one could use the lab book without programming at all by dragging and dropping reified instruments onto the conceptual workflow represented in the medium. The object of work is, then, the RNA representation itself. However, it is also possible to alter the scripts embedded in the medium, moving into the territory of programming for computation. Finally, if one has the capabilities, it is possible to change the medium itself, reprogramming its interface and behavior to suit one’s needs. Through the composition of scientist, medium, scripts, and server, an empowered scientist emerges.

The extra abstractions provided in Cauldron and Codestrates v2, however, were not solely positively received:

> *Because in every other aspect, a traditional editor is better than the Codestrates editor (...) Because it’s adding abstractions that I don’t need.* (P3, inq. 5).

First, it is quite clear that P3 did not readily distinguish between the editor and the medium as a whole. Second, the introduction of fragments as a conceptual entity did not sit well with P3 who was familiar with both previous iterations of the *-strates family and conventional IDEs.

Another participant in inquiry 5 seemed to appreciate the easy setup provided by the black boxing of the infrastructure:

> *(...) for these, sort of, little prototypes because it’s just easily accessible and such. I think that is really nice, and also because I hate setting things up. I just don’t want to spend my life on that kind of things. And just the fact that it just runs.* (P11, inq. 5).

In fact, one could argue that this is an example of delegation of complexity. By abstracting away some infrastructural complexities for user-programmers, the creators of the medium have dealt with complexity and delegated these concerns to a central server. The central server is similarly responsible for the ease of sharing and the possibility of Codestrates to be multiple things at once.

5.5.7 Computational media as shared artifacts

Computational media are not shared artifacts. At least, not in the same way that, for instance, a whiteboard is. A whiteboard has an ontological unity: It is one thing, one area of concern, that inhabits the world in a certain way. In contrast, there is a particular multiplicity to Webstrates-based computational media. The artifact can be shared between multiple clients, yet some parts are not fully shared.

> *And at one point we were trying to use a JSON fragment to share things (...) but it didn’t work, and I don’t remember exactly for what. (...) and that’s not the model of Codestrates. You don’t share state, yes, you only share view.* (P3, inq. 5).

At the same time, having a shared object of work was hailed by the same participant: "*(...) you open a fragment, and you can see who is touching it, that was nice.*" (P3, inq. 5). This is an example of mediated collaborative action, perceived through hints in the interface. However, the same collaborative feature of Codestrates v2 turned out to be less than ideal for another user-programmer: "*(...) it would have been difficult to sit and develop in two places in the file at the same time*" (P4, inq. 5).

This way of remotely sharing a medium across the web can be conceptualized as a complex mediation in which multiple artifacts and humans are enrolled. Not only are there two or more people involved, but protocols, programming languages, servers, databases, web browsers, and a multitude of other things are necessary to cocreate the illusion that everything works seamlessly. Sometimes, though, the illusion was broken, and the distribution of activity across the network failed:

> *(...) and I mean, it's really cool that everything is collaborative and that everyone can change the code and such, but it also makes it necessary to coordinate in quite a tough way. I mean, it was actually almost necessary for us to be sitting in the same room.* (P4, inq. 5).

This is not by necessity a purely technical issue, nor is it a human error. Rather, the breakdowns happen in the multiple, complex mediation between remote user-programmers and the medium that is at once both one and more entities. Another participant argued against P4, saying that the benefits of Codestrates v2 are most clear when collaborating remotely. Evidently, the complex mediation is unstable, and its meaning is contested across user-programmers.

Only rarely was the difference between final and interim media verbalized by participants. One exception is P3 who liked,

> *the immediacy, how easy it is to share a link and let the other person experience the final thing ... that was much better in Codestrates than with traditional code.* (P3, inq. 5).

P3's comment further points to an interesting insight about the multistability of computational media. The same medium is, simultaneously, a work in progress and an artifact to be shared and used. There is no inherent ontological difference between them, as a simple click of a button enables the switch between two modalities that both coexist in the same conceptual and visual space. This is likely one of those characteristics that allows computational media to land in the uncanny valley between development and use.

5.5.8 The uncanny valley of computational media

The contemporary computational medium seems to sit squarely in the thing-but-not-quite domain. As people engage with the medium, they try to make sense of it through previous experience and mental models. The look and feel of Codestrates v2, for instance, lends itself to be interpreted as a conventional IDE:

> *It has significant enough functionality that I begin to be annoyed at "Oh, but why isn't this like VS Code?" Which it looks exactly like, but except for this and this.* (P1, inq. 5).

Clearly, the semiotic aspects of the meeting between user-programmer and software draws from both human expectations and the meaning inscribed into the system by the developers. Yet things do not seem to properly *fit*. The software's inscription leads to estrangement. Other concepts from Codestrates v2 present different issues, because they have no existing basis for their conceptual blend. For instance, both P11 and P12 in inquiry 5 emphasize their difficulty with making sense of the notion of fragments. P12 later draws an analogy to fragments from web technologies and the desktop metaphor: "*'Insert DOM element' (...) it's really just a folder with some files inside*" (P12, inq. 5).

This mental model seemingly works for P12, even if the actual code fragments are neither pure DOM elements, folders, nor files. The abstraction of code snippets into fragments both represents a conceptual breakdown and leads to a remapping of the software's inscription. In this way, P12 manages to align his mental model with the metaphor of the medium.

In the history of Webstrates towards Codestrates v2, I see a development from software-as-document towards software-as-IDE-but-not-quite. The different media are inscribed with particular narratives that seek to make sense of the system. Sometimes, though, the narratives fall short:

> *(...) we didn't quite know what autorun*[20] *does and what's, kind of, the connection between the editor and the whole, kind of, like- yeah, what's different about programming here and programming in other editors.* (P5, inq. 5).

Another user-programmer, P3, *did* rationally understand what the same autorun functionality does. It just does not align with his expectations of either a computational medium or an IDE, between which Codestrates v2 sits uncomfortably:

> *For example, all the auto on run ...completely bananas to me. Even if I understand the concept (...) It is out of my execution model* (P3, inq. 5).

The computational medium thus presents itself as something in between. Not quite one thing, but also not quite another. It is simultaneously a website, an infrastructure, a document, an application, a medium, a development environment, and a runtime environment. It both embodies an ontological multiplicity and semantic instability, switching between possible interpretations. The computational lab book, on the other hand, was never really problematized by the scientists in the same way. The seemingly uncanny valley of computational media is likely the cause of trust issues and semantic shifts. But what is the cause of this uncanniness? I would argue that many of the issues experienced by user-programmers of Codestrates v2 are caused by imaginaries. As experienced developers, the participants in the game design challenge drew from experience and expectations to understand the medium, but this sometimes led to user-programmers inscribing visions into the artifact. These are not "sociotechnical imaginaries" on a grand scale in the sense presented by Jasanoff and Kim (2015). Rather, they are impromptu imaginaries, created by user-programmers presented with a vision of computational media made real.

5.6 Computational media as technical imaginaries

Superimposing a vision onto an artifact leads to technical imaginaries. Some imaginaries are probably drawn from a collective, cultural memory, whereas other things are likely made up in action by the user-programmer. For instance, the issue of synchronizing state across clients presented some user-programmers with conceptual issues:

> *(...) because we probably thought, like, "ah, this is super easy, this part about synchronizing state, because it's practically solved for us"* (P4, inq. 5).

The self-contained, persistent, and collaborative nature of Webstrates, inherited by its descendants, represents novel ways of interacting with, and programming through, a web-based computational medium. P12 even stated that Codestrates v2 mimics "classic web development". Interestingly, it seems as if the vision of computational media might lead user-programmers to think that Codestrates v2 is "magical":

> *(...) it was also the fact that, when you're sitting in this tool where the application is the same as the editor and such, then you can really have this idea, or get this sort of feeling, that "oh, then you can just sit and do all kinds of things and do things and build things superfast and such" But when things are, like, asynchronous, and it's an event system and stuff, then you actually still have to think quite a lot if you want to end up with something*

[20]A built-in feature that toggles if a script fragment is automatically executed upon page load or only when pressing the "run" button in the editor

> *that works in the end. (...) You could get the sense that it's a bit more free play than it is.* (P4, inq. 5).

Several user-programmers also believed that Cauldron would let them access and edit the runtime environment *in situ*, even though the execution model of JavaScript does not allow this, or that an HTML fragment would synchronize with changes to the DOM[21]. This is in spite of the fact that the documentation clearly says otherwise. P10 and P12, in different cases, had a conceptual misunderstanding of how the runtime environment of JavaScript actually worked in Codestrates v2, despite both of them having ample web programming experience. Contemporary computational media therefore not only embody multiple possible interpretations as discussed in section 5.5.8, but also sit uneasily between ideal visions and concrete artifacts. What this means for questions of literacy and empowerment is discussed in the next chapter.

5.7 Conclusion

While the concept of a breakdown is commonly used to designate events in which a workflow is broken, I have pointed to several types of breakdown in the preceding chapter. Some of these are caused by deliberate actions from the user-programmer, while others are an effect of the ontological instability of the software. A third group is characterized by being caused by the collaborative and shared nature of contemporary computational media, while yet another comes from expectations not being met. In the following chapter, I address these computational crises, as I call them, and show how people engage in various types of recovery strategies, and how literacy and empowerment feeds into this dynamic.

A conventional postphenomenological analysis would probably draw upon Heidegger's hammer and argue that breakdowns are caused by the tool that stops working. Breakdowns are, then, unintended switches from working *through* to working *on* an artifact. However, an important characteristic of the mediation of computational media is that the *technology* and the *world* blend together. In "conventional" programming, as illustrated in section 5.2, the mediating technology is the IDE, the punch card, or the wires that are separate from the world, that is, the object being worked upon. In computational media, however, the mediating technology *is* the object itself. This has consequences for multiple elements of mediation such as the ethical, semiotic, and transformative aspects. In the realization of a vision of computational media, programming the medium has taken on recursive qualities. Codestrates v2 is a computational meta-medium that can, in principle, be used to create other forms of computational media. There is, however, a crucial difference between visions of a medium and the practical implementations of those visions. The coming chapter begins with an elaboration of computational media as both vision and reality before presenting the full view of computational literacy and software in action.

[21] Fragments, as the primary entities, are unidirectionally translated to the DOM

Interactive qualities, software mediation, computational literacy 6

In computing, turning the obvious
into the useful is a living definition
of the word "frustration"

Perlis (1982)

ꟸꟸꟸ

In the preceding chapters, I have answered my research question through an engagement with computational literacy in theory and practice, an elaborate and holistic model of technological mediation, and mediation analyses of programming modalities in various human-computer configurations. This chapter will tie these closer together through an empirically grounded investigation of their relationships. This provides an answer to my research question:

Research question How do the mediating qualities of software for programming contribute to the development of computational literacy?

Based on what I have presented in the previous chapters (chapters 3, 4, and 5), this chapter merges my research findings to provide a high-level analysis of how computational literacy is enacted and experienced. Or, in some cases, how it is not. The chapter is sectioned into the following points.

Computational media, visions, and literacy addresses the relationship between computational media as visions and as practical implementations and how this influences computational literacy development.

Computational culture is about the importance of sociocultural foundations and how they emerge in people's experience with software for programming.

Crises, disempowerment, and recovery strategies are those instances where things fall apart. Where people's skills or self-concepts are insufficient, or where the software causes everything to go awry. Here, more concretely, I provide a view of computational disempowerment where people are estranged by the software. In the same vein, I present an overview of the ways that people overcome such obstacles.

Computational empowerment and programming as a craft is about the contrasting instances in which empowerment is created and sustained through the mediating qualities of software for programming. The craftmanship of programming is in this vein addressed as a viable path towards literacy.

A taxonomy of the foundations for computational literacy draws together my research findings and provides a sociotechnical model of computational literacy that takes into account not only the formal definitions of literacy, but also the mediating experiences and those qualities deemed important in the software.

Finally, the chapter—and the dissertation—ends with a discussion of the broader implications of my research for two central fields: HCI and computing education.

ꟸꟸꟸ

6.1 Computational media, visions, and literacy

Computational media are, to reiterate, those software expressions that seek to narrow the gap between development and use, between application and document. In my research, I have studied how both non-technical researchers and experienced programmers have used, experimented with, and developed forms of computational literacies with these tools.

One immediately interesting finding relates to how the nanoscientists experienced the computational lab book. They quite quickly grasped the environment and became fluid with it. Part of this can likely be explained by its domain specificity—after all, it was tailored directly to them and modelled on their existing practices. In general, they seemed to struggle a lot less than with their previous amalgamation of scripts and loose environments, indicating an alignment in their self-concepts between self-conceptions, prescriptions, and expectations. By bringing the document and the application closer together in one computational medium, we also brought their tools closer to the object of work:

> *I think just all this opening and closing of the text editor, the terminal, the folders. Not have to save it in a folder and the script in the folder. And having a bunch of files.* (N8, inq. 3).

N8 vividly discussed the *messiness* of their existing workflows, tools, and materials. In terms of the distributional aspect of mediation, our computational lab book was able to *gather* rather than *distribute*. Or, more correctly, the lab book was distributing different aspects of the mediation than their existing digital tools. By moving away from the complex mediation (Bødker and Andersen 2005) of their previous technical assemblages and limiting the conceptual space of their work, the lab book would instead come to distribute and black box the executable code of their scripts. In short, the computational empowerment that the nanoscientists experienced was scaffolded by an abstraction from the concrete code, substituting it for reified *instruments* in a domain-specific computational medium. This was illustrated by N11, reminiscing about a previous experience: "*I tried to use [Overleaf], but sometimes it's also like …coding. And I don't understand. I just want a title.*" (N11, inq. 3).

N11's previous experiences with Overleaf were clearly disconfirmations of his abilities and knowledge. In contrast, the lab book is an example of a medium that allowed them to gain positive confirmations as computational users. Were the scientists empowered by our computational lab book? Most likely, yes, but not as an emancipatory literacy. They most likely did not develop the computational literacy to really understand, much less modify, the computational lab book. For them, the lab book's most important quality was by all accounts its ability to fluently black box the scripts that, while being integral to their work, were seen as nuisances. The fluency that, according to Kay (1984), is a key element of literacy, was evident only in their use of the medium as an application, not as a reconfigurable, dynamic, computational medium. This is not inherently bad, though. A good medium for thinking should ideally allow people with various skill levels and backgrounds to participate on equal foot (if not on equal level) (see, e.g., Badam et al. 2018). In this perspective, the nanoscientists can be considered *adaptively functionally literate*, if not critically literate or empowered. For the scientists, literacy meant being able to work with adequate *tools* for their *actual* work. These adequate tools are successful insofar as they are able to align the scientists' self-descriptions with their expectations of the tools. In contrast, an interviewee in the Codestrates v2 study discussed her feeling of agency and control as an integral part of feeling empowered and computationally literate:

> *(…) for me, the inspirational one for changing, having some alternatives, and have the freedom to change your practices, change your routines, is the key in that kind of activity.* (P2, inq. 5).

A related insight comes from my engagement with the students. While they used a Codestrates v2-based implementation of their weekly programming exercises, none of them seemed to realize that they were working in a computational medium. For them, the computational medium was just another application: "*It's really a confirmation from the application [Codestrates exercises] with those checkmarks. I miss that so much*" (S3 (pre), inq. 7). These findings suggest that a certain level of literacy is needed to even distinguish the medium from other applications, much less being able to understand the vision and its possibilities.

The vision of computational media is based on the idea that by providing mechanisms for easily switching between development and use—or even blurring those lines completely—people are empowered to change their own tools and develop computational literacy. In practice, it turns out, this is not enough. As pointed out by Kato and Shimakage (2020), one of the biggest pitfalls of "live programming environments" is the fact that they "still require the user to have the same level of expertise in programming as the original program developer" (ibid.). By live programming environments they specifically refer to environments such as Codestrates, although Codestrates was never inherently made for live reprogramming, nor is it a "literate programming environment" as the authors define it. Nevertheless, extrapolating their arguments to computational media more broadly, they bring forward an important point: The floors are too high. While the infrastructural aspects of the *-strates family are largely plug-and-play-oriented, to actually program the software requires a certain level of computational literacy. In contrast, the nanoscientists found the floors to be appropriately low, as their computational lab books were mainly being used as infrastructural artifacts: "*Being able to run any language and not have to set up an environment is super useful.*" (N11, inq. 3). Similarly, as we also experienced when constructing the lab book itself, another participant was able to see the benefit of such an ease of engagement: "*It worked really well as sort of a prototyping tool*" (P4, inq. 5).

Another issue is presented when turning the conversation from *computability* to *reprogrammability*. The findings from the retrospective study of *-strates agree with Kato and Shimakage (ibid.), and it became clear that "reprogrammable software written by others can be intractable, hard to reason about, and daunting to change" (Borowski, Fog, et al. 2022). There were also clear examples of participants being acutely aware of the need to reorient and rebuild their material intelligence into this unfamiliar medium, implicitly feeding back into their self-concept, e.g.,

> *(...) unfortunately, I didn't have time, too much time, to explore and prepare myself, get warmed up with the programming tools, etc. And in that sense I am not sure if you are interested in this level of user for your platform* (P2, inq. 5).

And while just giving access to and providing the possibility of manipulating source code does not guarantee that people actually possess the competencies to do so, a related issue arises around what *kind* of interaction is made possible. As sharply captured by P1 when using Codestrates v2: Reprogrammability is not the same as malleability. Providing access to the code is not enough. Malleability is an emergent property of technical possibilities, personal motivation, competencies and capabilities, and trust. Malleability is, therefore, the product of a particular mediation between human and computer.

There is a final point to be addressed regarding the potentials of computational media, particularly concerning the support for literacy development. Computational media are largely accompanied by specific visions, prescriptions of what and how software *ought* to be. The visions can be based on ideals related to pedagogy (e.g., Logo and Boxer), empowering users (Codestrates), or transparency (Eve) In that sense, all visions represent a break with what *is* and a look towards what *could be*. This unfortunately has consequences when it comes to computational literacy. First, computational media are often one-off software built by individuals or small groups to pursuit said vision. There is no guarantee of sustained maintenance and development of the system. As argued in chapter 3, a central aspect of computational literacy is *material intelligence* (diSessa 2001), the embodied knowledge-in-action and tool familiarity that people build individually and collectively over time. Computational media as one-off systems neither offer people the chance to reuse their existing material competencies nor to transfer any material intelligence to new systems. One participant in inquiry 5 phrased this criticism of the medium and vision in terms of investment:

> *For me, it's not a technical barrier (...) It was a value proposition. I will explain. What I mean is what is this editor giving me (...) In terms of anything. Compared to my current tools.* (P3, inq. 5).

In this statement we see how P3 addresses his frustration in terms of value and comparison. There is no new language to learn. The person already knew JavaScript and web programming. He was familiar with the vision and the previous iterations of the platform. His competencies as a software developer are, however, deeply connected to the tools that he uses, and one does not simply substitute for another tool without good reason. His computational literacy is both intellectual capacities and material intelligence. Even staying within the same software family, participants struggled to make sense of Codestrates v2 despite their experience with Codestrates v1. One could argue that the medium itself is not overly important, and that it is instead the underlying concepts that computational thinking addresses that are important for computational literacy (e.g., Wing 2006). However, my research suggests that computational literacy is in fact deeply entangled with and inseparable from the concrete materiality. In my study with experienced knitters (inquiry 8), for example, it became evident how they were able to transfer embodied material intelligence from knitting to programming. Of course, one could argue that the knitters had merely internalized computational concepts such as loops and variables, but my preliminary findings indicate that they were able to draw on their knitting experience precisely *because* it was an embodied material intelligence, not an abstract cognitive one.

Second, the vision of a computational medium not only defines a scope of possibilities and ideals of the system. It also carries an implicit promise of what the system can do. It carries particular expectations for the potential user-programmers. As such, the vision is capable of suspending even experienced programmers' understanding of software. In inquiry 5, we saw multiple times how participants expected the software to take care of, for instance, synchronization of code, even if they logically knew that the medium was incapable of that. Even those that had become familiar with the "old" vision were taken aback by the changes in Codestrates v2:

> *(...) overall this gave me a sort of sense that this set of tools, these Codestrates [v2] and (...) Cauldron, that they were designs with the standards of programmers in mind, rather than the vision of Webstrates* (P1, inq. 5).

As has become very clear in my inquiries of computational media, the fact remains that no matter the vision behind a computational medium, the actual software will almost always represent a distorted manifestation of the vision. This, in turn, can cause breakdowns when people's identities as, e.g., *programmers* are disturbed by the disconfirming experiences of unfulfilled expectations. For instance, the vision behind the Webstrates platform entailed the complete blurring of any lines between development and use: Whether you are developing or using the medium is an emergent property of your current activity. In practice, however, most people seem to need a conceptual and material distinction between those modalities. The transition from Webstrates to Codestrates v2 is characterized by an increasing compartmentalization of use and development.

The in-between materiality of computational media is a double-edged sword. The promise of the vision and the practicalities of software saw the emergence of two important challenges to literacy transfer. Some participants embraced the vision and were frustrated when it did not hold, while others held on to their existing competencies and were frustrated with the introduction of unfamiliar concepts and practices, for instance P3 who complained that "*(...) it wasn't compatible with Codestrates at the end, so we had a dead end there*" (P3, inq. 5). Interestingly, in my research computational media seemed most promising for those without prior knowledge of or interest in the vision like the students and the nanoscientists. This can likely be attributed to the importance of having and sustaining a certain level of computational self-concept among those who already see themselves as computationally literate. At least part of this finding can probably also be interpreted in the context of computational cultures which is addressed in the next section.

6.2 Computational culture

The following addresses a condition for fostering computational literacy: The sociocultural environments of people. Often, particularly in discussions of literacy as computational thinking, computational literacy is at risk of being seen as an individual feat. Even if we acknowledge the role of the material, computational literacy tends to be considered in isolation from the broader communities in which people take part. For instance, Kay's definition of literacy as fluency is the individual working on software (Kay 1984). diSessa, in contrast, points to the social as an important pillar of literacy (diSessa 2001). Computational culture is not limited to discussions of groupware and collaboration, either. Rather, computational cultures designate the broader social and cultural *conditions* for the development of literacy. These cultures can be bound to specific work domains (e.g., nanoscience), institutions (e.g., universities), or societies more broadly (e.g., emancipatory versus utilitarian views of computational literacy). Vee's perspective on literacy captures this quite well. By her definition, computational literacy grows from both individual competencies and broader sociocultural concerns (Vee 2013). For a historical example of the importance of computational cultures, consider the so-called Lisp curse that illustrates how social and technological issues feed into each other:

> Lisp is so powerful that problems which are technical issues in other programming languages are social issues in Lisp. (...) The moral of this story is that secondary and tertiary effects matter. Technology not only affects what we can do with respect to technological issues, it also affects our social behavior. This social behavior can loop back and affect the original technological issues under consideration. (Winestock 2017)

The Lisp curse addresses a computational culture for people who are already computationally literate, much in the same spirit as Kelty's concept of the *recursive publics* (Kelty 2005). However, even for those who are not as computationally literate, the computational cultures matter deeply, albeit differently. For instance, in the case of the nanoscientists from inquiry 3, the computational culture is very different from the ones found among professional programmers. One participant, for instance, stated that biomolecular nanoscientists with the computational literacy to produce or modify scripts are "pretty rare" (N10, inq. 3). Computational culture is not purely social; it is a sociomaterial phenomenon. It frames how people understand themselves as well as the computational artifacts in accord with values, norms, descriptions, and prescriptions. For instance, one of the nanoscientists managed to alter a UNIX-based script to make it run on his Windows machine which does, in fact, require some competencies and knowledge about operating systems and programming languages. Yet at the same time, he say that he does not have the computational skills to manage his workflows (N10, inq. 3). There are seemingly cultural norms at play which define what counts as real programming. We can speculate that the nanoscientists are so used to feeling disempowered with regard to their computational artifacts that they never feel computationally literate as a result, thus not being able to see themselves as "real" programmers in the process. This is at least partly backed up by N11, who argues that programming is a "whole other career". The consequence is that the scientists' computational culture is characterized by a dependency on cutting-edge computational tools such as scripts to do their job, yet without a tradition for training scientists to have these skills and competencies.

Whereas the participants in the game design study were largely all experienced programmers, the students and nanoscientists could more accurately be described as belonging to quasi-programming communities than part of a programming community proper: They are not programmers and do not see themselves as such on a cultural level. According to Tissenbaum, Sheldon, Seop, et al. (2017), computational identities develop from learners seeing themselves as programmers and feeling that they are engaged in the authentic practices of programmers. This is evidently not even happening for the programming students: "*I don't think I would ever see myself as a real programmer, even if I could do everything*" (S2 (pre), inq. 7).

Even the experienced programmers saw the need for more established spaces and communities for helping each other: "*So maybe some kind of community where people put their problems ...could be something that solves [technical issues]*" (P3, inq. 5). Likely, this can be attributed to them being used to having a community of peers (if not in physical form, then on StackOverflow). The students, in contrast, engaged in the formation of a classroom culture in which students shared successes and failures and helped each other: "*we ride together, we die together*" (S1 (pre), inq. 7). This is elaborated further in section 6.4.1. The computational culture of the nanoscientists seems to have stagnated in a resignation that they can do just enough (using whatever workarounds necessary), but will not be able to become truly literate:

> *And I think [head of research center] is really keen on getting everyone more up to speed on the basics, enough so they can troubleshoot the error on line 2765 or whatever it was (...) [head of research center] wants to put the scripts online on a server and go further with this*

> *database. And that's some advanced programming that we as basic users cannot do.* (N10, inq. 3).

This clash between their self-concepts and others' expectations is reflected by N8: "*Yeah, we don't code much in this lab. We should, but...*" (N8, inq. 3). The pedagogy of formal computing education often advocates the use-modify-create progression (Lee et al. 2011). Yet it seems as if this progression is not a given outside formal education and without an associated pedagogy and didactics. An interesting perspective to this comes from a study of Danish information workers, where it was found that more people know how to program than to modify existing software (Nouwens and Nylandsted Klokmose 2021). Section 6.8.1 discusses this further.

A much more concrete aspect of computational culture comes down to collaboration. When the programmers from the game design study had to work together in an unfamiliar environment, they needed to establish a culture of collaboration through in-group negotiation, as stated by P4. Similarly, P12 mentions how:

> *It's not quite in the same way as Google Docs where you can, like, have your own territory inside Google Docs. Here it is very much in a way where you are dependent upon one another. Otherwise, it won't compile, and it doesn't run.* (P12, inq. 5).

This is an interesting contrast to the more established structures that programmers are often used to, for instance versioning systems like git and Subversion. The findings presented in this section are indicative of the need to create sociotechnical cultures in order for computational literacy to develop and flourish. This is discussed further in section 6.7. In the following section we see what happens when people's computational competencies are insufficient and how they engage in various strategies to mend.

6.3 Computational crises

As argued by Hertzum and Hornbæk (2023), frustrations are still important challenges in HCI research. This section explores frustrations and crises as products of computational literacy, material intelligence, and the mediating qualities of software artifacts. As a starting point, we can, for instance, see frustrations grow from the lack of control over software as experienced by nanoscientists. N11, for example, says that,

> *if you change the script for some reason, and then it just keeps lying around there. With so many copies of these scripts around, you get confused in the end* (N3, inq. 3).

The accumulation of what could be called *residual artifacts* creates feelings of confusion and annoyance, as also evident from another scientist:

> *So I would get a folder with my sequences and that was really convenient [with the old script]. All my sequences wrapped up in one folder. (...) But with the new tool [script], they just give you too many subfolders, one folder for each. So the output is this really tedious...* (N8, inq. 3).

These frustrations are not due to computational media. In fact, they were largely solved by the computational lab book as shown in chapter 5. Nonetheless, the examples illustrate the importance of control which is addressed further in section 6.7. They also exemplify computational crises, albeit of a mild character. Sometimes, though, these frustrations escalate into feelings of disempowerment by not being able to reason with or work through those frustrations. When that happens, people seem to resort to a few distinct strategies. The following section presents key examples of computational disempowerment and an investigation of the strategies used to counter them.

6.3.1 Computational disempowerment

In the meeting between software and human, it seems as if some people blame themselves for their difficulties. One programmer, P2, who never finished the game design challenge and struggled to make sense of the unfamiliar environment, stated that,

> *there would have been quite a few things where I would just have thought "it's probably because I'm super stupid and don't know how to program"* (P2, inq. 5).

P2 created a new potential identity as "stupid" which could perhaps be attributed to low self-efficacy. She did not, however, blame the software and the design decisions behind it but herself. Several students similarly talked about the frustrations of programming:

> *I was ill all of last week and had to, for the first time, sit and do, like, a full week's curriculum and the assignment at home by myself. And it was* awful, *and I didn't understand it at all (...) But I think it was frustrating to feel that way, that I really did everything I could to try and understand it, but I couldn't (...)* (S2 (post), inq. 7).

This sense of being stuck was mirrored by another student who emphasized the lack of material intelligence: "*when you have to sit and do it, then it's just like, then it's just a black screen*" (S3 (pre), inq. 7). The lack of computational literacy turns programming into a difficult task that cannot be reasoned about. This lack of literacy similarly affected several of the nanoscientists who were also not capable of helping themselves:

> *So now I'm following this tutorial on how to use this other script, and it doesn't work and that's it. I have no idea what to do. I don't understand how it works, at all.* (N11, inq. 3).

Despite enrolling a more capable peer as delegated knowledge in the form of a tutorial, N11 is not capable of moving on. This feeling is echoed by N10 who tried looking at a faulty script, and "*that's about as far as I got. Because this means nothing to me*" (N10, inq. 3). N11 had issues with installing a virtual environment, was not able to figure out why it did not work, and described it vividly:

> *I don't really know why. So at the end, I think I managed to install it in some other way. But it was just painful. More pain.* (N11, inq. 3).

Even those participants with programming experience were challenged through a disruption of competencies. Several people from the game design study experienced this in a variety of ways, whether in the form of mental models, skills, or habits. P3 acknowledged that a lot of the issues that he experienced were "*actually minimal things*" (P3, inq. 5). Yet, he was still affected in the development process to such a degree that it outweighed the perceived benefits of a new system. One of P3's issues was, for instance, a mismatch between the software patterns that he knew and the affordances of Codestrates v2. This was echoed by P4:

> *(...) so the patterns that you'd normally use (...) I mean, you don't have a sort of basic architecture to fall back on.* (P4, inq. 5).

Especially debugging and reasoning about code proved difficult in an unfamiliar environment. A programmer, P6, explains how he experienced bugs that he could not pin down and were not sure whether they originated in their own code, in the Codestrates platform, or their externally loaded libraries. P12 similarly explained how the lack of line numbers presented significant challenges to existing skills, which was also discussed by P1: "*(...) and [Codestrates v2] completely removes my ability to use the debugging practices that I've learned as a semi-professional programmer*" (P1, inq. 5). Even in the computational lab book, at least one scientist was missing his usual terminal environment for debugging practices. Although not a programmer, he had seemingly developed a "sense" for errors: "*sometimes for [scripts], I understand because I faced [the errors] multiple times*" (N11, inq. 3).

6.4 Strategies for workarounds and recovery

Having presented a variety of computational crises, the following section presents workarounds, recovery strategies, or other ways of handling a lack of computational literacy in practice. As beautifully phrased by N8: "*But then I found a workaround. Because that's what you do.*" (N8, inq. 3). The strategies presented here are not equally distributed; some, like the enrollment of a more capable peer or giving up, are more prevalent than others. This is not a ranked list, either, but it serves to illustrate the ways that people make sense of and navigate the situation when challenged on their computational literacies by the material conditions.

6.4.1 More capable peer

One quite common strategy for all participants is the enrollment of a more capable peer (Vygotsky and Cole 1978, p. 86). For instance, P3 of the game design challenge enrolled not just any peer, but the actual developers of the medium:

> *Well, we were trying a lot of ideas on how to overcome this, sharing the model (...) we talked with the developers. They gave us a little code that was stripping the protected mode, so we could share* (P3, inq. 5).

Whereas P3 enrolled developers to teach him how to fish (as per the idiom), N11 instead asked the more capable peer to give him a fish. The nanoscientists had access to an HCP[1] cluster for heavy computations, yet this was not an option in practice for N11:

> *I ask [external collaborator] to do it. I just needed to do it once, and he just did it for me, so I don't really know how to do it. (Interviewer: "So you need to run something, you send it to a person, and they send the output back?") Yep. So it would be nice if I could do that myself.* (N11, inq. 3).

This not only happened when faced with HPC computation, but also when he was working with those scripts that are so important for their work:

> *(...) and it works for him, but it doesn't work for me. (...) So what I do is, I try to design the pattern that I think is going to fold in the way we expect, and I send it to him, and he runs it, and he's like "no". And then I go like, "ok" [*laughs*]* (N11, inq. 3).

In the original study, we found that the nanoscientists expressed a great deal of computational disempowerment. As their work depended on state-of-the-art computational tools without having the expertise to operate them properly, the scientists were often left to their own devices, unless they were able to enroll a more capable peer to help them. These were just two different ways of enrolling a peer. The first was knowledge sharing between programmers with different experiences; the second was the total delegation of programming to a peer. There are, of course, other ways. Enrollment of a more capable peer through textual documents is one such option:

> *[Coworker] wrote a basic tutorial on how to execute these scripts and what kind of output they like and that kind of stuff.* (N10, inq. 3).

Especially the students brought up the need for enrolling more capable peers. S4, for instance, called them her "safety nets" and even places them in a tiered hierarchy, between study group and class. S2 and S1 agreed with this and address the necessity of working with others to overcome being locked into "*a certain way of thinking*" (S2 (post), inq. 7). Interestingly, several students were highly aware of their particular situation as students and the ambiguous effect of enrolling a peer:

[1] High-performance computer

> *There is always someone you can get help to assignment from, but then I just don't feel that I learn it. I mean, at least I don't. I need to sit with it at home, and then I need to sit with it on my own* (S3 (post), inq. 7).

This was clearly a very loaded situation for the students, one that demanded balancing a more capable peer with the development of their own literacy. S4 explicitly discussed how it "sucks" to be the one who always needs help, while S3 said that it's a "*give and take situation*" (S3 (pre), inq. 7). A different experience was recounted by S2:

> *And I think I have slowly started to realize, like, it doesn't necessarily have to be that way. That it's acceptable to, like, need help with programming* (S2 (post), inq. 7).

All of these statements clearly indicate that the relationship between their self-concepts, their competencies, and their social worlds were complex and evolving. Only a few times did the students mention the teacher as a source of help, and then typically phrased as someone who is able to cast the runes: "*[The teacher] presses something (...) or puts the right semicolon, and then it works all of a sudden*" (S1 (pre), inq. 7).

Despite their intentions, the inclusion of a more capable peer is not always helpful. It could be the case that the capable peer is not that capable after all, as recounted by N8:

> *(...) and then sometimes it just gets stuck. Like this one. (...) I asked [coworker who wrote the script], and he doesn't know either what's going on.* (N8, inq. 3).

Or, it may be the case that just the physical presence of a more capable peer might present difficulties:

> *I mean, it made sense for me, I would easily be able to do it myself, but when I sort of sat there about to do it together with my dad, I was just like "is this right?"* (S2 (pre), inq. 7).

While this might also be attributed purely to their interpersonal relationship, it does point to a central finding. A more capable peer is often necessary to develop computational literacy, but the enrollment needs to happen in a way that aligns with the expectations, self-concept, and competencies of both parties.

6.4.2 Documentation

As a bespoke, one-off computational lab book, the nanoscientists did not have any written documentation of the system. Interestingly enough, none of them seemed to need it. This can perhaps be attributed to our instrumental interaction approach and the fluency of interaction that they experienced. Their existing patchwork of scripts and environments mainly consisted of custom software, so it might also be the case that documentation was simply never a part of their lifeworld.

The programmers in the game design study only rarely turned to the documentation for Codestrates v2. They were explicitly asked in the interviews, and only two of the 12 participants had used it (P3 and P9). This is in spite of the fact that all of them had programming experience. The behavior is therefore difficult to explain purely from an experience point of view. What is more likely is that many of them relied on their existing material intelligence, being able to reason their way through by drawing on competencies that they developed with other software.

6.4.3 Replaying steps

An interesting strategy was employed by a group in the game design challenge (P6 and P8), who took advantage of the fact that the computational medium allowed for cloning existing work.

> *So, at some point we produced a state that didn't run anymore and again didn't work, and we couldn't recover from that. (...) But what we did instead was just, since we're working*

> *with this tank game that was already out there, we just cloned another WEBSTRATE and then added our code, checked if this part works, and if it did, we added the next part. So basically replayed all our changes up until we ran into the bug again.* (P8, inq. 5).

While seemingly challenged on their competencies, they were able to reason their way out by using the affordances of the medium in a truly mediating experience of actions and feedback, i.e., an ongoing conversation with the material (Schön 1983).

6.4.4 Incorporation of other artifacts

Another strategy seems to be the incorporation of other artifacts, whether physical or virtual. For example, to be able to run the software needed for his workflow, N11 used two different laptops; one running macOS and another running Windows. Had N11 been more computationally literate, he could perhaps have managed to run a virtual operating system through, e.g., Wine, which he tried and could not:

> *So it's something I can't run on here unless I have a virtual machine and when I tried to install it, obviously problems as well. So, I don't know, it didn't work* (N11, inq. 3).

Yet his current workaround was seemingly adequate for his competencies. A related move was acted out by N8 who had a script that was running in Jupyter. When his Python installation stopped working, he switched to a different script that was less good rather than figuring out how to reinstall Python:

> *It's just convenience, just a matter of convenience. (...) you have to take some time and reinstall it. Or just use another tool. So far. I've just been using another tool.* (N8, inq. 3).

It seems that N8 felt confident enough in his own competencies to be able to reinstall an environment, and that it was more a question of convenience. He did not, however, seem similarly confident in his programming competencies:

> *Yes, if I had the skills [to edit scripts] that would be nice. But now, I either try a different script, or I hack the structure a bit. (...) [coworker] wrote different versions, so you can just go back a version, and it helps.* (N8, inq. 3).

The incorporation of other artifacts is perhaps a subcategory of a larger strategy of *adaption of work* to the concrete material circumstances. This can happen through the recruitment and use of known and trusted technologies N11, for example, enrolled other familiar artifacts for sharing files, even though he was aware of the potential issues that this could bring:

> *I think the issues usually arise because we try to share files in an old school kind of way, sending it by email. Or maybe using a USB. Or maybe uploading them in the lab book [Confluence], and then the other person kind of downloads them.* (N11, inq. 3).

Even more experienced programmers adapted this strategy. P9 and P10 enrolled Google Docs as an improvised versioning system, copying and pasting code as a safe backup. This was likely a consequence of the ethical aspects of the mediation—a lack of trust and confidence in the new and unfamiliar medium presented to them. A very interesting type of workaround happened when P5 enrolled the medium itself:

> *So what I ended up doing and becoming happy about, it was this [*shows two tabs with the same CODESTRATE*]. It then has other consequences, but I did that a lot. Then I used this "edit", where you can maintain your context over here, right? And then you can refresh over here, right?* (P12, inq. 5).

By playing along with the affordances of the medium, P5 managed to enroll a different modality of the same software by exploiting the shareability of CODESTRATES v2. These "other consequences" recounted by P12 were, among other, an erosion of the server's ability to count and manage clients. This hack is a very clear example of the multistability of computational media. There is not "one" artifact which might change over time (as in traditional software development) or a distributed process (as in GIT), but a strange, simultaneous doubling in which the artifact can literally be several things at once.

6.4.5 Fall back on manual workarounds

N10 manually added an inordinate amount of spaces to a textual molecule representation by hand:

> *I think this is one of those things where it's definitely not the most efficient way of doing this, but it'd take me longer to find a more efficient way, so this is what I do. (Interviewer: "Does anyone do something different in the lab?") I don't think so.* (N10, inq. 3).

He knew that it was inefficient, yet the lack of competencies and not having a more capable peer around made it necessary to get the work done in whatever way he could. In extension of this, N10 also had a faulty script that was supposed to check the validity of an RNA structure. When that did not work, he used the textual representation of the structure to reason his way around the structure, in effect playing computer in his head. Sometimes, however, there are seemingly no solutions to people's issues, and one way out is to simply give up.

6.4.6 Give up and move on

The examples presented here are not directly related to the act of programming but are more infrastructural. As argued by Vee (2013), this infrastructural aspect is also central to computational literacy. N8 had, for example, issues with a script that kept getting stuck in an infinite loop. Lacking the competencies to investigate, debug, and fix the problem, "*I just give up. This way, at least, it takes up less of my computer*" (N8, inq. 3). Further, N8, just like N11 mentioned earlier, had trouble with using multiple operating systems on the same machine. He was also not able to make it work, but instead of incorporating another machine, he once again coped by giving up:

> *(...) and I couldn't get it to work at that time. Probably because it's for, like, LINUX, and my WINDOWS just didn't. So I had to either do a virtual machine, get LINUX, and do it from there. But then I just gave up and didn't do it.* (N8, inq. 3).

Resignation seems to be a very common strategy, at least for those who are not part of a computational culture with peers that can help. People seemingly also adapt their work practices to account for their missing competencies like N10 did:

> *I tried to get it to work on my home computer and ran into some difficulties and decided to just give up and that it's better not to work from home* (N10, inq. 3).

6.4.7 "The ostrich"

Finally, this strategy is one I have humorously called "the ostrich". N11 had some problems with his computational environment, i.e., his PYTHON installation. And even though he knows that something is wrong with the environment, he argues that,

> *(...) every time I open [the terminal] it tries to do it, and it doesn't work. Everything seems to be working fine, so I'm not going to worry about it, but ...* (N11, inq. 3).

It is very likely that at least some of these findings are specific to computational media. Due to the bespoke nature of most computational media, there are often limited ways of recovering from breakdowns, as there might be a lack of documentation and no community of more capable peers, which, combined with the unfamiliarity of the medium, provides a recipe for breakdowns. Especially regarding the enrollment of more capable peers, it is interesting to note that these people are often researchers who are not permanently present. Researchers leaving the site and stopping maintenance of their artifacts is a well-known challenge in HCI (Hayes 2011; Taylor et al. 2013), and this problem is likely exacerbated by the dual roles of researchers as observers and researchers as more capable peers. As illustrated, there are plenty of times when the mediating qualities of software and people's competencies clash, leading to disempowerment and crises. In the following section, on the other hand, the opposite situations are elaborated.

6.5 Computational literacy in action

Sometimes, people's perceived and actual competencies align with their tasks and goals. In the following, I use the term computational empowerment in the sense of mastery, competence, and literacy. I am especially interested in those moments where people experience confirmations from the mediating experience that feed back into their self-concepts. While the knitters were not always aware of it, in first *decode and recreate* activity (inquiry 8), for instance, some participants realized that they could improve their knitting recipe by introducing loops, indicating *in situ* development of their competencies through confirmations from the environment. Unsurprisingly, many of these confirming experiences are associated with the students interviewed in inquiry 7. For example, S1 literally talked about a digital epiphany in which he suddenly realized both his own potential and the importance of programming as a literacy. This was supported by S2 who had previously used math software in high school: "*There were some things in Maple that suddenly made sense to me*" (S2 (pre), inq. 7). This is a sign of computational literacy as materially grounded competencies; not as abstract concepts but linked with concrete materiality and practice. We see a similar development happening with S4 who reflected on the material intelligence that she was in the process of developing:

> *I have already started to think a little about, like, what goes on behind a website that I'm visiting, or what exactly happens when I click this (...) "Well, I wonder if it's an array that comes here?"* (S4 (pre), inq. 7).

S4 was not thinking about abstract computational concepts, but a reified data structure that made sense to her. Empowerment is arguably, like computational literacy, tied to the concrete materiality. S2 similarly focused on the material conditions when stating how

> *Just this thing about the screen turning black [i.e., using the terminal], just that made it feel more real or something (...) Being able to access folders from our file systems. It became a little more real, I felt* (S2 (pre), inq. 7).

For computational literacy to develop, as argued in chapter 3, it requires the development of a being-in-the-world that is able to align competencies, practices, and material conditions. It must feel "authentic" as also emphasized by (Tissenbaum, Sheldon, Seop, et al. 2017). Empowerment is the condition of being able to align these aspects and *feel* competent. One participant in the knitting workshop, for example, expressed her surprise that programming was "really this simple". This was supported by other participants, e.g., "if you can knit you can also program", and the realization that—at least for them—the programming interface was more user-friendly than a knitting recipe.

Even if the students or the nanoscientists did not express any sense of being "true" programmers, the students were able to see themselves as becoming something else than their "normal" peers. For instance, S2 mentioned how she thought it was cool to be able to "*open people's eyes a little*" (S2 (post), inq. 7). Although not a true programmer by her own account, she had seemingly still taken on an identity of someone with more knowledge and skills than others, apparently beginning to develop an *emancipatory* computational literacy. An important aspect of computational literacy as Bildung and empowerment is the ability to reflect on one's own capabilities. S4 and S5, for example, emphasized the importance of being able to read code found online and reuse it for their own purposes. Here, the students conceptualized their literacy as digital literacy, the ability to find and decode information. It seems that for learners, different literacies (e.g., digital literacy, computer literacy, and computational literacy) are indistinguishable from each other. The same image was painted by nanoscientists who did not readily discuss the differences between, e.g., programming, setting up environments, and using other tools. To them, these activities are more like points on a scale than different modalities. It further became quite clear that the nanoscientists did not necessarily lack programming skills. Rather, they lacked the competencies to manage the complex *computational environments* necessary to run the scripts and software that they amassed to do their work.

One key aspect of computational literacy as mastery and empowerment is the ability to feel in control and express agency. A student, S1, had a previous image of programmers as being "nerdy". According to him, at some point he suddenly realized that you can "*make it what you want to*" (S1 (pre), inq. 7). Whereas S1's feeling of agency comes down to the larger goal and activity of programming, a similar expression of agency regarding the material conditions was expressed by a nanoscientist, N8. A script he used has clearly inscribed plans of use, but N8 ignored these due to a combination of personal preference, contextual knowledge, and, very likely, a need for ownership and agency:

> *Yeah, it's because [coworker] has this, like, "forbidden" written in, where you can't have duplicates of the same sequence. (...) like, RNA-wise, it makes sense. But for some of these, it doesn't. I kind of like having duplicates. So it's like, I put them in, anyway.* (N8, inq. 3).

An experienced programmer, on the other hand, felt a lack of control due to the idiosyncrasies of the computational medium, in particular the encapsulation and black boxing of parts of the medium:

> *But also the Cauldron sort of relative sophistication sucked a bunch of my work, a bunch of my labor into this context that I don't have full control over* (P1, inq. 5)).

Agency and empowerment as tied to literacy also emphasizes the importance of enjoyment. Not as a feeling of fun, but of being in control. As Kay (1984) argues, literacy is (also) about fluency. Going back to section 6.3, Hertzum and Hornbæk (2023) found that frustration in computing was correlated with task importance and lost time, but not with computer experience or computer self-efficacy. This indicates that fluency and control rather than expertise are important to avoid those feelings of frustration which can hinder literacy development.

6.5.1 Fluency and enjoyment

Among the participants involved in my research, enjoyment was attributed to feelings of competence and meaning-making in the activity and to fluency in the interaction itself. Regarding programming, a student mentioned how she prioritized the programming course over other courses, seemingly not due to any perceptions of its importance for future job prospects but simply because of the enjoyment of the activity: "*(...) maybe I'm spending a lot of time on [programming], but it makes sense and I actually think it's awesome*" (S5 (pre), inq. 7). The ability to find joy points to a key consideration for fostering computational literacy as will be unfolded in section 6.7.

Another perspective on fluency was directed towards the interactive qualities of computational media and the perceived fluency and smoothness of the internal processes of the artifact itself: "*Hey, this [lab book] went pretty quickly. Nice! Niiice!*" (N8, inq. 3). This was likewise stated by P5 in the game design study who "*personally quite like[s] this environment [Codestrates v2]*" (P5, inq. 5). Fluency can also be experienced as a mediating quality. Such is the case of P12 who liked the immediacy and liveness of the computational medium. In fact, not as an abstract knowledge of liveness but directly experiencing "*all kinds of styling being changed on the codestrate in the background*" (P12, inq. 5).

6.6 Embodiment and programming as a craft

Fluency, as a core element of literacy, can also be considered an embodied, interactive experience, for instance when drawing on a repertoire of keyboard shortcuts or using a computer mouse with a certain resolution and speed. This leads to a notion of programming as a craft, which highlights the development of a growing familiarity of techniques, tools, and knowledge about the domain. An example of this is the enrollment of design patterns to solve tasks as was emphasized by P3 and P4 (see section 6.3.1). While the embodiment of programming is a poor version of the full human capabilities, it nonetheless matters deeply. For example, it is quite clear that S2's burgeoning computational literacy depended on her embodied relationship with the material. Here, she reflected on a written multiple-choice test from the course:

> *I mean, we were used to, like, I can declare a variable, it is sort of in my fingers (...) But then when you get four different [options], where you are, like "oh, I don't remember if there needs to be– Oh, each one could all be correct". Yeah, that's really, really frustrating. (...) My fingers can just remember how to declare a variable, but it's not given that my brain can, when I just have to look at it.* (S2 (pre), inq. 7).

Related to this embodied programming experience are the keyboard shortcuts that become part of the physical vocabulary of actions in literacy.

> *Like, I don't know how many times I pressed `CTRL-S` to save. Or I don't know how many times I just click once to select the editor and the thing I was editing, it was still a previous tab, so I need to double-click.* (P3, inq. 5).

This is a case where Codestrates v2 went against the interaction language so deeply engrained in the bodily movements of P3. This was also reported by P8 who added that—apart from the built-in keyboard shortcuts—he was also missing "*some macros that I've made for myself when developing for the web*" (P8, inq. 5). This is an interesting point, as macros represent a series of actions being delegated to the system. By missing those macros, P8 was forced to manually perform those interactions previously distributed into the environment. This latter point is a reminder that people shape and navigate their environments, forming their competencies around those environments. The concept of affordances has been used extensively in HCI research to understand the room for action offered by software. In the case of P3 and P8, it likely has less to do with software affordances and more to do with embodied action. Or, as in the following case of P5, familiarity and trust:

> *Especially in the beginning, we really wanted to go out in an editor that we already know, and then write in that and then copy it to Codestrates* (P5, inq. 5).

In the KnitxCode study (inquiry 8) we sought to investigate this craft-oriented approach to computing. While results from the study are preliminary, it seems that treating computing as an embodied craft allows people to transfer competencies from other kinds of embodied craftsmanship. This is supported by Curzon et al. (2019, p. 521). One participant mentioned how, according to her, "a programmer would not be able to make sense of a knitting recipe", while she could now do both. She was feeling empowered through the ability to transfer her craft.

Our findings are generally in line with other research on craftsmanship-driven approaches to computing that emphasize mastery, aesthetics, and the emotional value of creation as driving forces (Buechley et al. 2008; Hyllegard, Ogle, and Diddi 2019). Further, by considering the craftsmanship-oriented aspects of computing, tacit knowledge becomes clear. P3 explains, for instance, the sort of blissful ignorance of those who have not developed materially grounded competencies: "*I think people without training, they wouldn't see the versioning problem.*" (P3, inq. 5). Interestingly, the same person later says that he does not "*see a person without programming knowledge using this tool*" (P3, inq. 5). This perspective is elaborated further in section 6.8.1.

Finally, by understanding computational literacy as a craft, we can see how this craftsmanship allows people to reflect-in-action and act accordingly, relying on competencies to manage unforeseen complexities. Being computationally literate means being fluent enough with the material, techniques, and tools to act *with*, not against, the environment and its idiosyncrasies. P11 explained how they accidentally had "autorun"[2] activated, and "everything crashed completely". This breakdown did not

[2]A feature of the web-based Codestrates v2 environment

affect them as they had the competencies to understand and work around the issue. Later, the same participant had issues with uploading assets in Codestrates v2[3] and was able to reflect in action and handle the unintended complexities:

> *I struggled a bit with that, and I was, like, "Damn it, did I delete that?" (...) right at the time, I was, like, "okay, then, workaround [*laughs*], and then just give them 1, 2, 3"* (P11, inq. 5).

Computational literacy in practice seems to rely not only on a material intelligence (i.e., thinking through and with the material), but also on a much more embodied and tacit set of competencies that can be expressed as craftsmanship. In chapter 3 I presented computational literacy in theory and practice. In the following section, I provide a taxonomy of the foundations of computational literacy based on my research.

6.7 A taxonomy of the foundations of computational literacy

Based on my results presented throughout the dissertation, this final section provides an interim taxonomy of the foundations of computational literacy. With inspiration from diSessa's three pillars of literacy and the definition presented in section 3.3.4, I here lay out a number of themes that any researcher on computational literacy ought to consider.

6.7.1 Material

The successful development of computational literacy requires adequate material conditions. While computational literacy is a product of multiple factors, having the appropriate material conditions goes a long way. I use the term material conditions rather than medium or software, as material conditions are more than just the individual software. They also include, for example, operating systems, development environments, and loosely organized files, not to speak of the physical materiality such as screens, keyboards, and other devices (which are outside the scope of this dissertation).

One of the findings from the retrospective trioethnography of the *-strates evolution was that the imperative paradigm of JavaScript was insufficient. The use of an imperative language demands that people play computers in their head. JavaScript in particular proved confusing in a web-based environment where everything shares state. It is important for computational literacy that the programming language provides the right level of abstraction and expressiveness. This is an ongoing discussion with plenty of viewpoints (see, e.g., Kelleher and Pausch 2005; Noone and Mooney 2018). An alternative is the use of declarative or reactive programming languages that allow for a more direct mediation. Similarly, the medium itself should allow for multiple levels of abstraction to allow for growing literacy and the enrollment of peers.

There are furthermore mediating qualities related to trust and conviviality that are necessary to be aware of. The ideals of absolute transparency (Eve Team n.d.) and naïve realism (diSessa and Abelson 1986) still deserve consideration. These should not only be clear during the actual interaction, but also in people's expectations of the software. The infrastructural aspect of setting up a development environment is similarly seen to be a potential obstacle to inexperienced programmers, so the medium must support low floors (as also emphasized by Resnick et al. (2009)), not only in terms of programming competencies but also for the circumstantial conditions. A final consideration is the support for transferring competencies from other domains, ideally leveraging people's existing material intelligence such as textual literacy or knitting experience.

6.7.2 Self-concept

For computational literacy to develop, it is important to support the development of an appropriate self-concept. If people cannot see themselves as being and acting competently in the world, no medium will alleviate those feelings. The self-concept must further be aligned with their competencies which necessitates the availability of appropriate identities for the individuals in question. A

[3]When deleting an asset (e.g., an image file) from Codestrates v2, the environment keeps a cached copy by the same name. Thus, when uploading a new asset with the same name, the old asset is loaded from cache

part of this identity development is also the ability to separate actions from identity like P4: "*I also just think it was because we did dumb stuff. But you just have to think carefully*" (P4, inq. 5). Just because you do "dumb stuff" does not mean you are, in fact, "dumb".

Literacy development requires confirmations that let learners know if they are on the right track. This is hardly a new insight, but computational artifacts are interesting since they allow for interactive and mediating experiences in which the artifacts themselves can talk back to learners:

> *In the very beginning it's important for me to get those "okay, am I on the right track?" I mean, like, yeah, there is a green checkmark (...) and also want to play with it a little more broadly and not necessarily have a single answer, but just kind of play with it, but it sort of demands that you are able to do something before it's fun* (S2 (post), inq. 7).

These confirmations help cement basic competencies that can then develop into a more playful engagement. It is likewise important for computational literacy development to let people take ownership, feel in control, and find motivation in projects that are close to their lifeworld.

> *I've never become really great at programming because I don't know how to do the things that others do, where they say "I will write my own [thing]", but a perfectly good alarm clock already exists, so why should I, like ...I'm not able to create a good one, and then I don't want to.* (24df, inquiry 5).

6.7.3 Computational culture

The development of computational literacy also necessitates a broader sociotechnical culture in which to embed this literacy. For instance, by providing access to more capable peers; by fostering a culture of not only gaining but maintaining competencies; and by providing people with enough basic literacy to be able to help themselves. In formal education, the continued maintenance of competencies can ideally be ensured by integrating computing into other courses, and providing a basic set of competencies can allow people to help themselves:

> *Yeah, but where now I think it's rare that something comes up where I don't understand it, like, if I don't understand it then I know where to look it up. So in that way you have gotten a foundation that you can build on. (...) Yeah, so I'd say that [that makes you] competent, and then that it's not so frightening, like, and just build on that if you want to* (S4 (post), inq. 7).

A different aspect of computational culture is establishing a space in which people can comfortably navigate and find help. Through the enrollment of more capable peers, people can leverage other, more experienced peers' competencies which can shape how people see themselves as part of their computational cultures:

> *And I think I have slowly started to realize, like, it doesn't necessarily have to be that way. That it's acceptable to, like, need help with programming* (S2 (post), inq. 7).

6.7.4 Beyond programming

Computational literacy can, in principle, develop from working in a fully self-contained environment such as Codestrates v2. In practice, though, the development of computational literacy also requires competencies beyond programming. Competencies in operating systems, programming environments, file systems, and other abstractions can be considered part of a more adaptive functional *computer literacy*. Managing two separate computers with different operating systems, as done by N11, functions as a workaround and allows work to be done. The same participant also had issues with some software: "*Maybe it's Linux, I have no clue*" (N11, inq. 3). However, these makeshift workarounds seemingly do not provide for an emancipatory literacy. File management and sharing is another aspect of computer literacy that appears to be foundational for computational literacy. A final point concerns programming languages and environments. It seems that people need basic literacy to even be able to differentiate programming languages. People like N11, who cannot distinguish shell scripts from Python, or the students who confuse HTML, CSS, and JavaScript lack the basic comprehension of the textuality of programming languages to even be able to move on without a more capable peer to help them.

The findings presented in this chapter represent various insights into computational literacy in practice. They are reminiscent of diSessa's concept of social niches (diSessa 2001, pp. 27–28) which—despite being written more than twenty years ago—is as relevant as ever. Computing educators and HCI researchers interested in new media and literacy should therefore pay particular attention to the implications for their fields. In this second-to-last section of the dissertation, I present these implications.

6.8 Implications

6.8.1 Implications for computing education

Although most of my research has not been conducted in formal educational settings, I would argue that my results are relevant for computing education research. The main reason for this is that formal education is likely where most people would encounter computational technologies beyond use-oriented practices. Even outside computer science, fields such as statistics, biology, economics, political science, and more recently humanities increasingly rely on computational tools as a core part of curricula and didactics.

The main implication that educators in computing education (and other fields) should care about is the importance of the material conditions. Even in computing education, there seems to be the idea that the materiality should be interchangeable. Of course, discussions abound about the best tools for teaching programming, but the underlying notion often seems to be that the tools are less important than developing abstract concept knowledge and associated cognitive effects. This is evident in the numerous calls for computational thinking efforts in all levels of education (Wing 2006; Denning 2017). The right materiality is, of course, no substitute for pedagogy, didactics, and engaged teachers, but we absolutely cannot ignore the role that materiality plays in the development of computational literacy. This was already pointed out in Waltz' criticism of the sociocentrism of educational research (Waltz 2004). Computational literacy develops in an intimate, embodied, and mediated relationship with the concrete materiality.

Second, contemporary computational media seems a particularly promising software paradigm for teaching. The ease of sharing assignments, the possibilities for collaboration, and the low infrastructural floors provide a useful learning environment in the vein of the PLATO system from the '60s (Kaiser 2023). P6 captured this quite accurately:

> *(...) for example, in some teaching scenario, I think it would be very useful. For, kind of, people to follow or even collaborate together on, kind of, coding. Yeah, and also afterwards the student will still have the link and have the access. (...) But instead, you can actually focus on what they're trying to say and why they're doing this. So you actually have a deeper understanding of the code. (...) Especially I think, for someone who is new to web programming, it will be very useful, 'cause they don't need to learn, like, that much about different setups and shortcuts first.* (P6, inq. 5).

Third, computing education must keep in mind the identities and self-concepts of students, especially non-CS students who are not necessarily going to work as software developers or similar computing-centric professions. There must be appropriate identities available to them, and their particular identities must be in alignment with their expectations, the material conditions, and the kinds of communities of practice that they are becoming members of. A focus on student experiences in the form of confirmations seems important, as does the formation of and support for computational cultures in which students can interpret and appropriate programming under different guises such as creativity, fun, motivation, drive, and societal benefit.

6.8.2 Implications for HCI

My research also brings forward a number of implications for HCI research. One of them is directed towards the way we as researchers relate to previous projects. Constructive HCI research on artifacts for empowerment and sustainable change often focuses on short-term evaluation and novelty (Balestrini, Rogers, and P. Marshall 2015). One solution is, of course, to conduct more long-term in-the-wild engagement. Another counterweight to the focus on novelty is to consider research artifacts not as single inquiries, but as a series of continued engagements that can reasoned with as a processual phenomenon. As part of my research, the group of computational media collectively known as the *-strates family in my dissertation was reviewed through a retrospective trioethnographic lens. Simply put, I would argue that looking back at former research projects and assessing them in an honest light enables us to be better researchers tomorrow. By reviewing our contributions as a continuous learning process in the spirit of participatory design (Bødker, Dindler, et al. 2022), we can pay our due respects to previous prototypes and the people who were involved in their construction and evaluation.

The second implication concerns computational media more specifically. Despite being a subject of research for at least forty years (and drawing their visions even further back), their concrete implementations are not yet realized in any full capacity. In their vision of meta-design, Fischer and Giaccardi (2006) argue that the problem of computational media (what they call "interactive programming environments") is the Turing tar pit:

> These tools provide the ultimate level of openness and flexibility (...) they by themselves are insufficient for meta-design. The essential problem with these systems is that they provide the incorrect level of representation for most problems (ibid.)

Combined with my previous findings (see section 6.1) that people need the same level of competencies as the developers of the medium, and that people have a hard time reasoning about and working with other people's code, future research on computational media *must* address these challenges to support any kind of general uptake and sustainability. Ways of countering these issues are, for instance, to allow people to engage with the medium on different levels of abstraction or by providing alternative programming paradigms such as reactive or declarative languages. An inspirational framework for concrete design dimensions of computational media can be found in Jakubovic, Edwards, and Petricek (2023).

Third, HCI research can benefit from more engagement with fields like technology of philosophy and software studies. While their epistemological and methodological foundations vary, their engagements with digital artifacts provide a fruitful supplement to HCI's existing repertoire. For instance, the postphenomenological view of technological mediation benefits from HCI's empirical methods, while HCI can benefit from the larger discussions of what technology does for the human condition. In chapter 4 I have sought to make the coupling between mediation and interaction which are essentially two conceptualizations of the human-computer relationship that inform each other well. Finally, along that line I would encourage my fellow HCI researchers to be mindful of how we talk about computational artifacts and the people who use them. By referring to the artifacts as "tools" and the people as "users", which is not atypical for HCI research, we implicitly paint an image of the user (who is not a developer) as a goal-oriented person who just needs the right tool to do the task that they already set out to do (Fuchsberger, Murer, and Tscheligi 2013). Where does that leave chance, serendipity, and the mutual relationship between person and artifact?

Conclusion 7

In the preceding six chapters, I have answered the following research question:

Research question How do the mediating qualities of software for programming contribute to the development of computational literacy?

I have presented computational literacy as a set of competencies that is oriented beyond the individual itself. Computational literacy is different from computer literacy and digital literacy due to the focus on the actual computational capabilities of the medium rather than the ability to use and navigate a computer. This qualifies computational literacy to have the potential for emancipation, the empowering and Bildung-oriented competencies that allow people to not only fill a role, but to reflect and critique. Computational literacy is also about fluency and the development of material intelligence. This is contrasted with the new school of computational thinking as a mainly cognitive phenomenon and thus provides an alternative to the dominant paradigm of learning and doing. To better understand the components that lead to identity formation and, ultimately, literacy, I have created a model of self-concept which provides a series of subcomponents necessary to take into consideration when understanding computational literacy development. Split into descriptions, prescriptions, and expectations, the model also emphasizes the importance of experiences, confirmations, and disconfirmations in such a development.

As computational literacy develops in close relationship with the material, it is necessary to have an understanding of this relationship. To this end, I have provided an integrated model of technological mediation that bridges theory from HCI, science and technology studies, and the philosophy of technology. A particular contribution is the interdisciplinary approach that combines use-oriented aspects with, e.g., emancipation, ethical considerations, and user inscriptions. Following the development of this integrated model of mediation, my analyses of selected human-computer configurations showed how the mediating qualities of programming can be understood through degrees of mediation, abstraction, and translation. In my work on computational media, I have examined historical visions and contrasted them with contemporary visions and implementations. Part of this is a presentation and critique of literate computing for computational media in which I have shown that literate computing did not fulfill its promise for computational literacy development. This served as the backdrop for the subsequent analysis of computational media in action. Here, I showed how the mediating qualities are realized in practice through a number of themes such as use, development, semantic shifts, malleability, roles, trust, abstractions, and shared artifacts. The analysis concludes that a significant challenge for computational media is that they largely fall in between visions, imaginations, and practicalities. This has a number of consequences for computational literacy. First, it seemed that a computational medium was able to provide an appropriate level of abstraction for people who are still in the process of developing their competencies. On the other hand, more experienced programmers seemed to get caught in the gap between the visions of the medium and the practicalities of the actual implementation.

Another foundation for the development of computational literacy is the sociotechnical cultures in which people find themselves, what I called computational cultures. This points to the necessity of being embedded in communities of norms, values, peers, and potential identities for the formation of computational literacy. An important finding is that not all communities are equal: Nanoscientists and humanities students do not align themselves with professional programmers but create and belong to what can be called quasi-programmatic cultures.

As people's competencies are sometimes insufficient, they experience various forms of crises that might lead to computational disempowerment. That can be due to, e.g., not having the competencies to help themselves or having their existing competencies challenged by unfamiliar media. To counteract those experiences of disempowerment, people engage in a variety of strategies to regain control of their material conditions. One of the more used strategies is the enrollment of other actors, whether human or technical. A more capable peer is one example of this, while the enrollment of known and trusted artifacts is another. A different group of strategies is giving up or deciding to ignore potential issues.

Other times, however, people's competencies are sufficient, leading to second-order effects such as computational epiphanies, in which previous experiences suddenly "make sense" to them, or feelings of control and mastery. Literacy might also be experienced as a sense of fluency and enjoyment in working with their material. This latter point can in some circumstances be attributed to the embodiment of programming—the intimate bodily engagement with the medium. Considering programming and computing as crafts, as embodied and material activities, seems to be a promising alternative to the cognitive view of computing advocated by the new school of computational thinking initiatives. My research indicates that computing education research can benefit from this reorientation from *thinking* towards *doing* in materially grounded experiences.

The development of computational literacy must therefore be conceptualized with serious consideration for the material, social, and cognitive pillars. Part of this perspective consists of taking into account the mediating qualities of software for programming and how they shape, create, hinder, and support various forms of engagement between user-programmers and their material conditions. Finally, computational literacy is a complex phenomenon which requires alignment between people's self-concepts, their embodied and material competencies, and their outlook on the world in terms of empowerment, agency, and identity formation. This demands the support of communities, computing cultures, and adequate material conditions which can support emancipatory computational literacy. After all, everyone should be able to cast and parse the runes of contemporary life.

Bibliography 7

Aagaard, Jesper (2017). “Introducing postphenomenological research: a brief and selective sketch of phenomenological research methods”. In: *International Journal of Qualitative Studies in Education* 30.6, pp. 519–533. DOI: 10.1080/09518398.2016.1263884.

Aagaard, Jesper and Noomi Matthiesen (2016). “Methods of materiality: participant observation and qualitative research in psychology”. In: *Qualitative Research in Psychology* 13.1, pp. 33–46. DOI: 10.1080/14780887.2015.1090510.

Abbate, J. (2018). “Code Switch: Alternative Visions of Computer Expertise as Empowerment from the 1960s to the 2010s.” In: *Technology and Culture* 59.4, pp. 134–159. DOI: 10.1353/tech.2018.0152.

Aho, Alfred V. (2011). “Ubiquity symposium: Computation and Computational Thinking”. In: *Ubiquity* 2011 (January). DOI: 10.1145/1922681.1922682.

Andersen, Peter Bøgh (1991). “Computer semiotics”. In: *Scandinavian Journal of Information Systems* 3.1.

— (2001). “What Semiotics can and cannot do for HCI”. In: *Knowledge-Based Systems* 14.8, pp. 419–424. DOI: 10.1016/S0950-7051(01)00134-4.

Anderson, Nate (Nov. 2018). *First encounter: COMPUTE! magazine and its glorious, tedious type-in code.* Ars Technica. URL: https://arstechnica.com/staff/2018/11/first-encounter-compute-magazine-and-its-glorious-tedious-type-in-code/ (visited on 12/22/2022).

Anderson, Terry and Julie Shattuck (2012). “Design-Based Research: A Decade of Progress in Education Research?” In: *Educational Researcher* 41.1, pp. 16–25. DOI: 10.3102/0013189X11428813.

Antonsen, Kristian B., Michel Beaudouin-Lafon, James Eagan, Clemens Nylandsted Klokmose, Wendy E. Mackay, and Roman Rädle (2017). “Webstrates for the Future Web?” In: *ProWeb 2017 - Programming Technology for the Future Web.* URL: https://hal.inria.fr/hal-01614236.

Auerbach, David (May 9, 2014). “The Oldest Rivalry in Computing”. In: *Slate.* URL: https://slate.com/technology/2014/05/oldest-software-rivalry-emacs-and-vi-two-text-editors-used-by-programmers.html (visited on 12/22/2022).

Badam, Sriram Karthik, Andreas Mathisen, Roman Rädle, Clemens Nylandsted Klokmose, and Niklas Elmqvist (2018). “Vistrates: A Component Model for Ubiquitous Analytics”. In: *IEEE Transactions on Visualization and Computer Graphics* 25.1, pp. 586–596. DOI: 10.1109/TVCG.2018.2865144.

Balestrini, Mara, Yvonne Rogers, and Paul Marshall (2015). “Civically engaged HCI: tensions between novelty and social impact”. In: *Proceedings of the 2015 British HCI Conference.* Lincoln, Lincolnshire, United Kingdom: ACM, pp. 35–36. DOI: 10.1145/2783446.2783590.

Bannon, Liam, Susanne Bødker, et al. (1991). “Beyond the interface: Encountering artifacts in use”. In: *Designing interaction: Psychology at the human-computer interface*, pp. 227–253.

Barnes, Susan B. (2007). “Alan Kay: Transforming the Computer into a Communication Medium”. In: *IEEE Annals of the History of Computing* 29.2, pp. 18–30. DOI: 10.1109/MAHC.2007.17.

Barnett, Ronald (2009). “Knowing and becoming in the higher education curriculum”. In: *Studies in Higher Education* 34.4, pp. 429–440. DOI: 10.1080/03075070902771978.

Beaudouin-Lafon, Michel (2000). “Instrumental interaction”. In: *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI ’00.* ACM Press. DOI: 10.1145/332040.332473.

Beaudouin-Lafon, Michel, Susanne Bødker, and Wendy E. Mackay (2021). “Generative Theories of Interaction”. In: *ACM Transactions on Computer-Human Interaction* 28.6, pp. 1–54. DOI: 10.1145/3468505.

Bergin, T.J. (2006). “The Proliferation and Consolidation of Word Processing Software: 1985-1995”. In: *IEEE Annals of the History of Computing* 28.4, pp. 48–63. DOI: 10.1109/MAHC.2006.77.

Berners-Lee, Tim (1996). “The World Wide Web-past, present and future”. In: *Journal of Digital information* 1.1.

Berry, David M. (2011). “What Is Code?” In: *The Philosophy of Software*. London: Palgrave Macmillan UK, pp. 29–63. isbn: 978-1-137-49027-8. doi: 10.1057/9780230306479_2.

Bertelsen, Olav W. (2004). “Transparency by Tertiary Artefactness”. In: *Aesthetic Approaches to Human-Computer Interaction*.

Bertelsen, Olav W. and Susanne Bødker (2003). “Activity theory”. In: *HCI models, theories, and frameworks: Toward a multidisciplinary science*, pp. 291–324.

Bertelsen, Olav W. and Marx Wartofsky (1999). “Mediation and Heterogeneity in Design”. In: *Social Thinking–Software Practice*, p. 16.

Bevir, Mark (2008). “What is Genealogy?” In: *Journal of the Philosophy of History* 2.3, pp. 263–275. doi: 10.1163/187226308X335958.

Bier, Eric A. (1991). “EmbeddedButtons: documents as user interfaces”. In: *Proceedings of the 4th annual ACM symposium on User interface software and technology - UIST ’91*. Hilton Head, South Carolina, United States: ACM Press, pp. 45–53. doi: 10.1145/120782.120787.

Bjørndahl, Johanne Stege, Riccardo Fusaroli, Svend Østergaard, and Kristian Tylén (2014). “Thinking together with material representations: Joint epistemic actions in creative problem solving”. In: *Cognitive Semiotics* 7.1, pp. 103–123. doi: 10.1515/cogsem-2014-0006.

Bødker, Susanne (2006). “When second wave HCI meets third wave challenges”. In: *Proceedings of the 4th Nordic conference on Human-computer interaction changing roles - NordiCHI ’06*. Oslo, Norway: ACM Press, pp. 1–8. doi: 10.1145/1182475.1182476.

Bødker, Susanne and Peter Bøgh Andersen (2005). “Complex Mediation”. In: *Human-Computer Interaction* 20.4, pp. 353–402. doi: 10.1207/s15327051hci2004_1.

Bødker, Susanne, Christian Dindler, Ole S. Iversen, and Rachel C. Smith (2022). *Participatory Design*. Synthesis Lectures on Human-Centered Informatics. Cham.: Springer International Publishing. isbn: 978-3-031-01107-8. doi: 10.1007/978-3-031-02235-7.

Bødker, Susanne and Clemens Nylandsted Klokmose (2012). “Dynamics in artifact ecologies”. In: *Proceedings of the 7th Nordic Conference on Human-Computer Interaction Making Sense Through Design - NordiCHI ’12*. Copenhagen, Denmark: ACM Press, p. 448. doi: 10.1145/2399016.2399085.

Borenstein, Nathaniel S. and James Gosling (1988). “UNIX Emacs: a retrospective (lessons for flexible system design)”. In: *Proceedings of the 1st annual ACM SIGGRAPH symposium on User Interface Software*. Alberta, Canada: ACM, pp. 95–101. doi: 10.1145/62402.62417.

Borowski, Marcel, Bjarke Vognstrup Fog, Carla F. Griggio, James R. Eagan, and Clemens Nylandsted Klokmose (2022). “Between Principle and Pragmatism: Reflections on Prototyping Computational Media with Webstrates”. In: *ACM Transactions on Computer-Human Interaction*. doi: 10.1145/3569895.

Borowski, Marcel, Janus Bager Kristensen, Rolf Bagge, and Clemens Nylandsted Klokmose (2021). *Codestrates v2: A Development Platform for Webstrates*. Tech. rep. Aarhus University. url: https://pure.au.dk/portal/en/publications/codestrates-v2-a-development-platform-for-webstrates(66e1d4d9-27da-4f6b-85b3-19b0993caf22).html.

Borowski, Marcel and Ida Larsen-Ledet (2021). “Lessons Learned from Using Reprogrammable Prototypes with End-User Developers”. In: *End-User Development*. Ed. by Daniela Fogli, Daniel Tetteroo, Barbara Rita Barricelli, Simone Borsci, Panos Markopoulos, and George A. Papadopoulos. Cham.: Springer International Publishing, pp. 136–152. doi: 10.1007/978-3-030-79840-6_9.

Borowski, Marcel, Roman Rädle, and Clemens Nylandsted Klokmose (2018). “Codestrate Packages”. In: *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM. doi: 10.1145/3170427.3188563.

Bouvin, Niels Olof and Clemens Nylandsted Klokmose (2016). “Classical Hypermedia Virtues on the Web with Webstrates”. In: *Proceedings of the 27th ACM Conference on Hypertext and Social Media*. Halifax, Nova Scotia, Canada: ACM, pp. 207–212. doi: 10.1145/2914586.2914622.

Bowen, Glenn A. (2006). “Grounded theory and sensitizing concepts”. In: *International journal of qualitative methods* 5.3, pp. 12–23.

Braun, Virginia and Victoria Clarke (2006). “Using Thematic Analysis in Psychology”. In: *Qualitative Research in Psychology* 3.2, pp. 77–101.

Brennan, Karen and Mitchel Resnick (2012). “New frameworks for studying and assessing the development of computational thinking”. In: *Proceedings of the 2012 annual meeting of the American educational research association, Vancouver, Canada.*

Brown, Trent (2016). “Sustainability as Empty Signifier: Its Rise, Fall, and Radical Potential”. In: *Antipode* 48.1, pp. 115–133. DOI: 10.1111/anti.12164.

Bruce, Christine Susan (2004). “Information Literacy as a Catalyst for Educational Change: A Background Paper”. In: *Keynote address, for “Lifelong Learning: Whose responsibility and what is your contribution?”, the 3rd International Lifelong Learning Conference, Yeppoon, 13-16 June 2004.*

Buechley, Leah, Mike Eisenberg, Jaime Catchen, and Ali Crockett (2008). “The LilyPad Arduino: using computational textiles to investigate engagement, aesthetics, and diversity in computer science education”. In: *Proceeding of the twenty-sixth annual CHI conference on Human factors in computing systems - CHI ’08.* Florence, Italy: ACM Press, p. 423. DOI: 10.1145/1357054.1357123.

Bundsgaard, Jeppe (2017). *Digital dannelse.* Pædagogisk rækkevidde 1. Aarhus: Aarhus Universitetsforlag. 69 pp. ISBN: 978-87-7184-246-3.

Burns, Ryan and Grace Wark (2020). “Where’s the database in digital ethnography? Exploring database ethnography for open data research”. In: *Qualitative Research* 20.5, pp. 598–616. DOI: 10.1177/1468794119885040.

Bush, Vannevar et al. (1945). “As we may think”. In: *The atlantic monthly* 176.1, pp. 101–108.

Callon, Michel (1984). “Some Elements of a Sociology of Translation: Domestication of the Scallops and the Fishermen of St Brieuc Bay”. In: *The Sociological Review* 32.1, pp. 196–233. DOI: 10.1111/j.1467-954X.1984.tb00113.x.

Campbell-Kelly, Martin, William Aspray, Nathan Ensmenger, and Jeffrey R. Yost (2018). *Computer: A History of the Information Machine.* 3rd ed. Routledge. ISBN: 978-0-429-49537-3. DOI: 10.4324/9780429495373.

Carrington, Victoria (2018). “The changing landscape of literacies: Big data and algorithms”. In: *Digital Culture and Education* 10.1, pp. 67–76. URL: https://ueaeprints.uea.ac.uk/id/eprint/68380.

Christensen, Kasper Skov, Mikkel Hjorth, Ole Sejer Iversen, and Paulo Blikstein (2016). “Towards a formal assessment of design literacy: Analyzing K-12 students’ stance towards inquiry”. In: *Design Studies* 46, pp. 125–151. DOI: 10.1016/j.destud.2016.05.002.

Coiro, Julie, ed. (2008). *Handbook of research on new literacies.* New York: Lawrence Erlbaum Associates/Taylor & Francis Group. 1367 pp. ISBN: 978-0-8058-5651-4.

Creswell, John W. (2014). *Research Design.* SAGE. 273 pp. ISBN: 978-1-4522-2609-5.

Curzon, Paul, Tim Bell, Jane Waite, and Mark Dorling (2019). “Computational Thinking”. In: *The Cambridge Handbook of Computing Education Research.* Cambridge University Press, pp. 513–546. ISBN: 978-1-108-65455-5.

Dalsgaard, Peter (2010). “Research in and through design: an interaction design research approach”. In: *Proceedings of the 22nd Conference of the Computer-Human Interaction Special Interest Group of Australia on Computer-Human Interaction - OZCHI ’10.* Brisbane, Australia: ACM Press, p. 200. DOI: 10.1145/1952222.1952265.

Davis, Jenny L. and James B. Chouinard (2016). “Theorizing Affordances: From Request to Refuse”. In: *Bulletin of Science, Technology & Society* 36.4, pp. 241–248. DOI: 10.1177/0270467617714944.

Denning, Peter J. (2017). “Remaining trouble spots with computational thinking”. In: *Communications of the ACM* 60.6, pp. 33–39. DOI: 10.1145/2998438.

Dirckinck-Holmfeld, Lone, Jørgen Lerche Nielsen, and Thomas W. Webb (1988). “Almendannelse og informationsteknologisk fantasi i et højteknologisk samfund”. In: *Datamatbeherskelse og almen* *dannelse*. Ed. by Oluf Danielsen and Benny Karpatschof. Aarhus, Denmark: Aarhus Universitetsforlag, pp. 9–37. isbn: 87-7288-207-7.

diSessa, Andrea A. (2001). *Changing Minds: Computers, Learning, and Literacy*. MIT Press. 271 pp. isbn: 978-0-262-54132-9.

diSessa, Andrea A. and Hal Abelson (1986). "Boxer: a reconstructible computational medium". In: *Communications of the ACM* 29.9, pp. 859–868. doi: 10.1145/6592.6595.

Dourish, Paul (1999). "Embodied interaction: Exploring the foundations of a new approach to HCI". In: *Work*, pp. 1–16.

— (2014a). "No SQL: The Shifting Materialities of Database Technologies". In: *Computational culture* 4. url: http://computationalculture.net/no-sql-the-shifting-materialities-of-database-technology/.

— (2014b). "Reading and interpreting ethnography". In: *Ways of Knowing in HCI*. Ed. by Judith S. Olson and Wendy A. Kellogg. Springer, pp. 1–23.

— (2022). *STUFF OF BITS: an essay on the materialities of information*. MIT Press. isbn: 978-0-262-54652-2.

Ducasse, Stéphane, Dmitri Zagidulin, Nicolai Hess, and Dimitris Chloupis (2016). *Pharo by Example 5.0*. Lille, France: Stéphane Ducasse. isbn: 978-1-365-65459-6.

Engeström, Yrjö (2015). *Learning by expanding*. Cambridge University Press.

Eve Team (n.d.). *Eve: Programming designed for humans*. url: http://witheve.com/.

Fallman, Daniel (2007). "Why Research-Oriented Design Isn't Design-Oriented Research: On the Tensions Between Design and Research in an Implicit Design Discipline". In: *Knowledge, Technology & Policy* 20.3, pp. 193–200. doi: 10.1007/s12130-007-9022-8.

Fallman, Daniel and Erik Stolterman (2010). "Establishing criteria of rigour and relevance in interaction design research". In: *Digital Creativity* 21.4, pp. 265–272. doi: 10.1080/14626268.2010.548869.

Finley, Klint (Mar. 26, 2014). "Microsoft Finally Gave Away MS-DOS. Now It Should Open Source Everything Else". In: *Wired*. url: https://www.wired.com/2014/03/msdos-source-code/ (visited on 12/22/2022).

Fischer, Gerhard and Elisa Giaccardi (2006). "Meta-design: A Framework for the Future of End-User Development". In: *End User Development*. Ed. by Henry Lieberman, Fabio Paternò, and Volker Wulf. Red. by John Karat and Jean Vanderdonckt. Vol. 9. Human-Computer Interaction Series. Dordrecht: Springer, Netherlands, pp. 427–457. isbn: 978-1-4020-4220-1. doi: 10.1007/1-4020-5386-X.

Fischer, Gerhard, Elisa Giaccardi, Y. Ye, A. G. Sutcliffe, and N. Mehandjiev (2004). "Meta-design: a manifesto for end-user development". In: *Communications of the ACM* 47.9, pp. 33–37. doi: 10.1145/1015864.1015884.

Fog, Bjarke Vognstrup and Clemens Nylandsted Klokmose (2019). "Mapping the Landscape of Literate Computing". In: *Proceedings of the 30th Annual Workshop of the Psychology of Programming Interest Group*. Newcastle, UK. url: https://www.ppig.org/papers/2019-ppig-30th-fog/.

Fog, Bjarke Vognstrup, Blanka Pálfi, Alberte Uhre Mortensen, and Line Have Musaeus (2023). "Computational self-concept: Towards an understanding of students' identities, attitudes, and beliefs". Submitted to Education and Information Technologies.

Frayling, Christopher (1993). "Research in art and design". In: *Royal College of Art research papers*, pp. 1–5.

Free Software Foundation, Inc. (n.d.). *Introduction (GNU Emacs Manual)*. url: https://www.gnu.org/software/emacs/manual/html_node/emacs/Intro.html.

Fritz, W.B. (1996). "The women of ENIAC". In: *IEEE Annals of the History of Computing* 18.3, pp. 13–28. doi: 10.1109/85.511940.

Fuchsberger, Verena, Martin Murer, and Manfred Tscheligi (2013). "Materials, materiality, and media". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Paris France: ACM, pp. 2853–2862. doi: 10.1145/2470654.2481395.

Fuegi, J. and J. Francis (2003). “Lovelace & Babbage and the creation of the 1843 ’notes’”. In: *IEEE Annals of the History of Computing* 25.4, pp. 16–26. DOI: 10.1109/MAHC.2003.1253887.

Galey, A. and S. Ruecker (2010). “How a prototype argues”. In: *Literary and Linguistic Computing* 25.4, pp. 405–424. DOI: 10.1093/llc/fqq021.

Gaver, William (2012). “What should we expect from research through design?” In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* Austin Texas, USA: ACM, pp. 937–946. DOI: 10.1145/2207676.2208538.

Granger, Brian and Fernando Pérez (2021). “Jupyter: Thinking and Storytelling with Code and Data”. In: *Computing in Science & Engineering* 23 (2), pp. 7–14. DOI: 10.1109/MCSE.2021.3059263.

Große-Bölting, Gregor, Dietrich Gerstenberger, Lara Gildehaus, Andreas Mühling, and Carsten Schulte (2021). “Identity in K-12 Computer Education Research: A Systematic Literature Review”. In: *Proceedings of the 17th ACM Conference on International Computing Education Research.* ACM. DOI: 10.1145/3446871.3469757.

Grudin, Jonathan (2017). *From Tool to Partner: The Evolution of Human-Computer Interaction.* Synthesis Lectures on Human-Centered Informatics. Cham.: Springer International Publishing. ISBN: 978-3-031-02218-0. DOI: 10.1007/978-3-031-02218-0.

Guo, Philip J. (2018). “Non-Native English Speakers Learning Computer Programming: Barriers, Desires, and Design Opportunities”. In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems.* Montreal, QC, Canada: ACM, pp. 1–14. DOI: 10.1145/3173574.3173970.

Guzdial, Mark (2008). “Education: Paving the way for computational thinking”. In: *Communications of the ACM* 51.8, pp. 25–27. DOI: 10.1145/1378704.1378713.

— (2019). “Computing Education as a Foundation for 21st Century Literacy”. In: *Proceedings of the 50th ACM Technical Symposium on Computer Science Education.* ACM. DOI: 10.1145/3287324.3290953.

Hanington, Bruce (2022). “Methods in the Making: A Perspective on the State of Human Research in Design”. In: *Design Issues* 19.4, pp. 9–18. URL: https://www.jstor.org/stable/1512087.

Haraway, Donna (1988). “Situated Knowledges: The Science Question in Feminism and the Privilege of Partial Perspective”. In: *Feminist Studies* 14.3, p. 575. DOI: 10.2307/3178066.

Hattie, John A. (1992). *Self-concept.* Psychology Press. 307 pp.

Hayes, Gillian R. (2011). “The relationship of action research to human-computer interaction”. In: *ACM Transactions on Computer-Human Interaction* 18.3, pp. 1–20. DOI: 10.1145/1993060.1993065.

— (2014). “Knowing by doing: action research as an approach to HCI”. In: *Ways of Knowing in HCI.* Ed. by Judith S. Olson and Wendy A. Kellogg. Springer, pp. 49–68.

Heintz, Fredrik, Linda Mannila, and Tommy Farnqvist (2016). “A review of models for introducing computational thinking, computer science and computing in K-12 education”. In: *2016 IEEE Frontiers in Education Conference (FIE).* Erie, PA, USA: IEEE, pp. 1–9. DOI: 10.1109/FIE.2016.7757410.

Hertzum, Morten and Kasper Hornbæk (2023). “Frustration: Still a Common User Experience”. In: *ACM Transactions on Computer-Human Interaction.* DOI: 10.1145/3582432.

Hoffman, Mark E. and David R. Vance (2005). “Computer literacy”. In: *ACM SIGCSE Bulletin* 37.1, p. 356. DOI: 10.1145/1047124.1047467.

Hollan, James, Edwin Hutchins, and David Kirsh (2000). “Distributed cognition: toward a new foundation for human-computer interaction research”. In: *ACM Transactions on Computer-Human Interaction (TOCHI)* 7.2, pp. 174–196. DOI: 10.1145/353485.353487.

Holtzblatt, Karen and Sandra Jones (2017). “Contextual inquiry: A participatory technique for system design”. In: *Participatory design: Principles and practice.* Ed. by Douglas Schuler and Aki Namioka. CRC Press, pp. 177–210.

Höök, Kristina, Jeffrey Bardzell, Simon Bowen, Peter Dalsgaard, Stuart Reeves, and Annika Waern (2015). “Framing IxD knowledge”. In: *Interactions* 22.6, pp. 32–36. DOI: 10.1145/2824892.

Hornbæk, Kasper, Aske Mottelson, Jarrod Knibbe, and Daniel Vogel (2019). “What Do We Mean by “Interaction”? An Analysis of 35 Years of CHI”. In: *ACM Transactions on Computer-Human Interaction* 26.4, pp. 1–30. doi: 10.1145/3325285.

Hornbæk, Kasper and Antti Oulasvirta (2017). “What Is Interaction?” In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. Denver, Colorado, USA: ACM, pp. 5040–5052. doi: 10.1145/3025453.3025765.

Howell, Noura, Audrey Desjardins, and Sarah Fox (2021). “Cracks in the Success Narrative: Rethinking Failure in Design Research through a Retrospective Trioethnography”. In: *ACM Transactions on Computer-Human Interaction* 28.6, pp. 1–31. doi: 10.1145/3462447.

Hsu, Yu-Chang, Natalie Roote Irie, and Yu-Hui Ching (2019). “Computational Thinking Educational Policy Initiatives (CTEPI) Across the Globe”. In: *TechTrends* 63.3, pp. 260–270. doi: 10.1007/s11528-019-00384-4.

Hyllegard, Karen, Jennifer Ogle, and Sonali Diddi (2019). “’Making’ as a Catalyst for Engaging Young Female Adolescents in STEM Learning”. In: *Theorizing STEM Education in the 21st Century*. IntechOpen. isbn: 978-1-78985-702-3. doi: 10.5772/intechopen.87036.

Ihde, Don (1975). “The Experience of Technology: Human-Machine Relations”. In: *Cultural Hermeneutics* 2.3, pp. 267–279. doi: 10.1177/019145377500200304.

— (1990). *Technology and the lifeworld: from garden to earth*. The Indiana series in the philosophy of technology. Bloomington: Indiana University Press. 226 pp. isbn: 978-0-253-20560-5.

Iversen, Ole Sejer, Rachel Charlotte Smith, and Christian Dindler (2018). “From computational thinking to computational empowerment”. In: *Proceedings of the 15th Participatory Design Conference: Full Papers - Volume 1*. ACM. doi: 10.1145/3210586.3210592.

Jacobsen, Bo, Lene Tanggaard, and Svend Brinkmann (2020). “Fænomenologi”. In: *Kvalitative metoder: en grundbog*. Ed. by Svend Brinkmann and Lene Tanggaard. Hans Reitzels Forlag, pp. 281–308.

Jakubovic, Joel, Jonathan Edwards, and Tomas Petricek (2023). “Technical Dimensions of Programming Systems”. In: *The Art, Science, and Engineering of Programming* 7.3, p. 13. doi: 10.22152/programming-journal.org/2023/7/13. url: https://tomasp.net/techdims/ (visited on 05/30/2023).

Jamieson, Andrew (Mar. 1, 2021). *Excel Won’t Go Away*. Medium. url: https://towardsdatascience.com/excel-wont-go-away-fb856378151d (visited on 06/17/2023).

Jansen, K. and S. Vellema (2011). “What is technography?” In: *NJAS: Wageningen Journal of Life Sciences* 57.3, pp. 169–177. doi: 10.1016/j.njas.2010.11.003.

Jasanoff, Sheila and Sang-Hyun Kim (2015). *Dreamscapes of modernity: Sociotechnical imaginaries and the fabrication of power*. University of Chicago Press. isbn: 0-226-27666-X.

Johnson, J., T.L. Roberts, W. Verplank, D.C. Smith, C.H. Irby, M. Beard, and K. Mackey (1989). “The Xerox Star: a retrospective”. In: *Computer* 22.9, pp. 11–26. doi: 10.1109/2.35211.

Justice, Josh (2021). “Modifiable Software Systems: Smalltalk and HyperCard”. The Seventh Workshop on Live Programming (LIVE 2021). url: https://2021.splashcon.org/details/live-2021-papers/9/Modifiable-Software-Systems-Smalltalk-and-HyperCard.

Kafle, Narayan Prasad (2013). “Hermeneutic phenomenological research method simplified”. In: *Bodhi: An Interdisciplinary Journal* 5.1, pp. 181–200. doi: 10.3126/bodhi.v5i1.8053.

Kaiser, Cameron (Mar. 17, 2023). *PLATO: How an educational computer system from the ’60s shaped the future*. Ars Technica. url: https://arstechnica.com/gadgets/2023/03/plato-how-an-educational-computer-system-from-the-60s-shaped-the-future/ (visited on 05/30/2023).

Kaptelinin, Victor and Bonnie A. Nardi (2006). *Acting with technology: Activity theory and interaction design*. MIT press. isbn: 0-262-11298-1.

— (2012). “Affordances in HCI: Toward a Mediated Action Perspective”. In: *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems - CHI ’12*. ACM Press, pp. 967–976. doi: 10.1145/2207676.2208541.

Kato, Jun and Keisuke Shimakage (2020). “Rethinking programming “environment”: technical and social environment design toward convivial computing”. In: *Conference Companion of the 4th*

International Conference on Art, Science, and Engineering of Programming. Porto, Portugal: ACM, pp. 149–157. DOI: 10.1145/3397537.3397544.

Kay, Alan (1972). "A Personal Computer for Children of All Ages". In: *Proceedings of the ACM Annual Conference - Volume 1*. ACM '72. ACM. DOI: 10.1145/800193.1971922.
— (1984). "Computer Software". In: *Scientific American* 251.3, pp. 52–59. DOI: 10.1038/scientificamerican0984-52.
— (2013a). *Afterword: What is a Dynabook?* URL: https://tinlizzie.org/VPRIPapers/hc_what_Is_a_dynabook.pdf.
— (2013b). "The Future of Reading Depends on the Future of Learning Difficult to Learn Things". In: *VPRI Related Writings*.

Kay, Alan and Adele Goldberg (1977). "Personal Dynamic Media". In: *Computer* 10.3, pp. 31–41. DOI: 10.1109/C-M.1977.217672.

Kelleher, Caitlin and Randy Pausch (2005). "Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers". In: *ACM Computing Surveys (CSUR)* 37.2, pp. 83–137. DOI: 10.1145/1089733.1089734.

Kelty, Christopher (2005). "Geeks, Social Imaginaries, and Recursive Publics". In: *Cultural Anthropology* 20.2, pp. 185–214. DOI: 10.1525/can.2005.20.2.185.

Kery, Mary Beth and Brad A. Myers (2018). "Interactions for Untangling Messy History in a Computational Notebook". In: *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE. DOI: 10.1109/vlhcc.2018.8506576.

Kery, Mary Beth, Marissa Radensky, Mahima Arya, Bonnie E. John, and Brad Myers (2018). "The Story in the Notebook". In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM. DOI: 10.1145/3173574.3173748.

Kien, Grant (2008). "Technography = Technology + Ethnography: An Introduction". In: *Qualitative Inquiry* 14.7, pp. 1101–1109. DOI: 10.1177/1077800408318433.

Kinnula, Marianne, Netta Iivari, Tonja Molin-Juustila, Eino Keskitalo, Topi Leinonen, Eetu Mansikkamäki, Toni Käkelä, and Martti Similä (2017). "Cooperation, Combat, or Competence Building-What Do We Mean When We Are 'Empowering Children' in and through Digital Technology Design?" In: Thirty-eighth International Conference on Information Systems. Seoul. URL: http://jultika.oulu.fi/files/nbnfi-fe2018121150380.pdf (visited on 05/24/2023).

Kiran, Asle H. (2015). "Four Dimensions of Technological Mediation". In: *Postphenomenological Investigations*. Ed. by Robert Joseph Rosenberger and Peter-Paul Verbeek. London: Lexington Books, pp. 123–140. ISBN: 978-0-7391-9436-2.

Kiran, Asle H. and Peter-Paul Verbeek (2010). "Trusting Our Selves to Technology". In: *Knowledge, Technology & Policy* 23.3, pp. 409–427. DOI: 10.1007/s12130-010-9123-7.

Klokmose, Clemens Nylandsted, James R. Eagan, Siemen Baader, Wendy E. Mackay, and Michel Beaudouin-Lafon (2015). "Webstrates: Shareable Dynamic Media". In: *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. Charlotte, NC, USA: ACM, pp. 280–290. DOI: 10.1145/2807442.2807446.

Knobel, Michele and Colin Lankshear (2014). "Studying New Literacies". In: *Journal of Adolescent & Adult Literacy* 58.2, pp. 97–101. DOI: 10.1002/jaal.314.

Knuth, Donald Ervin (1984). "Literate Programming". In: *The Computer Journal* 27.2, pp. 97–111. DOI: 10.1093/comjnl/27.2.97.
— (1992). *Literate Programming*. CSLI Lecture Notes. Stanford, Calif.: Center for the Study of Language and Information. 368 pp. ISBN: 978-0-937073-80-3.

Konzack, Lars (1999). *Softwaregenrer*. Århus: Aarhus Universitetsforlag. ISBN: 978-87-7288-765-4.

Koskinen, Ilpo Kalevi, John Zimmerman, Thomas Binder, Johan Redström, and Stephan Wensveen (2011). *Design Research Through Practice: From the Lab, Field, and Showroom*. Waltham, MA: Morgan Kaufmann/Elsevier. 204 pp. ISBN: 978-0-12-385502-2.

Kuutti, Kari (1995). “Activity Theory as a Potential Framework for Human-Computer Interaction Research”. In: *Context and Consciousness: Activity Theory and Human-Computer Interaction.* Ed. by Bonnie A. Nardi, pp. 9–22. isbn: 9780262280419. doi: 10.7551/mitpress/2137.003.0006.

Latour, Bruno (1992). “Where Are the Missing Masses? The Sociology of a Few Mundane Artifacts”. In: *Shaping Technology/Building Society: Studies in Sociotechnical Change.* Ed. by John Bijker Wiebe E. & Law. Cambridge, Mass.: MIT Press, pp. 225–258.

— (1994). “On Technical Mediation”. In: *Common Knowledge* 3.2, pp. 29–64.

— (2004). “Why Has Critique Run out of Steam? From Matters of Fact to Matters of Concern”. In: *Critical Inquiry* 30.2, pp. 225–248. doi: 10.1086/421123.

Lau, Sam, Ian Drosos, Julia M. Markel, and Philip J. Guo (2020). “The Design Space of Computational Notebooks: An Analysis of 60 Systems in Academia and Industry”. In: *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC).* Dunedin, New Zealand: IEEE, pp. 1–11. doi: 10.1109/VL/HCC50065.2020.9127201.

Lave, Jean and Etienne Wenger (1991). *Situated learning: Legitimate peripheral participation.* Cambridge university press.

Lee, Irene, Fred Martin, Jill Denner, Bob Coulter, Walter Allan, Jeri Erickson, Joyce Malyn-Smith, and Linda Werner (2011). “Computational thinking for youth in practice”. In: *ACM Inroads* 2.1, pp. 32–37. doi: 10.1145/1929887.1929902.

Li, Yeping, Alan H. Schoenfeld, Andrea A. diSessa, Arthur C. Graesser, Lisa C. Benson, Lyn D. English, and Richard A. Duschl (2020a). “Computational Thinking Is More about Thinking than Computing”. In: *Journal for STEM Education Research* 3.1, pp. 1–18. doi: 10.1007/s41979-020-00030-2.

— (2020b). “On Computational Thinking and STEM Education”. In: *Journal for STEM Education Research* 3.2, pp. 147–166. doi: 10.1007/s41979-020-00044-w.

Light, Jennifer S. (1999). “When Computers Were Women”. In: *Technology and Culture* 40.3, pp. 455–483. doi: 10.1353/tech.1999.0128.

Löwgren, Jonas and Erik Stolterman (2007). *Thoughtful interaction design: a design perspective on information technology.* 1. paperback ed. Cambridge, Mass.: MIT Press. 198 pp. isbn: 978-0-262-62209-7.

Lubar, Steven (1992). ““Do Not Fold, Spindle or Mutilate”: A Cultural History of the Punch Card”. In: *The Journal of American Culture* 15.4, pp. 43–55. doi: 10.1111/j.1542-734X.1992.1504_43.x.

Lund-Larsen, Michael (Sept. 13, 2017). *E-læringschef: Diskussionen om it som fag er ikke død.* Altinget. url: https://www.altinget.dk/uddannelse/artikel/e-laeringschef-diskussion-om-it-som-fag-er-ikke-doed (visited on 12/27/2022).

Mackay, Wendy E. (1990). “Users and customizable software: A co-adaptive phenomenon”. PhD thesis. Massachusetts Institute of Technology.

Mackay, Wendy E. and Anne-Laure Fayard (1997). “HCI, natural science and design”. In: *Proceedings of the conference on Designing interactive systems processes, practices, methods, and techniques - DIS ’97.* ACM Press. doi: 10.1145/263552.263612.

Madsen, Kim Halskov (1988). “Breakthrough by breakdown: Metaphors and structured domains”. In: *DAIMI Report Series* 243.

Malmi, Lauri, Judy Sheard, Päivi Kinnunen, Simon, and Jane Sinclair (2019). “Computing Education Theories: What Are They, and How Are They Used?” In: *Proceedings of the 2019 ACM Conference on International Computing Education Research.* Toronto, ON, Canada: ACM, pp. 187–197. doi: 10.1145/3291279.3339409.

Marwan, Samiha, Ge Gao, Susan Fisk, Thomas W. Price, and Tiffany Barnes (2020). “Adaptive Immediate Feedback Can Improve Novice Programming Engagement and Intention to Persist in Computer Science”. In: *Proceedings of the 2020 ACM Conference on International Computing Education Research.* Virtual Event New Zealand: ACM, pp. 194–203. doi: 10.1145/3372782.3406264.

Mateas, Michael (2005). “Procedural literacy: educating the new media practitioner”. In: *On the Horizon* 13.2. Ed. by Drew Davidson, pp. 101–111. doi: 10.1108/10748120510608133.

McCarthy, John and Peter Wright (2004). “Technology as experience”. In: *Interactions* 11.5, pp. 42–43. doi: 10.1145/1015530.1015549.

McDonald, Nora, Sarita Schoenebeck, and Andrea Forte (2019). “Reliability and Inter-rater Reliability in Qualitative Research: Norms and Guidelines for CSCW and HCI Practice”. In: *Proceedings of the ACM on Human-Computer Interaction* 3 (CSCW), pp. 1–23. DOI: 10.1145/3359174.

McHugh, Mary L. (2012). “Interrater reliability: the kappa statistic”. In: *Biochemia Medica* 22.3, pp. 276–282.

Microsoft, Inc. (n.d.). *Microsoft software license terms: Microsoft Visual Studio Code.* URL: https://code.visualstudio.com/license.

Mol, Annemarie (2002). *The body multiple: Ontology in medical practice.* Duke University Press.

Myers, Brad, Scott E. Hudson, and Randy Pausch (2000). “Past, present, and future of user interface software tools”. In: *ACM Transactions on Computer-Human Interaction* 7.1, pp. 3–28. DOI: 10.1145/344949.344959.

Naur, Peter (1967). “Datalogi — læren om data”. Lecture. Lecture. The second of five Rosenkjær Lectures in Danish Broadcasting Corporation 1966-67 (published as Datamaskinerne og Samfundet, Munksgaard). (Visited on 05/23/2023).

Nelson, Greg L. and Amy J. Ko (2018). “On Use of Theory in Computing Education Research”. In: *Proceedings of the 2018 ACM Conference on International Computing Education Research.* Espoo, Finland: ACM, pp. 31–39. DOI: 10.1145/3230977.3230992.

Nelson, Theodor Holm (2003). “Computer Lib / Dream Machines”. In: *The New Media Reader.* Red. by Noah Wardrip-Fruin and Nick Montfort. Cambridge, Mass.: MIT Press, pp. 301–338. ISBN: 978-0-262-23227-2.

Noone, Mark and Aidan Mooney (2018). “Visual and textual programming languages: a systematic review of the literature”. In: *Journal of Computers in Education* 5.2, pp. 149–174. DOI: 10.1007/s40692-018-0101-5.

Nouwens, Midas, Marcel Borowski, Bjarke Vognstrup Fog, and Clemens Nylandsted Klokmose (2020). “Between Scripts and Applications: Computational Media for the Frontier of Nanoscience”. In: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems.* ACM. DOI: 10.1145/3313831.3376287.

Nouwens, Midas and Clemens Nylandsted Klokmose (2018). “The Application and Its Consequences for Non-Standard Knowledge Work”. In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems.* ACM. DOI: 10.1145/3173574.3173973.

Nouwens, Midas and Clemens Nylandsted Klokmose (2021). “A Survey of Digital Working Conditions of Danish Knowledge Workers”. In: *Proceedings of 19th European Conference on Computer-Supported Cooperative Work.* European Society for Socially Embedded Technologies (EUSSET).

O’hara, Keith, Douglas Blank, and James Marshall (May 2015). “Computational Notebooks for AI Education”. In: *Proceedings of the Twenty-Eighth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2015.* DOI: 10.13140/2.1.2434.5928.

Osler, Audrey (2000). “The Crick Report: difference, equality and racial justice”. In: *The Curriculum Journal* 11.1, pp. 25–37. DOI: 10.1080/095851700361375.

Oudshoorn, Nelly and Trevor J. Pinch (2008). “User-Technology Relationships: Some Recent Developments”. In: *The Handbook of Science and Technology Studies.* Ed. by Edward J. Hackett, Olga Amsterdamska, Michael Lynch, and Judy Wajcman. 3rd. Cambridge, Mass.: MIT Press, pp. 541–565.

Oulasvirta, Antti and Kasper Hornbæk (2016). “HCI Research as Problem-Solving”. In: *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems.* San Jose, California, USA: ACM, pp. 4956–4967. DOI: 10.1145/2858036.2858283.

Overgaard, Søren and Dan Zahavi (2009). “Phenomenological sociology”. In: *Encountering the everyday: An introduction to the sociologies of the unnoticed*, pp. 93–115.

Papert, Seymour (1993). *Mindstorms: children, computers, and powerful ideas.* 2nd edition. New York, NY: Basic Books.

Perez, Fernando and Brian Granger (2007). "IPython: A System for Interactive Scientific Computing". In: *Computing in Science & Engineering* 9.3, pp. 21–29. doi: 10.1109/mcse.2007.53.

Pérez, Fernando (2013). *"Literate computing" and computational reproducibility: IPython in the age of data-driven journalism.* url: http://blog.fperez.org/2013/04/literate-computing-and-computational.html.

Pérez, Fernando and Brian Granger (2015). *Project Jupyter: Computational Narratives as the Engine of Collaborative Data Science.* url: http://archive.ipython.org/JupyterGrantNarrative-2015.pdf.

Pérez-Escoda, Ana and Ma José Rodríguez-Conde (2015). "Digital literacy and digital competences in the educational evaluation". In: *Proceedings of the 3rd International Conference on Technological Ecosystems for Enhancing Multiculturality - TEEM '15.* ACM Press. doi: 10.1145/2808580.2808633.

Perlis, Alan J. (1982). "Special Feature: Epigrams on programming". In: *ACM SIGPLAN Notices* 17.9, pp. 7–13. doi: 10.1145/947955.1083808.

Petrick, Elizabeth R. (2020). "A Historiography of Human-Computer Interaction". In: *IEEE Annals of the History of Computing* 42.4, pp. 8–23. doi: 10.1109/MAHC.2020.3009080.

Pinch, Trevor J. and Wiebe E. Bijker (1984). "The Social Construction of Facts and Artefacts: or How the Sociology of Science and the Sociology of Technology might Benefit Each Other". In: *Social Studies of Science* 14.3, pp. 399–441. doi: 10.1177/030631284014003004.

Rädle, Roman, Midas Nouwens, Kristian Antonsen, James R. Eagan, and Clemens N. Klokmose (2017). "Codestrates: Literate Computing with Webstrates". In: *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology.* ACM. doi: 10.1145/3126594.3126642.

Raymond, Eric S. (1999). *The cathedral & the bazaar: musings on Linux and open source by an accidental revolutionary.* 1st ed. Beijing; Cambridge, Mass.: O'Reilly. 268 pp. isbn: 978-1-56592-724-7.

Regmi, Krishna, Jennie Naidoo, and Paul Pilkington (2010). "Understanding the Processes of Translation and Transliteration in Qualitative Research". In: *International Journal of Qualitative Methods* 9.1, pp. 16–26. doi: 10.1177/160940691000900103.

Resnick, Mitchel, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai (2009). "Scratch: programming for all". In: *Communications of the ACM* 52.11, pp. 60–67. doi: 10.1145/1592761.1592779.

Rosenberger, Robert Joseph and Peter-Paul Verbeek (2015). "A Field Guide to Postphenomenology". In: *Postphenomenological Investigations.* Ed. by Robert Rosenberger and Peter-Paul Verbeek. Lexington Books, pp. 9–42. isbn: 978-0-7391-9436-2.

Rule, Adam, Ian Drosos, Aurélien Tabard, and James Hollan (2018). "Aiding Collaborative Reuse of Computational Notebooks with Annotated Cell Folding". In: *Proceedings of the ACM on Human-Computer Interaction* 2 (CSCW), pp. 1–12. doi: 10.1145/3274419.

Rule, Adam, Aurélien Tabard, and James Hollan (2018). "Exploration and Explanation in Computational Notebooks". In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems.* ACM. doi: 10.1145/3173574.3173606.

Rychen, Dominique Simone and Laura Hersh Salganik (2003). "A holistic model of competence". In: *Key competencies for a successful life and a well-functioning society.* Ed. by Laura Hersh Salganik and Dominique Simone Rychen. Cambridge, MA; Toronto: Hogrefe & Huber, pp. 41–62. isbn: 978-0-88937-272-6.

Salovaara, Antti, Antti Oulasvirta, and Giulio Jacucci (2017). "Evaluation of Prototypes and the Problem of Possible Futures". In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems.* Denver, Colorado, USA: ACM, pp. 2064–2077. doi: 10.1145/3025453.3025658.

Satchell, Christine and Paul Dourish (2009). "Beyond the user". In: *Proceedings of the 21st Annual Conference of the Australian Computer-Human Interaction Special Interest Group on Design: Open 24/7 - OZCHI '09.* ACM Press. doi: 10.1145/1738826.1738829.

"Sayings of the High One" (2014). In: *The poetic Edda.* Trans. by Carolyne Larrington. Revised edition. Oxford world's classics. Oxford: Oxford University Press, pp. 13–35. isbn: 978-0-19-967534-0.

Schneider, Hanna, Malin Eiband, Daniel Ullrich, and Andreas Butz (2018). “Empowerment in HCI - A Survey and Framework”. In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. Montreal, QC, Canada: ACM, pp. 1–14. DOI: 10.1145/3173574.3173818.

Schön, Donald A. (1983). *The reflective practitioner: how professionals think in action*. New York: Basic Books. 374 pp. ISBN: 978-0-465-06878-4.

Schulte, Carsten and Lea Budde (2018). “A Framework for Computing Education: Hybrid Interaction System: The need for a bigger picture in computing education”. In: *Proceedings of the 18th Koli Calling International Conference on Computing Education Research*. Koli, Finland: ACM, pp. 1–10. DOI: 10.1145/3279720.3279733.

Schulte, Eric, Dan Davison, Thomas Dye, and Carsten Dominik (2012). “A Multi-Language Computing Environment for Literate Programming and Reproducible Research”. In: *Journal of Statistical Software* 46.3, pp. 1–24. DOI: 10.18637/jss.v046.i03.

Schuurman, Nadine (2008). “Database Ethnographies Using Social Science Methodologies to Enhance Data Analysis and Interpretation”. In: *Geography Compass* 2.5, pp. 1529–1548. DOI: 10.1111/j.1749-8198.2008.00150.x.

Shavelson, Richard J., H.W. Marsh, and B.M. Byrne (1992). “A multidimensional, hierarchical self-concept”. In: SUNY Press, pp. 44–95.

Singer, Jeremy (2020). “Notes on notebooks: is Jupyter the bringer of jollity?” In: *Proceedings of the 2020 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. Virtual, USA: ACM, pp. 180–186. DOI: 10.1145/3426428.3426924.

Song, F., S. Parekh, L. Hooper, Y. K. Loke, J. Ryder, A. J. Sutton, C. Hing, C. S. Kwok, C. Pang, and I. Harvey (2010). “Dissemination and publication of research findings: an updated review of related biases”. In: *Health Technology Assessment* 14.8. DOI: 10.3310/hta14080.

Sørensen, Marie-Louise Stisen Kjerstein, Bjarke Vognstrup Fog, Line Have Musaeus, and Marianne Graves Petersen (2022). “KnitxCode: Exploring a Craftsmanship-driven Approach to Computational Thinking”. In: *Adjunct Proceedings of the 2022 Nordic Human-Computer Interaction Conference*. Aarhus, Denmark: ACM, pp. 1–5. DOI: 10.1145/3547522.3547680.

Spangsberg, Thomas Hvid and Martin Brynskov (2017). “Towards a dialectic relationship between the implicit and explicit nature of computational thinking”. In: *Proceedings of the 17th Koli Calling Conference on Computing Education Research - Koli Calling ’17*. ACM Press. DOI: 10.1145/3141880.3144591.

Stallman, Richard M. (1981). “EMACS — the extensible, customizable self-documenting display editor”. In: *Proceedings of the ACM SIGPLAN SIGOA symposium on Text manipulation*, pp. 147–156.

— (2015). *Free Software, Free Society: Selected Essays of Richard M. Stallman*. 3. ed. Boston, MA: Free Software Foundation. URL: https://www.gnu.org/doc/Press-use/fsfs3-hardcover.pdf.

Star, Susan Leigh (1999). “The Ethnography of Infrastructure”. In: *American Behavioral Scientist* 43.3, pp. 377–391. DOI: 10.1177/00027649921955326.

Stigler, George J. (1970). “The case, if any, for economic literacy”. In: *The Journal of Economic Education* 1.2, pp. 77–85.

Straube, Till (2016). “Stacked spaces: Mapping digital infrastructures”. In: *Big Data & Society* 3.2. DOI: 10.1177/2053951716642456.

Streeck, Jürgen, Charles Goodwin, and Curtis LeBaron (2011). “Embodied interaction in the material world: An introduction”. In: *Embodied interaction: Language and body in the material world* 1, p. 26.

Suchman, Lucille Alice (1987). *Plans and situated actions: the problem of human-machine communication*. Cambridge [Cambridgeshire]; New York: Cambridge University Press. ISBN: 978-0-521-33739-7.

Taylor, Nick, Keith Cheverst, Peter Wright, and Patrick Olivier (2013). “Leaving the wild: lessons from community technology handovers”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Paris, France: ACM, pp. 1549–1558. DOI: 10.1145/2470654.2466206.

Tedre, Matti (2020). “From a Black Art to a School Subject”. In: *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*. ACM. doi: 10.1145/3341525.3394983.

Tedre, Matti and Peter J. Denning (2016). “The long quest for computational thinking”. In: *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*. Koli, Finland: ACM, pp. 120–129. doi: 10.1145/2999541.2999542.

The Org Mode Community (n.d.). *Org Mode*. url: https://orgmode.org/.

Thurmond, Veronica A. (2001). “The Point of Triangulation”. In: *Journal of Nursing Scholarship* 33.3, pp. 253–258. doi: 10.1111/j.1547-5069.2001.00253.x.

Tissenbaum, Mike, Josh Sheldon, and Hal Abelson (2019). “From computational thinking to computational action”. In: *Communications of the ACM* 62.3, pp. 34–36. doi: 10.1145/3265747.

Tissenbaum, Mike, Josh Sheldon, Lissa Seop, Clifford H. Lee, and Natalie Lao (2017). “Critical computational empowerment: Engaging youth as shapers of the digital future”. In: *2017 IEEE Global Engineering Education Conference (EDUCON)*. Athens, Greece: IEEE, pp. 1705–1708. doi: 10.1109/EDUCON.2017.7943078.

Tracy, Sarah J. (2010). “Qualitative Quality: Eight “Big-Tent” Criteria for Excellent Qualitative Research”. In: *Qualitative Inquiry* 16.10, pp. 837–851. doi: 10.1177/1077800410383121.
— (2013). *Qualitative Research Methods*. John Wiley & Sons. 368 pp. isbn: 978-1-4051-9203-3.

Trois, Celio, Marcos D. Del Fabro, Luis C. E. de Bona, and Magnos Martinello (2016). “A Survey on SDN Programming Languages: Toward a Taxonomy”. In: *IEEE Communications Surveys & Tutorials* 18.4, pp. 2687–2712. doi: 10.1109/COMST.2016.2553778.

Tuhkala, Ari, Marie-Louise Wagner, Ole Sejer Iversen, and Tommi Kärkkäinen (2019). “Technology Comprehension — Combining computing, design, and societal reflection as a national subject”. In: *International Journal of Child-Computer Interaction* 20, pp. 54–63. doi: 10.1016/j.ijcci.2019.03.004.

Vee, Annette (2013). “Understanding computer programming as a literacy”. In: *Literacy in Composition Studies* 1.2, pp. 42–64. url: http://d-scholarship.pitt.edu/id/eprint/21695.
— (2017). *Coding literacy: how computer programming is changing writing*. Software studies. Cambridge, MA: The MIT Press. 361 pp. isbn: 978-0-262-03624-5.

Veenman, Marcel V. J., Bernadette H. A. M. Van Hout-Wolters, and Peter Afflerbach (2006). “Metacognition and learning: conceptual and methodological considerations”. In: *Metacognition and Learning* 1.1, pp. 3–14. doi: 10.1007/s11409-006-6893-0.

Verbeek, Peter-Paul (2001). “Don Ihde: The Technological Lifeworld”. In: *American philosophy of technology: the empirical turn*. Ed. by Hans Achterhuis. The Indiana series in the philosophy of technology. Bloomington: Indiana University Press, pp. 119–146. isbn: 978-0-253-21449-2.
— (2005). “Artifacts in Design”. In: *What Things Do: Philosophical Reflections on Technology, Agency, and Design*. Penn State University Press, pp. 203–236. isbn: 978-0-271-03322-8. doi: 10.1515/9780271033228.
— (2015). “Beyond interaction: a short introduction to mediation theory”. In: *Interactions* 22.3, pp. 26–31. doi: 10.1145/2751314.

Vista, Alvin (2020). “Teaching coding as a literacy: Issues, challenges, and limitations”. In: *Academia Letters*. doi: 10.20935/AL5.

Vygotsky, Lev Semenovich and Michael Cole (1978). *Mind in society: Development of higher psychological processes*. Harvard University Press. isbn: 0-674-57629-2.

Waltz, Scott B. (2004). “Giving artifacts a voice? Bringing into account technology in educational analysis”. In: *Educational theory* 54.2, pp. 157–172.

Weintrop, David, Nathan Holbert, and Mike Tissenbaum (2020). “Considering alternative endpoints: An exploration in the space of computing educations”. In: *Proceedings of the Constructionism Conference, Dublin, Ireland*.

Winestock, Rudolph (Oct. 6, 2017). *The Lisp Curse*. Winestock Webdesign. url: http://www.winestockwebdesign.com/Essays/Lisp_Curse.html (visited on 05/12/2023).

Wing, Jeannette M. (2006). “Computational thinking”. In: *Communications of the ACM* 49.3, pp. 33–35. doi: 10.1145/1118178.1118215.

Yeung, Alexander Seeshing, Dennis M. McInerney, and Deirdre Russell-Bowie (2001). “Hierarchical, multidimensional creative arts self-concept”. In: *Australian Journal of Psychology* 53.3, pp. 125–133. doi: 10.1080/00049530108255134.

Zimmerman, John and Jodi Forlizzi (2014). “Research through design in HCI”. In: *Ways of Knowing in HCI*. Ed. by Judith S. Olson and Wendy A. Kellogg. Springer, pp. 167–189. doi: 10.1007/978-1-4939-0378-8_8.

Zimmerman, John, Erik Stolterman, and Jodi Forlizzi (2010). “An analysis and critique of *Research through Design*: towards a formalization of a research approach”. In: *Proceedings of the 8th ACM Conference on Designing Interactive Systems - DIS ’10*. Aarhus, Denmark: ACM Press. doi: 10.1145/1858171.1858228.

List of Figures 7

List of Tables 7