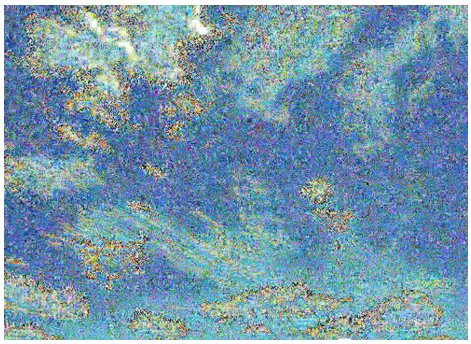
Towards Critical Abstraction for Software

Alice Mira Chung
University of California, San Diego
La Jolla, USA
a3chung@ucsd.edu

Devamardeep Hayatpur
University of California, San Diego
La Jolla, USA
dshayatpur@ucsd.edu

Philip J. Guo
University of California, San Diego
La Jolla, USA
pjguo@ucsd.edu

Tomas Petricek
Charles University
Prague, Czech Republic
tomas@tomasp.net



❶ David Elliott's **Generation Loss** (2009)

❷ Yeesookyung's **Translated Vase** series (2001-ongoing)

❸ Félix González-Torres's **"Untitled" (Portrait of Ross in L.A.)** series (1991)

Figure 1. Different conceptions of abstraction in contemporary art. ❶) A digital photograph is saved 600 times with compression, where the final image degrades into an abstract color field. There is a gradual loss of information (by JPEG's encoding algorithm) until only structural artifacts remain [41, 42]. ❷) Discarded piles of ceramics are collected and re-assembled into a series of sculptures. The sculpture no longer represents or functions as recognizable ceramic objects, but is instead abstracted into an amorphous form to reflect on healing after the violent disruption of Korea's cultural heritage [64, 112]. ❸) A candy pile equals to the weight of the artist's partner Ross Laycock, who died of complications from AIDS. Museum visitors were asked to take away candies from this installation piece, using abstraction to convey loss and physical decline [51, 109].

Abstract

Abstraction in computer science is typically understood as the process of generalization that removes non-essential details and captures key shared characteristics across multiple instances. This is a limited view of abstraction when contrasted with fields like art, architecture, media theory, or philosophy, where abstraction has been repeatedly re-examined and re-imagined, providing space for self-reflection and critical discourse. To show that abstraction can play a similarly critical role in computer science, we examine three pairs of related software artifacts where one part of each pair uses abstraction in a conventional way, while the other invites critical reflection. We demonstrate how different modes of abstraction express critical ideas about computing, in the form of software artifacts. Taken together, we argue that pluralistic approaches to abstraction can make computing a more self-reflective field aware of the methodological and sociopolitical issues it faces.

CCS Concepts: • **Human-centered computing** → **Empirical studies in visualization**; *Graphical user interfaces.*

Keywords: abstraction, representation, critical artifacts, critical computing

Onward! '26, Oakland, CA, USA

ACM ISBN 979-8-4007-2934-8/2026/10
https://doi.org/10.1145/3840586.3843748

ACM Reference Format:
Alice Mira Chung, Devamardeep Hayatpur, Philip J. Guo, and Tomas Petricek. 2026. Towards Critical Abstraction for Software. In *Proceedings of the 2026 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! '26), October 4–9, 2026, Oakland, CA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3840586.3843748

1 Introduction

In computing, "abstraction" usually means to hide unnecessary details, remove irrelevant content, or generalize to a new level of description. The field has long treated this as productive, and many advances in computing have been credited to new ways of abstracting. In the late 1950s, high-level programming languages such as Algol and Cobol reduced the need to deal with machine-specific instruction sets and hardware configurations [83]. In the 1970s, abstract data types and object-oriented programming [74] made it possible to decompose software into components, while hiding their specific implementations. The common thread is that hiding the details repeatedly introduced a new vocabulary to describe computation [27, 80].

We will refer to this mode of abstraction as a type of *symbolic abstraction*, in line with prior work in computer science [40, 52, 82] and the history of computing [11, 83]. Symbolic abstraction is concerned with designing new representations: a set of discrete symbols, a formal grammar that governs how they compose and transform. These new representations provide an interface for people (programmers and users) to operate them without needing to know what lies beneath. [57]. Symbolic abstraction replaces complex behavior with simpler, general-purpose representations through an interplay of two moves [80]: *decontextualization* [23, 35, 47, 80], omitting non-essential details (*e.g.* digitization, datafication, hiding the implementation of a module or a user interface) and *generalization* [63, 80, 86], extracting similarities across cases into a new structure that subsumes instances (*e.g.*, subroutines, parameterized functions, inheritance). When we design and teach, we tend to organize these symbolic abstractions hierarchically, hinting at this structure through familiar terms like *levels* (*e.g.*, high-*level* languages).

Shortcomings of this model of abstraction have been discussed within computer science. Kell [62] contends that information-hiding (*i.e.* abstraction through encapsulation) produces software that is opaque and closed into "islands of functionality" that hinder the development of "flexible and easily-interoperable software." The "Law of Leaky Abstractions" [101] states that non-trivial abstractions often expose the underlying implementation they were meant to hide. Steimann [102] presents examples of software where abstraction is overly specific, overly general, or delusive putting the software at risk of failure, while Petricek [87] argues that the lack of a unified abstraction benefited the historical development of the notion of a 'type'. Abstraction has also come under critique from interaction design, for enabling a particular kind of power relations [71] on what information to expose at each level and who gets to decide it.

Outside computing, in fields like art and architecture, notions of abstraction have been repeatedly questioned and expanded, at times aligned with, at times against, the symbolic abstraction framework. This essay is an exercise in re-interpreting abstraction for software artifacts following the precedent set by art and architecture. It questions the assumptions embedded in symbolic abstraction.

For example, media studies has examined how abstraction processes 'compress.' *Degradation* (e.g. material degradation in analog media, poor images generated by lossy compression) [47, 103] can be viewed as a form of abstraction as a viable mode for information compression, which blurs the distinction between essential and non-essential information (Figure 1, ❶). On the other hand, critical theory has questioned ways abstraction processes 'collects' instances through the concept of an *assemblage* [32, 81] (*e.g.*, a collage assembled with different objects), which creates a formation without generalizing and substituting its constituents into a new high-level object (Figure 1, ❷). Alternatively, linguists and cognitive scientists have examined conceptual *metaphor*, entities and structures that 'map' from one domain to another [49]. Their model explains how conceptual metaphor maps and grounds between representational systems, bridging between concrete experiences and abstract thoughts [49, 50, 69, 70] (Figure 1, ❸).

Moreover, software itself might be a promising vehicle through which to critique and question abstraction. We take inspiration from *criticism from within* [99] in architecture, a technique through which post-modern architects playfully critique and question existing practices. To introduce this approach, we will juxtapose three conventional artifacts against three critical uses of abstraction in art and architecture. Then, in the main part of this essay, we will similarly juxtapose a computational artifact that uses abstraction in a conventional manner (*e.g.*, a diagram from a textbook) against an artifact that uses abstraction critically (*e.g.*, an artistic exploration).

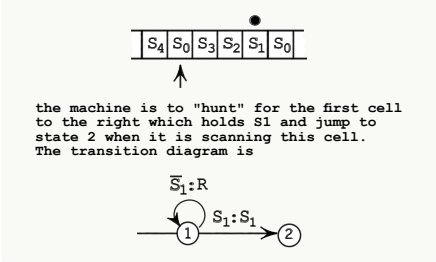
We hope to prompt reflection on how the prevailing understanding of abstraction narrows the design and study of computing artifacts, and how software practice might open itself to more pluralistic approaches.

2 Symbolic Abstraction in Computing

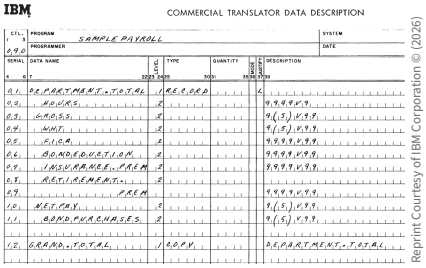
Separation of structure from content. The notion of software[1] as we understand it today did not exist in the early 1940s, when programming consisted entirely of physical tasks: making connections, setting switches, and inputting values. In her article "On Software, or the Persistence of Visual Knowledge," new media theorist Wendy Hui Kyong Chun [23] describes how computing was isolated from its physical machine into reusable code during the 1950s:

> In order to emerge as a language or a text, software and the "languages" on which it relies had

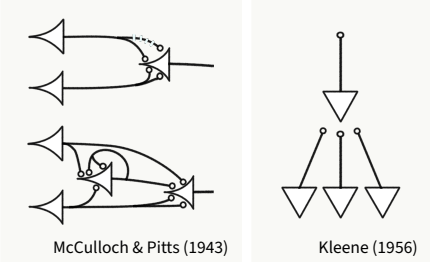
[1]The term 'software' is itself somewhat problematic. As discussed by De Mol and Bullynck [31], it only appeared in the early 1960s with the rise of the business computing industry, apparently first in job ads.



❶ **Turing Machine's Tape**
Clark (1955)



❷ **Data Description Sheets**
IBM Commercial Translator Manual (1960)



❸ **Neural Networks**

Figure 2. Examples of dematerialization in computing. ❶) A notation of a Turing machine by Clark [25] (1955) shown as "tape divided into squares" (Top) alongside a dematerialized node-and-wire state diagram (Bottom) [83]; redrawn from the original Clark illustration. ❷) A part of data description coding form specifying data layout on a physical tape (1960); reproduced from the system's original manual [58]. ❸) A notation of neural nets using symbols for neurons, axons, and synapses by McCulloch and Pitts [79] (1943), and a dematerialized notation of finite automata by Kleene [68] (1956); redrawn portions of the figures from the respective sources.

> to become iterable. With programming languages, the product of programming would no longer be a running machine but rather this thing called software—something theoretically (if not practically) iterable, repeatable, reusable, no matter who wrote it or what machine it was destined for. Programming languages inscribe the absence of both the programmer and the machine in its so-called writing. Programming languages enabled the separation of instruction from machine, of imperative from action.

Since then, computing has taken a "linguistic turn" through symbolic abstraction [11, 84], shifting away from the specifics of the physical machine [35].

This notion of abstractness became recognized as a cornerstone in academic computer science. Computer scientists began emphasizing the *structure* of information over its material content [11]. Historian David Nofre points to George Forsythe's 1965 talk as emblematic of this shift: "[o]pen up a computer, find only 0's and 1's [...] content is meaningless, and structure is all-important" [45]. This tendency to abstract away from material content is also visible in various visual artifacts (see Figure 2) from the discipline's formative years, which progressively lost their direct reference to material forms and become incorporated into machine-independent, mathematical notational systems.

Computing has continued to shift towards formal rigor [11, 56, 83, 89] by separating structure from "communicational matters" [11]. This perspective positions computer code as a move away from ambiguous, fuzzier, and 'messier' symbolic systems of human communication (*i.e.*, spoken language), where the meaning of a symbol is loosely determined by its use in social context [105]. Modern computing practice, instead, treats "words and other symbols as arbitrary tokens with no inherent meanings" [11] beyond how they will be interpreted and executed by the machine [2].

This division between computer code and human communication, however, appears less stable when examining code in its social settings where code often holds performative and communicative meanings [5, 29, 76, 94, 108, 111]. The familiar "Hello world" program, for instance, is an example of a speech act [94], a greeting in English that initiates interaction among the programmer, the machine, and the user. The 2009 Climatic Research Unit email controversy offers another example. Source code for the research unit's climate model contained comments (*e.g.*, "`fudge factor`" or "`APPLY ARTIFICIAL CORRECTION`")[3], which were interpreted beyond the programmer's original intent, transforming into "a means of public debate" through the way they were received and circulated [76]. As these examples demonstrate, the formal separation of structure from content is incomplete in practice: code circulates as a social artifact with readings that exceed its execution. In the pursuit of formal separation, code inevitably stands in place of things beyond the programmable actions, until it loses the illusion of control [24] over the hidden 'messy surplus' of material reality. Code remains a source of things beyond execution, producing a surplus that symbolic abstraction cannot fully discipline away.

[2]Jeffrey M. Binder points out that this separation is again reversed by LLMs, where tokens are defined by the large context of sequences in which they appear [11]. LLMs' performance has renewed debate over the symbol grounding problem: for example, Bender and Koller [10] argue that a system trained only on linguistic form, with no signal of speakers' communicative intent, cannot in principle acquire the relation between form and meaning, while Søgaard [100] and others have pushed back.

[3]These comments were temporary labels that were made while investigating a discrepancy between multiple data sources.

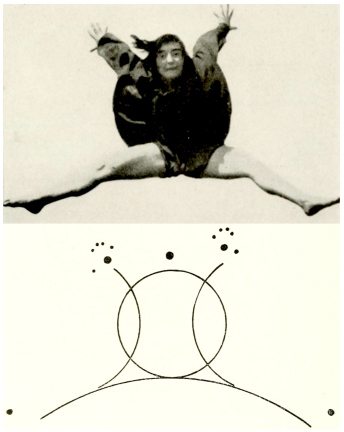


Figure 3. Gret Palucca leaping and Kandinsky's diagram from *Point and Line to Plane* [61] – Wassily Kandinsky (1926).

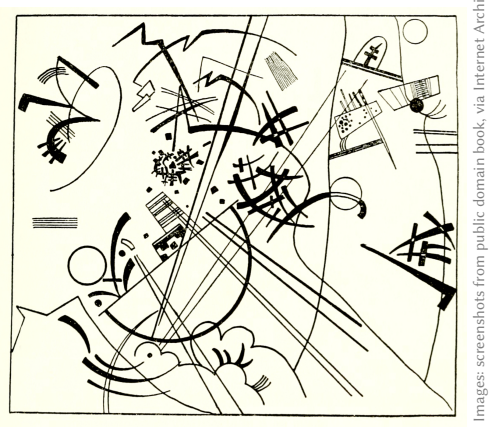


Figure 4. Linear structure of *Little Dream in Red* from *Point and Line to Plane* [61] – Wassily Kandinsky (1926).

Construction of Level of Abstraction. Parallel to developments in symbolic abstraction, the mid-20th century saw various scientific frameworks reorient themselves around "*levelism*," where complex phenomena are modeled by organizing information into modules nested in layers, or Levels of Abstraction (LoA) [43, 110]. Although the ontological roots of levelism trace back to Plotinus in ancient Greece, by the 1970s and 1980s it had become "a standard approach both in science and philosophy" [43]. Philosopher Jaegwon Kim described its prominence in physics: The physical world is modeled as "a layered world, a hierarchically stratified structure of 'levels' or 'orders' of entities and their characteristic properties," with a fundamental bottom level consisting of "the most basic physical particles out of which all matter is composed (electrons, neutrons, quarks, or whatever)" [65].

As levelism rose in prevalence, it invited closer critical examinations from philosophy of science. First, critiques questioned whether *certain levels have causal power over other levels* (or constrain other levels) in LoA [65, 66, 95]. The causal asymmetry of LoA implies that all phenomena ultimately trace back to the physical reality (*i.e.*, the lowest level). Then higher level of abstraction can only have "derivative causal power," where independence between each level becomes questionable[4] [65, 66]. This leads to another point of contention: How does *one level relate to another*? LoA often defines a higher level in terms of its purpose or a functional description which does not clarify how different levels connect (*i.e.*, *what it is for* versus *how it is implemented*), leading to semantic indeterminacy [72]. Lastly, LoA may conflate a change in context with a change in unit of analysis [78].

[4]This characterization from general science does not map cleanly onto computer science's accounts, where levels are designed, not discovered, and their adequacy is judged by how well they capture intended behavior at a given level of abstraction.

Whether LoAs are built bottom-up or top-down is a more complex issue in the philosophy of computer science than in the natural sciences. For example, Floridi [43, 44] argues that the method of abstraction "underpins top-down construction," where a computational system is built through iterative passes, each refining an initial approximation. The machinery is built by organizing observables into a hierarchical LoA, allowing multiple LoAs to be nested or branched into progressively complex architectures. In practice, Floridi claims, designers typically begin with high-level descriptions (specifications, functions, behaviors, etc.) and then descend through the LoA to implement details [43].

Tanaka-Ishii [105] provides a complementary account from semiotics: Unlike other organically *structured* communicative systems, programming languages are artificially *constructed* (*i.e.*, "generated from a minimal core of signs" where all relations among signs are designed "by necessity" [105]) bottom-up to ensure a program runs under formal requirements. Tanaka-Ishii highlights one such requirement, "halting," that forces construction to be bottom-up: because designers must guarantee that programs will eventually terminate, it makes sense to first establish this property for the smallest unit of language and then build upward, ensuring the guarantee is preserved at each step [105]. In practice, the LoAs in computing likely proceed via a continuous back-and-forth between top-level functional specifications and bottom-level executional guarantees.

Levelism has also been critiqued as an insufficient model of abstraction for computing. Ganascia [48] contends that there are different meanings of abstraction in computing and "the partial order induced by the relationship of abstraction is not universal" in the field. Colburn and Shute point out that abstraction in computer science centers around *modeling of interaction patterns*, built on "pluralistic and multilayered" [26] formalisms unlike those in other scientific fields.

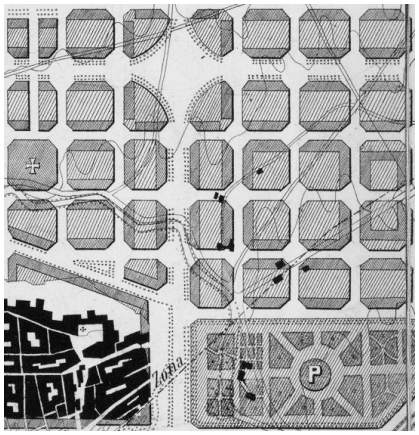


Figure 5. Plan for Barcelona [22] – Ildefons Cerdà (1859).

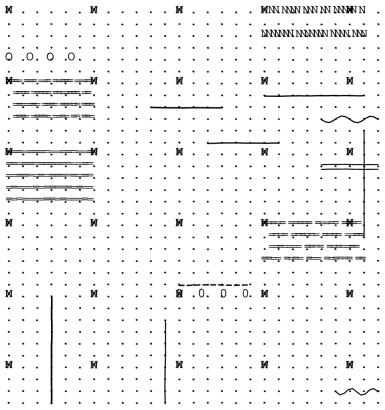


Figure 6. *No-Stop City* [17] – Archizoom (1967) recreated[5].

This pluralism in our field invites closer scrutiny in practice, where specific conceptions of abstraction settle into defaults that shape how software is built and used.

3 Abstraction in Art and Architecture

To help us imagine a more pluralistic conception of abstraction in software, we will take a look at three pairs of artworks and designs from art and architecture. Modern and postmodern art and architecture have repeatedly questioned and re-conceptualized abstraction, using it as an instrument to question and refuse established practices. We will juxtapose three pairs of cases from art and architecture, each presenting two perspectives that diverge on abstraction – one that adheres to established field norms, and another that critiques those conventions by creating a new artifact.

Abstracting form in art. The abstract art movement in the early 20th century was a forking point for two interpretations of abstraction: (i) abstraction as removing concreteness from a representation, and (ii) abstraction as an *encoding*, a systematic conversion of an image into a composition of symbols and cultural codes. For example, in *Point and Line to Plane* (1926) [61] Wassily Kandinsky analyzed how the geometric point and line operate as universal primitive forms that translate across different expressive mediums (*e.g.*, speech, sculpture, music, and dance) (Figure 3). Going further, he proposed to isolate the primitives from representations as building blocks for compositions under organizational "laws," aiming to create resonance with viewers (Figure 4) [61]. This structural approach to abstraction, free from figurative references, marked a shift in visual art from realistic fidelity to "systematic production" [36] based on visual codes. Through it, early abstract painters pursued graphic languages that operated "beyond the simple binary distinction of representational and abstract" [34].

This distinction shows that the process of abstraction can be interpreted in one way, as *omitting features* or removing "concreteness" from the observed subject, and also in another way as *extracting and encoding* a set of visual elements (shape, color, etc.) to synthesize a new form. This shift foreshadows the various reinterpretations of abstraction in the following century, which Section §5 explores.

Grid in urban planning. In architecture, abstraction has been discussed as a formal tool to systematize space, form, and flow, serving the functional needs of the inhabitants since the early 20th century. In this sense, both architectural works and software converge in their prioritization of functional concerns.

Figures 5 and 6 show two examples of urban city plans based off a grid structure[6]. Cerdà's 1859 city plan for Barcelona [22] proposed a continuous grid of city blocks and was mainly motivated by egalitarian principles, public health, and infrastructural reasons. Cerdà used scientific and statistical methods to design multiple aspects of the plan: the height of the buildings was chosen so that sunlight would reach the ground in the winter, schools and other amenities were distributed in a way that made them accessible on foot, the width of the streets allowed for the future inclusion of trams, while the truncated edges were justified by air movement and traffic considerations.

In contrast, Archizoom's *No-Stop City* [17] takes the rationalist ideology of architectural modernism to its most absurd limits. Using a typewriter to generate repetitive, bureaucratic plans, it imagines the city as a possibly infinite homogeneous space. The city is no longer defined by its buildings and architectural forms, but becomes shaped by flows of people,

[5]The image is a screenshot of a procedural system made in p5.js by the first author that computationally reconstructs the concept of *No-Stop City* drawings [17].

[6]The conventional modernist view of a grid as a rational structure has been challenged by Aureli [6], who points out the political significance of the grid. Historically, the grid structure emerged as a way of subdividing land into measurable plots, allowing private ownership, state land management, and deskilling through mechanization.

Figure 7. Casa del Fascio [107] – Giuseppe Terragni (1936).

Figure 8. A model of House X [37] – Peter Eisenman (1975).

information and services. Thanks to advanced technologies, there is no need for centralization. Although *No-Stop City* is presented as an urban plan, it is better seen as an ironic and critical statement that exposes the mannerisms and myths of modernism. By taking the underlying abstraction to an extreme, it reveals the contradictions between the egalitarian principles of abstraction and the resulting dehumanization.

Formal structures in architecture. Abstraction in architecture can be used to serve rationalist goals, or, alternatively, can be used to prompt critical reflection. Casa del Fascio's rationalist architecture [107] (Figure 7) epitomizes the orderly ideals of the Fascist regime it was designed to serve. It is a perfect half-cube (its height, being a half the length of its base) and its facade, formed by 20 rectangles, reflects its internal reinforced concrete grid structure. The central empty atrium provides space for public gathering and is surrounded by private administrative spaces, creating a power dialectic between being watched and watching. Peter Eisenman is directly influenced by Terragni's project. In his doctoral dissertation, The Formal Basis of Modern Architecture [39, 53], Eisenman analyzes the formal structure of architecture and uses Casa del Fascio as one example. Acknowledging the half-cube structure, he interprets the building in terms of volumes and planes. According to Eisenman's formal reading, the plane of the front facade is distorted by the entrance (three rectangles in the middle) and a corresponding distortion on the upper floor, resulting in a symmetric H shape.

Eisenman's own later work, including House X (Figure 8) [37, 38], adopts a language that is seemingly similar to modernist architecture, but rather than starting from function, his designs are the result of formal geometric considerations. In particular, House X starts with four L-shaped blocks, questioning the familiar cubic form as the starting point for a house. The use of the L shape disrupts the legibility of the house, "gives false clues about its reading" [38] and explores the limits of rationality. Later, Eisenman constructed an axonometric physical model for House X, a skewed 3D construct that collapses to his conceptual 2D drawing when viewed from a certain angle. This was to emphasize "model as idea," [9] where physical model serves as a primary site of critical thinking rather than a conventional miniature replica of a proposed building.

Critique through artifacts. The three examples from art and architecture suggest how abstraction as a lens can question established practices and ways of thinking. In all these cases, the critique or questioning was carried out in the form or language of art and architecture practice itself. So an artwork itself demonstrates an abstraction outside the concerns of fidelity and realistic representation; an architectural plan stands as an idea against how abstraction is expected to prioritize and be utilized in the field. We refer to these artworks or plans as *critical artifacts*, where a critical stance is proposed and propagated *through* an artifact [91].

Despite the fact that both our examples from architecture are unbuilt or unbuildable plans, this does not necessarily mean critical artifacts should be only performative and non-functional [91]. Many post-modern buildings are functional (*i.e.*, buildable and inhabitable) while posing a critical stance. As one example, the "inside-out" building of Centre Pompidou exposes the infrastructure on the exterior of the building, revealing technical details that are usually hidden [30].

4 Critical Software Artifacts

Can the critical methods and attitude we have encountered in art and architecture be applied to software as well? We can imagine critical software artifacts in the form of implemented software systems, as well as diagrams, plans, or code that does not need to run. We want to encourage more computer scientists and programmers at all levels to engage in critical discourse through creating critical software artifacts. As a start, we will reinterpret existing projects with this new priority of deconstructing the conventional design

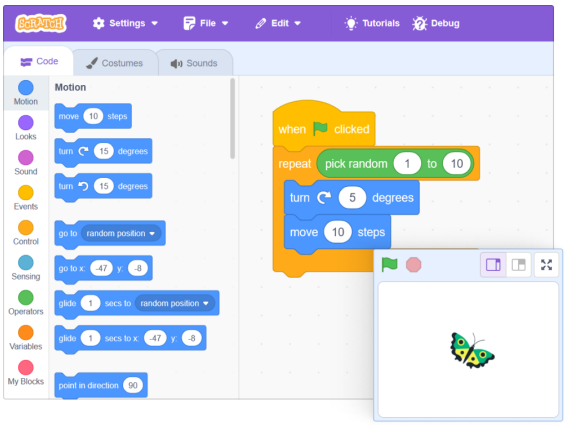


Figure 9. Scratch [75] is a structured programming environment where the user can program actors by putting together predefined code blocks.

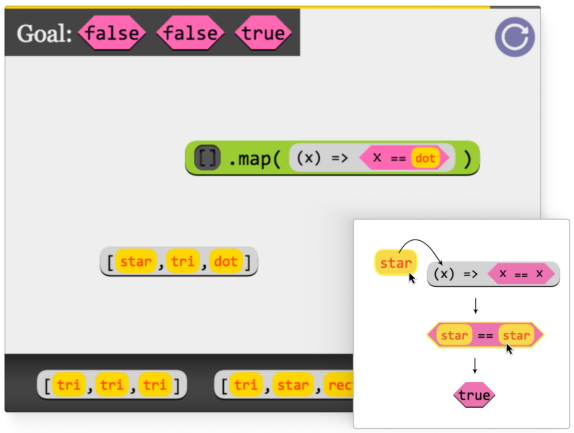


Figure 10. Reduct [4] is a programming puzzle game with a linear progression curriculum, where each stage corresponds to a pedagogical concept about the execution of JavaScript. The inset shows a step-by-step example of a user combing blocks to evaluate them.

choices around abstraction and representation in software. Specifically, we will describe three case studies which compare a conventional use of abstraction against a critical one. One case study involves a programming interface, while the other two focus on explanations of computation. For each of the new abstractions, we will contextualize it in existing discourse in art and media theory.

Block-based programming playgrounds. Scratch (Figure 9) and Reduct (Figure 10) are both block-based programming systems for learning programming. Scratch [75] is an open-ended playground for computing, while Reduct [4] is a puzzle-based video game designed with a "comprehension-first approach" to teach a curated set of core programming concepts (e.g. Booleans, conditionals, mapping functions over sets). Reduct uses the same visual metaphor of blocks as Scratch to represent code, but it radically branches out from how Scratch has conceptualized programming for learners.

Scratch is a comprehensive pedagogical tool for programming. Learners create reusable instruction-structures by manipulating and joining instruction code fragments (represented as manipulable blocks). Each code fragment has a dedicated visual style, which utilizes the 'matching shapes' metaphor of puzzle pieces or LEGO blocks. Learners can deduce syntactic compatibility of code blocks visually instead of memorizing syntax rules. Scratch, like many programming environments, frames programming as an activity of building persistent syntax-structures that can be easily reused and executed.

In contrast, Reduct focuses on a particular aspect of programming: executing code, and it does so by transforming blocks within the rules set by the programming language's operational semantics. In Reduct, each code block does not represent a reusable piece of code (like in Scratch), but instead it is a snapshot of program state that is expected to be evaluated and resolved into another state. Through Reduct, a learner can become intimately familiar with the operational semantics of the machine since they are actively directing the computation.

This, by design, omits the usual aspiration for programming to be a building site for instruction-structures with 'wide wall' and 'high ceiling.' Instead, a user of Reduct evaluates program state under a given set of rules and transforms it until they reach a goal. Through this process, the earlier program states are evaluated and irretrievably lost.

As such, Reduct suggests *degradation* as a mode of abstraction for representing computational processes under the hood. Degradation is a lossy form of abstraction where a material wears away over time. Degradation reflects how the material was constructed (physically or artificially) and how it is situated in an environment. In media theory, Alexander Galloway and Jason LaRivière [47] explore abstraction as a particular process and contrasts it with degradation to critique the digital world's fixation on lossless data transfer and representation. Where digital media with "lossless" compression succeeds by becoming invisible, lossy representations make the structure of their medium legible. Artist Hito Steyerl describes how images circulating online accumulate noise and artifacts ("poor image", see Figure 1, ❶) and how this degradation surfaces the 'physics' and intentions underlying the data infrastructure — an algorithm that prioritizes "velocity, intensity, and spread" [103]. Galloway and LaRivière further reclaim data loss, erasure, and decay as a mode to "refuse" surveillance and to being reduced for extraction [47].

Deep Neural Network (DNN) explanations. Figure 11 and Figure 12 illustrate a data acquisition pipeline for collecting images for a large image dataset (LAION-5b [97]).

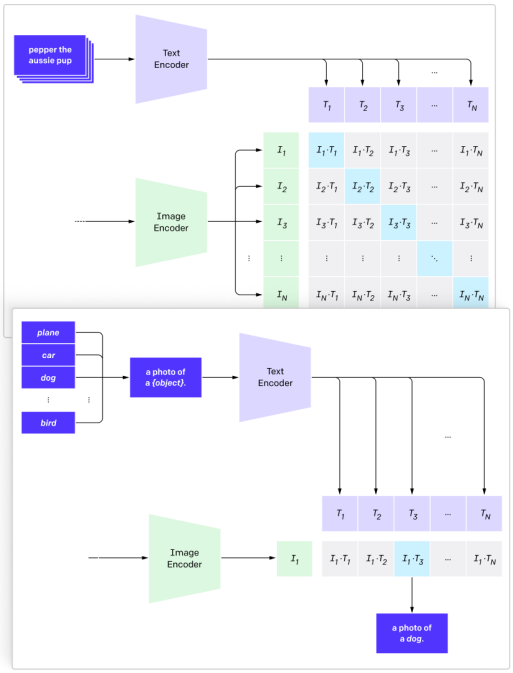


Image: ©OpenAI

Figure 11. A diagram explaining the architecture of OpenAI's *CLIP: Connecting text and images* [85] model.

Image: Buschek & Thorp / Courtesy of the artists

Figure 12. *Models All The Way Down* [19] is an interactive visual story explaining the construction of a large image dataset (LAION-5B) by Christo Buschek and Jer Thorp from Knowing Machines project. As a reader scrolls down the webpage, the narrative unfolds along the visualizations of sample data from the dataset.

Both communicate CLIP (Contrastive Language-Image Pre-training) [92], which is used to filter the dataset. However, they have clearly contrasting goals: the model diagram is from a communication channel [85] to introduce and promote a new commercial product, and the visual story [19] is from a multidisciplinary research team focused on a critical study of machine learning.

The model architecture in Figure 11 does not display the concrete images being operated on and instead emphasizes the mathematical construction of the model using symbolic notations (*e.g.*, matrices, etc.). It aggregates and *generalizes* raw web-scraped data into clean components of a functional diagram, foregrounding the effect of the process over individual data or actual techniques involved in the machine learning process. We see this sanitation via abstraction extend to the design of modern GenAI user interfaces where the noisy quality of internet-scraped data is papered over by an unassuming chat-bot user interface.

In contrast, in *Models All The Way Down* [19], as a reader scrolls through the sections of the article, a few dozen samples from the five billion images are dynamically re-sampled, re-arranged, and re-rendered. Instead of merging the data across various qualities and origins, each raw image is shown individually. As a result, the noise, removed context, and vast scale of the machine learning dataset become more salient.

Models All The Way Down recruits *assemblage* as a primary mode of abstraction. An assemblage is a collection of heterogeneous entities whose meaning is defined by relations among the constituent entities (see Figure 1, ❷) and is always free to recombine again and change its nature. The term assemblage was coined by Deleuze and Guattari [32] and has generated useful interpretations of abstraction. For example, Nail [81] invokes assemblages as "the rejection of unity in favor of multiplicity, and the rejection of essence in favor of events." By having "no eternally necessary defining feature," [81] *i.e.*, essence, that unites constituent entities, an assemblage resists aggregation. In addition to these implications for data visualization and Explainable AI, assemblage could offer us better ways to understand "types" [87], by framing them as dynamic, composable configurations, rather than fixed, monolithic categories.

Network protocols. Figure 13 and Figure 14 illustrate network protocols. The timing diagram in Figure 13 *decontextualizes temporal features* of network activities and idealizes them into a linear sequence of communications. Here, abstraction is used to prioritize temporal legibility, making the sequential nature of interactions immediately recognizable through clear phase demarcation and duration indicators. However, this temporal focus excludes aspects of agential interaction that cannot be captured through sequential models, such as simultaneity, overlap, and contextual richness.

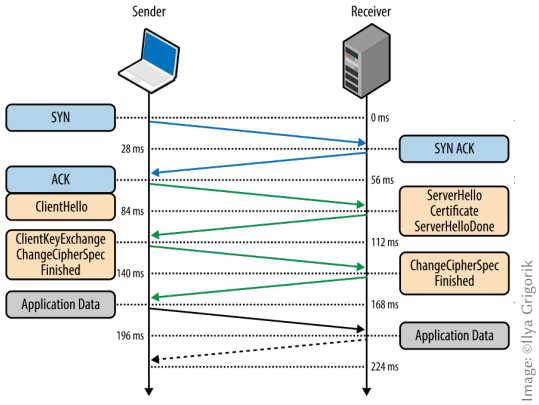


Figure 13. A timing diagram from *High Performance Browser Networking* textbook [54] on TLS handshake protocol.

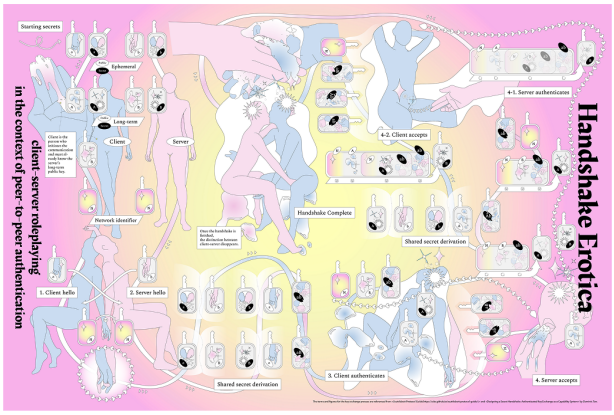


Figure 14. *Handshake Erotica* [67] created by Nahee Kim for SecureScuttleButt (SSB) gathering depicting the Secret Handshake cryptographic protocol for SSB.

In contrast, *Handshake Erotica* (Figure 14) anthropomorphizes protocol interactions between a client and a server as an intimate interaction between two people. *Handshake Erotica* is provocative and open to interpretation, valuing imaginative engagement over technical fidelity or clarity. It selects a culturally rich metaphor — sexuality — to reframe network security from the perspective of intimacy and trust-building.

We suggest that *Handshake Erotica* uses *metaphor* as its mode of abstraction, a conceptual method frequently used in art (also see Figure 1, ❸). Shifting a familiar metaphor to an unfamiliar one lets us reimagine decontextualized programming concepts and operations as rooted in the human world. The lambda calculus, for instance, is not merely a formalism, but can be grounded in a human approach (*e.g.*, how we tend to reason with container schemas or directionality) to mathematical problems [88]. In cognitive sciences, Gentner [49] define conceptual metaphors as entities that map from one structural domain to another, making metaphorical transformation and analogical reasoning possible. Embodied cognitivists, (*i.e.*, those who believe that all experiences are grounded in sensory input) also often use metaphorical mappings as the vehicle through which abstract reasoning is possible. For example, to experience *affection* (abstract) in terms of *warmth* or thermic experience (concrete), like in "send her a warm hello" [69].

Summary. We looked at three modes of abstraction that diverge from the conventional understanding of symbolic abstraction. *Degrade* challenges how symbolic abstraction strives for functionally lossless compression through prioritizing and extracting a certain feature or structure. So it asks what we mean by the permanence, fidelity, and integrity of our representation. *Assemblage* challenges the assumption that instances can be unified through a new structure. Where symbolic abstraction generalizes over a collection, assemblage asks when it is actually necessary to clump instances together and classify or parameterize them. Or, how the individual qualities of those instances remain visible and recognizable to end-users once they have been absorbed into a collection. *Metaphor* challenges the assumption that mapping needs to be explicitly defined. In symbolic abstraction, valid moves are fixed by the syntax and semantics of the abstraction itself. Metaphor instead asks how underspecified, accidental, intuitive mappings across semantic domains might be possible within software artifacts.

The artifacts in our case studies are discussed as critical software artifacts, some following the tradition of *marginal art* forms [20, 60] like "theater sets, street decorations, typography, photomontage, and clothing design – arts exempt from the structural and functional constraints of building," [60] which continued to challenge, parody, and subvert. For critical software artifacts these forms include diagrams, visual stories, games, speculative designs, instructions, mock ups, languages, and various forms of systems art. This vision of critical software is not entirely novel. It is a continuation and extension of the previous works including art forms such as esoteric programming languages [106], performative social network [29], community-built tools [93], transitional systems [1], and rhetorical software [33].

5 Grounds for Critical Abstraction

The artifacts juxtaposed in the previous section show that abstraction in computing reflects and enacts a specific set of premises. Every mode of abstraction occupies a particular perspective that decides how something is represented, circulated, arranged, and transformed through the computational system we have constructed. The politics of computing artifacts have a sustained history across science and technology studies (STS), media theory, and infrastructure and platform studies. Human-computer interaction scholars

have approached politics of mediation and computing from many angles: from how agency is negotiated [35, 104] to power relations embedded in technological interventions [7, 59] to generative design methods to critique these existing structures and values [2, 90, 98]. Two recent pieces, one from creativity support tools [71] and one from programming languages (PL) [56] help situate abstraction within power structures in computing and pose a set of generative questions.

Artifacts from the Culture of Domination. Hermans and Schlesinger [56] begins with a personal account of "not feeling at home" in the community — a feeling of *otherness*. By applying feminist theories [21, 28], they scrutinize the cultural norms of academic PL and connect them to exclusion: formal and quantitative approaches are prioritized, while the qualitative, humanistic, and design methods have been culturally and systematically excluded. In other words, there is a tendency to "classify, measure, map, and, ideally, dominate, and control" [18], which is explicitly reflected in the call-for-papers evaluation guidelines' emphasis on quantitative thinking and formal proofs.

Programming languages are a type of abstraction that, in part, have created exclusion in the field. Their focus on formalism not only marginalizes groups of people, but also limits and excludes *what counts as programming* and computing. Less abstract activities, like spreadsheet use, or weaving [3, 55], are traditionally excluded from programming since they are not perceived as "hard science" [56]. The choice of abstraction thus structures the power relations in the field, and in turn, forms a value system that excludes ways of knowing. This suggests a few questions: What new directions might CS research include once the assumptions of symbolic abstraction are loosened? And, how do our criteria of evaluation change to remain porous to the pluralistic epistemic perspectives such work would invite?

Normative Ground. Li et al. [71] describe how power relations are formed in creativity support tools between users and designers by evoking the notion of a *normative ground*. A normative ground defines abstractions that shape the 'ground' a user acts on. This is constructed by a network of material and cultural participants (*e.g.*, "mental models, documentation, online communities, licensing models, intended usage cases, cultural norms, ...") surrounding an artifact, originating from a designer's prenotion of "how someone should or could think, act, and express themselves." [71].

By interviewing artists and tool designers, they demonstrate how a normative ground sets two distinct types of power ("*power-to*" as in how a user can achieve through a tool versus "*power-over*" as in how a tool shapes user thoughts and actions [96]). Specifically, they argue that: (1) increasing the level of abstraction simultaneously increases capabilities while narrowing the bandwidth of *power-to* and (2) users are empowered when switching between different implementations of abstraction (*i.e.*, switching out of a normative ground). They thus offer a practical call-to-action by encouraging tools to enable movement of abstraction "up and down the abstraction ladder" [71] and jumping across different "ladders". While practical, this emphasis on the metaphor of vertical movement may again overlook directions beyond standard CS notions of abstraction, as well as the opportunity for broader participant involvement in new interpretations. This poses a further set of questions: What artifacts form the normative ground in computing and how do their modes of abstraction stabilize or destabilize it? What critical artifacts might unsettle that ground, created by and for whom?

Generative Ambiguity of Abstraction in Art. Art can, again, provide a precedent for answering questions regarding the pluralization of abstraction posed earlier. First, the ambiguity of the term 'abstraction' has been supported as a generative condition in art inside and outside institutions. In 1936, Alfred H. Barr Jr., a highly influential American art historian and the first director of the Museum of Modern Art (MoMA), wrote "ambiguity of the word abstract as applied to works of art is useful" [8] as a guide for critical understanding. The subsequent framing of abstract painting as "pure-abstract"[8] rejecting representational art (as discussed in §3) was possible because the term was allowed to remain unsettled. This new definition for abstract soon became unstable as the generation following the abstract painting of the 1950s began questioning the "purity" of visual form. For example, Cy Twombly's painting emphasized gesture and mark-making over pure form and, more broadly, visual artists such as Cindy Sherman challenge the notion of pure form through performing abject images ("formless body") to attack "structured," ideal body images [15]. Moreover, reflections on art institutions as perspective-producing systems have made way for *critiques of the systems and their mechanisms*[7] [46].

Artists across global scenes have engaged with interwoven questions concerning what makes art abstract: its forms, concepts, subject matter, or organizational logic. One curatorial framework, presented by curator Maria Lind [73], distinguishes three strands of abstraction since the late 1950s: formal abstraction withdraws from representing physical world and moves toward *creating new forms with aesthetic and structural visual codes* (Figure 4); economic abstraction treats the *abstraction of concrete relations into economic operations* (capital, finance, and communication) as its artistic subject [16] (Figure 15); social abstraction is performative and organizational, enacting a *strategic withdrawal* from dominant

[7] An example of an institutional critique is Andrea Fraser's *Museum Highlights: A Gallery Talk* (1989) is a satire performance where Fraser plays an archetype museum docent next to an ordinary to comment on the power relation between the museum and the viewers.

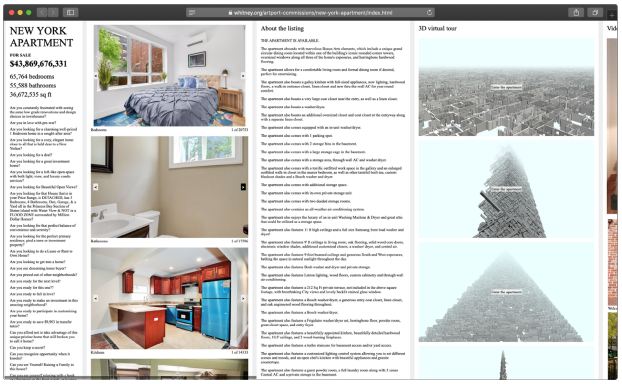


Figure 15. In Tega Brain and Sam Lavigne's *NY Apartment* (2020), online New York City apartment rental postings are scraped and assembled into one fictional mega-structure[16]. This is an example of economic abstraction revealing the tropes of how rental properties are represented and marketed on web.

Figure 16. Zach Blas' *Facial Weaponization* suites (2012-2014) [13] feature masks created from aggregating biometric facial data of a demographic of people. This is an example of social abstraction suggesting a strategy for historically over-surveilled populations to withdraw from facial recognition technology [12, 14].

institutional structures [13] toward self-organized or collectively autonomous forms of practice (Figure 16). These works continue to question the conceptions of abstraction that have become normative in art and unsettle that ground through provocation. We can see how abstraction in art remains unstable and open to ongoing reinterpretation through a new set of questions when any single conception of abstraction begins to settle.

Questions toward Critical Software Artifact. How do we reinterpret a fundamental concept like abstraction? This is especially difficult when the current norms around abstraction constrain not only what we do but foreclose what we can imagine doing otherwise [77]. On top of that, software has become extensive media that spans interface to infrastructure, constructed with arrangements of diverse human and nonhuman actors situated in different technical, economic, social, cultural, and political contexts. It would be unproductive (and even harmful) to propose a unifying framework of abstraction for computing or software.

Yet we *can* pose questions for orienting toward *criticism from within* [99]: a critique of abstraction, embedded in the software itself and extended through *in vivo* systemic intervention. One way critical software practice can begin is by examining the assumption behind the normative notion of abstraction. In our analysis of software artifacts (§4), we surfaced how a specific mode of abstraction rests on three assumptions: that functionally essential features or structure can be *compressed* without loss, that instances can be *collected* under a unifying structure, and that representations are *transformed* following formally defined syntax and semantics.

If we connect these assumptions to how symbolic abstraction operates, we can surface generative questions towards designing critical software artifacts. *Compression* operates *extractively* by deciding what counts as essential across contexts. Which aspects does abstraction then extract, model, and structure as reusable across contexts? What does this extractive work enable, with what risks? *Collection* operates *simulatively* by producing a new entity that stands in for the individual qualities it absorbs. What aspects of computing then become amplified as a result of this abstraction? What does the abstraction fabricate or simulate about computation? *Mapping* operates *prescriptively* by deciding which transformations are legitimate. How does the abstraction then reify normative rules and constraints? How does it draw boundaries in other communicative systems, such as spoken language or drawings?

We would like to leave with these questions as a starting point for imagining how abstractions may drastically vary and give way to possible meanings of abstraction, while remaining closely engaged with everyday practices in the field.

Acknowledgments

We are grateful to artists and organizations who permitted us to use their artwork in this paper. We thank the anonymous reviewers for their thorough reading and constructive feedback. We also thank Ian Arawjo for his valuable comments on the earlier versions of this manuscript. AMC thanks Unmake Lab (Binna Choi and Sooyon Song), Min-Hyung Kang, Seungbum Kim, and the participants of Forking Room 2026 for engaging with the core arguments of this work and providing feedback. TP thanks Pierre Depaz for numerous interesting discussions about architecture and abstraction and attendees of his talk at PPIG 2025 in Belgrade. TP was supported by the Charles University grant PRIMUS/24/SCI/021.

References

[1] Roel Roscam Abbing. 2021. 'This is a solar-powered website, which means it sometimes goes offline': a design inquiry into degrowth and ICT. In *Workshop on Computing within Limits*.

[2] Philip Agre. 1997. *Computation and human experience*. Cambridge University Press, Cambridge, UK.

[3] Lea Albaugh, James McCann, Lining Yao, and Scott E. Hudson. 2021. Enabling personal computational handweaving with a low-cost jacquard loom. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 1–10. doi:10.1145/3411764.3445750

[4] Ian Arawjo, Cheng-Yao Wang, Andrew C. Myers, Erik Andersen, and François Guimbretière. 2017. Teaching Programming with Gamified Semantics. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. Association for Computing Machinery, New York, NY, USA, 4911–4923. doi:10.1145/3025453.3025711

[5] Inke Arns. 2005. Code as performative speech act. *Artnodes* 4 (2005), 1–9. doi:10.7238/a.v0i4.727

[6] Pier Vittorio Aureli. 2023. *Architecture and Abstraction*. The MIT Press, Cambridge, MA.

[7] Shaowen Bardzell. 2010. Feminist HCI: Taking stock and outlining an agenda for design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 1301–1310. doi:10.1145/1753326.1753521

[8] Alfred H. Jr. Barr. 1936. *Cubism and Abstract Art*. The Museum of Modern Art, New York, NY.

[9] Paolo Belardi. 2024. Idea as Model, Model as Idea. The Axonometric Model of House X by Peter Eisenman. *diségno* 14 (2024), 23–26.

[10] Emily M. Bender and Alexander Koller. 2020. Climbing towards NLU: On meaning, form, and understanding in the age of data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Stroudsburg, PA, 5185–5198. doi:10.18653/v1/2020.acl-main.463

[11] Jeffrey M Binder. 2022. Language and the Rise of the Algorithm. In *Language and the Rise of the Algorithm*. University of Chicago Press, Chicago, IL. doi:10.7208/chicago/9780226822549.001.0001

[12] Zach Blas. 2012. Fag Face Mask - October 20, 2012, Los Angeles, CA. Painted, vacuum-formed recycled polyethylene terephthalate, from *Facial Weaponization Suite*. Courtesy of the artist.

[13] Zach Blas. 2012–2014. Facial Weaponization Suite. Mixed media installation. https://zachblas.info/works/facial-weaponization-suite/

[14] Zach Blas. 2014. Feminist Imperceptibilities - April 25, 2014, New York, NY. Conversation performance, from *Facial Weaponization Suite*. Courtesy of the artist.

[15] Yve-Alain Bois and Rosalind E. Krauss. 1997. *Formless: A User's Guide*. Zone Books, New York.

[16] Tega Brain and Sam Lavigne. 2020. NY Apartment. Web exhibition. https://whitney.org/exhibitions/new-york-apartment

[17] Andrea Branzi and Archizoom Associati. 1967. No-Stop City. Architectural drawings.

[18] Holly Jean Buck, Andrea R. Gammon, and Christopher J. Preston. 2014. Gender and geoengineering. *Hypatia* 29, 3 (2014), 651–669. doi:10.1111/hypa.12083

[19] Christo Buschek and Jer Thorp. 2024. Visual Story Models All The Way Down. https://knowingmachines.org/models-all-the-way. Accessed: August 26, 2026.

[20] Michael Camille. 1992. *Image on the edge: The margins of medieval art*. Reaktion Books, London, UK.

[21] Mark Carey, M Jackson, Alessandro Antonello, and Jaclyn Rushing. 2016. Glaciers, gender, and science: A feminist glaciology framework for global environmental change research. *Progress in Human Geography* 40, 6 (2016), 770–793. doi:10.1177/0309132515623368

[22] Ildefons Cerdà. 1859. Plan de los alrededores de la ciudad de Barcelona y del proyecto para su mejora y ampliación. Map, Museu d'Història de la Ciutat, Barcelona.

[23] Wendy Hui Kyong Chun. 2005. On software, or the persistence of visual knowledge. *Grey Room* 18 (2005), 26–51. doi:10.1162/1526381043320741

[24] Wendy Hui Kyong Chun. 2008. On "Sourcery," or Code as Fetish. *Configurations* 16, 3 (2008), 299–324. doi:10.1353/con.0.0064

[25] Wesley A. Clark. 1955. *The Logical Structure of Digital Computers*. Technical Report 6M-3938. MIT Lincoln Laboratory, Division 6, Cambridge, MA.

[26] Timothy R. Colburn and Gary M. Shute. 2007. Abstraction in computer science. *Minds and Machines* 17, 2 (2007), 169–184. doi:10.1007/s11023-007-9061-7

[27] Timothy R Colburn and Gary M Shute. 2008. Metaphor in computer science. *Journal of applied logic* 6, 4 (2008), 526–533. doi:10.1016/j.jal.2008.09.005

[28] Patricia Hill Collins. 2022. *Black feminist thought: Knowledge, consciousness, and the politics of empowerment*. Routledge, London, UK.

[29] Geoff Cox and Alex McLean. 2012. *Speaking code: Coding as aesthetic and political expression*. MIT Press, Cambridge, MA. doi:10.7551/mitpress/8193.001.0001

[30] Francesco Dal Co. 2016. *Centre Pompidou: Renzo Piano, Richard Rogers, and the Making of a Modern Monument*. Yale University Press, New Haven, CT.

[31] Liesbeth De Mol and Maarten Bullynck. 2022. What's in a Name? Origins, Transpositions, and Transformations of the Triptych Algorithm–Code–Program. In *Abstractions and Embodiments: New Histories of Computing and Society*, Janet Abbate and Stephanie Dick (Eds.). Johns Hopkins University Press, Baltimore, MD, 141–162.

[32] Gilles Deleuze and Félix Guattari. 1987. *A Thousand Plateaus: Capitalism and Schizophrenia*. University of Minnesota Press, Minneapolis, MN.

[33] Pierre Depaz. 2026. Global and Local Implications of Computational Artifacts. *Philosophia Scientiæ* 30, 2 (2026), 65–79. doi:10.4000/16955

[34] Leah Dickerman, Yve-Alain Bois, Masha Chlenova, Christoph Cox, Hal Foster, Mark Franko, Peter Galison, Philippe-Alain Michaud, Sam Cate-Gumpert, and R. H. Quaytman. 2013. Abstraction, 1910–1925: Eight Statements. *October* 143 (2013), 3–51. doi:10.1162/OCTO_a_00130

[35] Paul Dourish. 2017. *The Stuff of Bits: An Essay on the Materialities of Information*. MIT Press, Cambridge, MA. doi:10.7551/mitpress/10999.001.0001

[36] Johanna Drucker. 2014. *Graphesis: Visual Forms of Knowledge Production*. Harvard University Press, Cambridge, MA.

[37] Peter Eisenman. 1975. House X (Project, Bloomfield Hills, Michigan). Architectural drawings, diagrams, and models. Peter Eisenman fonds, Canadian Centre for Architecture.

[38] Peter Eisenman. 1982. *House X*. Rizzoli International Publications, New York, NY.

[39] Peter Eisenman. 2006. *The Formal Basis of Modern Architecture: Dissertation 1963, Facsimile*. Lars Müller Publishers, Baden.

[40] Calvin C. Elgot. 1959. Review: John W. Carr III, Languages, Logic, Learning, and Computers. *The Journal of Symbolic Logic* 24, 3 (1959), 222. doi:10.2307/2963859

[41] David Elliott. 2013. Generation Loss. Video. Available at https://vimeo.com/3750507 (Accessed: August 26, 2026).

[42] David Elliott. 2013. Generation Loss. Internet Archive. Archived July 18, 2013. Available at https://web.archive.org/web/20130718180736/http://hadto.net/sketchbook/generation-loss/ (Accessed: August 26, 2026).

[43] Luciano Floridi. 2008. The Method of Levels of Abstraction. *Minds and Machines* 18, 3 (2008), 303–329. doi:10.1007/s11023-008-9113-7

[44] Luciano Floridi and J. W. Sanders. 2004. *The Method of Abstraction*. Research Report 4. Information Ethics Research Group (IEG). doi:10.2139/ssrn.3920316

[45] George E. Forsythe. 1965. G. E. Forsythe Talk at Swarthmore College, 16 Apr. 1965. (1965). George Forsythe Papers, 1938–1972 (Series 1), Box 4, Folder 21. Stanford Digital Repository, Stanford University. http://purl.stanford.edu/sz985kd8531

[46] Andrea Fraser. 2005. From the Critique of Institutions to an Institution of Critique. *Artforum* (September 2005), 278–283. https://www.artforum.com/features/from-the-critique-of-institutions-to-an-institution-of-critique-172201/

[47] Alexander R. Galloway and Jason R. LaRivière. 2017. Compression in philosophy. *boundary 2* 44, 1 (2017), 125–147. doi:10.1215/01903659-3725905

[48] Jean-Gabriel Ganascia. 2015. Abstraction of levels of abstraction. *Journal of Experimental & Theoretical Artificial Intelligence* 27, 1 (2015), 23–35. doi:10.1080/0952813X.2014.940685

[49] Dedre Gentner. 1983. Structure-mapping: A theoretical framework for analogy. *Cognitive Science* 7, 2 (1983), 155–170. doi:10.1207/s15516709cog0702_3

[50] Dedre Gentner and Jennifer Asmuth. 2019. Metaphoric extension, relational categories, and abstraction. *Language, Cognition and Neuroscience* 34, 10 (2019), 1298–1307. doi:10.1080/23273798.2017.1410560

[51] Félix González-Torres. 1991. "Untitled" (Portrait of Ross in L.A.). Installation. The Art Institute of Chicago, Chicago, IL.

[52] Saul Gorn. 1982. Informatics (Computer and Information Science: Its Ideology, Methodology, and Sociology. *Knowledge* 4, 2 (1982), 173–198. doi:10.1177/107554708200400203

[53] Arie Graafland. 2007. Book Review: Peter Eisenman, 'The Formal Basis of Modern Architecture'. *Footprint* 1, 1 (2007), 93–96.

[54] Ilya Grigorik. 2013. *High Performance Browser Networking: What every web developer should know about networking and web performance.* O'Reilly Media, Sebastopol, CA. Accessed: August 26, 2026.

[55] Ellen Harlizius-Klück. 2017. Weaving as Binary Art and the Algebra of Patterns. *Textile* 15, 2 (2017), 176–197. doi:10.1080/14759756.2017.1298239

[56] Felienne Hermans and Ari Schlesinger. 2024. A Case for Feminism in Programming Language Design. In *Proceedings of the 2024 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! '24) (Onward! '24).* Association for Computing Machinery, New York, NY, USA, 205–222. doi:10.1145/3689492.3689809

[57] Edwin L. Hutchins, James D. Hollan, and Donald A. Norman. 1985. Direct manipulation interfaces. *Human–Computer Interaction* 1, 4 (1985), 311–338. doi:10.1207/s15327051hci0104_2

[58] International Business Machines Corporation. 1960. *General Information Manual: IBM Commercial Translator.* Number Form F28-8043 in IBM Systems Reference Library. International Business Machines Corporation, White Plains, NY.

[59] Lilly Irani, Janet Vertesi, Paul Dourish, Kavita Philip, and Rebecca E. Grinter. 2010. Postcolonial computing: A lens on design and development. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* Association for Computing Machinery, New York, NY, USA, 1311–1320. doi:10.1145/1753326.1753522

[60] Philip Johnson and Mark Wigley. 1988. *Deconstructivist Architecture.* The Museum of Modern Art, New York, NY.

[61] Wassily Kandinsky. 1947. *Point and Line to Plane: Contribution to the Analysis of the Pictorial Elements.* Solomon R. Guggenheim Foundation for the Museum of Non-Objective Painting, New York. Digitized and available electronically via the Internet Archive: https://archive.org/details/pointlinetoplane00kand.

[62] Stephen Kell. 2009. The mythical matched modules: overcoming the tyranny of inflexible software construction. In *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications.* Association for Computing Machinery, New York, NY, 881–888. doi:10.1145/1639950.1640051

[63] Stephen Kell. 2014. In Search of Types. In *Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software (Onward!).* Association for Computing Machinery, New York, NY, USA, 227–241. doi:10.1145/2661136.2661154

[64] Doyun Kim. 2022. Translated Vase: A Memoir of Korean Ceramics. https://www.artic.edu/articles/984/translated-vase-a-memoir-of-korean-ceramics. Accessed: August 26, 2026.

[65] Jaegwon Kim. 1993. The Nonreductivist's Troubles with Mental Causation. In *Supervenience and Mind: Selected Philosophical Essays.* Cambridge University Press, Cambridge, 336–357. doi:10.1017/CBO9780511625220.018

[66] Jaegwon Kim. 2002. The Layered Model: Metaphysical Considerations. *Philosophical Explorations* 5, 1 (2002), 2–20. doi:10.1080/10002002018538719

[67] Nahee Kim. 2018. Handshake Erotica. https://dev.scuttlebutt.nz/assets/handshake-erotica.png. Archived on Scuttlebutt Treasure Map. Accessed: August 26, 2026.

[68] Stephen Cole Kleene. 1956. Representation of events in nerve nets and finite automata. In *Automata Studies*, Claude E. Shannon and John McCarthy (Eds.). Number 34 in Annals of Mathematics Studies. Princeton University Press, Princeton, NJ, 3–41. doi:10.1515/9781400882618-002

[69] George Lakoff and Mark Johnson. 1980. The metaphorical structure of the human conceptual system. *Cognitive Science* 4, 2 (1980), 195–208. doi:10.1207/s15516709cog0402_4

[70] George Lakoff and Rafael Núñez. 2000. *Where mathematics comes from: How the embodied mind brings mathematics into being.* Basic Books, New York.

[71] Jingyi Li, Eric Rawn, Jacob Ritchie, Jasper Tran O'Leary, and Sean Follmer. 2023. Beyond the Artifact: Power as a Lens for Creativity Support Tools. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST '23).* Association for Computing Machinery, New York, NY, USA, Article 47, 15 pages. doi:10.1145/3586183.3606831

[72] Morten Lind. 2003. Making sense of the abstraction hierarchy in the power plant domain. *Cognition, Technology & Work* 5, 2 (2003), 67–81. doi:10.1007/s10111-002-0109-4

[73] Maria Lind (Ed.). 2013. *Abstraction.* MIT Press and Whitechapel Gallery, Cambridge, MA and London.

[74] Barbara Liskov and Stephen Zilles. 1974. Programming with abstract data types. *ACM Sigplan Notices* 9, 4 (1974), 50–59. doi:10.1145/800233.807045

[75] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The Scratch Programming Language and Environment. *ACM Transactions on Computing Education* 10, 4 (2010), 1–15. doi:10.1145/1868358.1868363

[76] Mark C Marino. 2020. *Critical code studies.* MIT Press, Cambridge, MA. doi:10.7551/mitpress/12122.001.0001

[77] Annette Markham. 2021. The limits of the imaginary: Challenges to intervening in future speculations of memory, data, and algorithms. *New media & society* 23, 2 (2021), 382–405.

[78] Ron McClamrock. 1990. Marr's three levels: A re-evaluation. *Minds and Machines* 1, 2 (1990), 185–196. doi:10.1007/BF00361036

[79] Warren S. McCulloch and Walter Pitts. 1943. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics* 5, 4 (1943), 115–133. doi:10.1007/BF02478259

[80] Claudio Mirolo, Cruz Izu, Violetta Lonati, and Emanuele Scapin. 2022. Abstraction in computer science education: An overview. *Informatics in Education* 20, 4 (2022), 615–639. doi:10.15388/infedu.2021.27

[81] Thomas Nail. 2017. What is an Assemblage? *SubStance* 46, 1 (2017), 21–37. doi:10.3368/ss.46.1.21

[82] Allen Newell and Herbert A Simon. 1976. Computer science as empirical inquiry: Symbols and search. *Commun. ACM* 19, 3 (1976), 113–126. doi:10.1145/360018.360022

[83] David Nofre. 2023. "Content Is Meaningless, and Structure Is All-Important": Defining the Nature of Computer Science in the Age

of High Modernism, c. 1950–c. 1965. *IEEE Annals of the History of Computing* 45, 2 (2023), 29–42. doi:10.1109/MAHC.2023.3266359

[84] David Nofre, Mark Priestley, and Gerard Alberts. 2014. When Technology Became Language: The Origins of the Linguistic Conception of Computer Programming, 1950–1960. *Technology and Culture* 55, 1 (2014), 40–75. doi:10.1353/tech.2014.0031

[85] OpenAI. 2021. CLIP: Connecting text and images. https://openai.com/index/clip/. Accessed: August 26, 2026.

[86] David Lorge Parnas. 1972. On the criteria to be used in decomposing systems into modules. *Commun. ACM* 15, 12 (1972), 1053–1058. doi:10.1145/361598.361623

[87] Tomas Petricek. 2015. Against a Universal Definition of 'Type'. In *Proceedings of the 2015 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!)*. Association for Computing Machinery, New York, NY, USA, 254–266. doi:10.1145/2814228.2814249

[88] Tomas Petricek. 2018. Would aliens understand lambda calculus? http://tomasp.net/blog/2018/alien-lambda-calculus/

[89] Tomas Petricek. 2026. *Cultures of Programming: The Development of Programming Concepts and Methodologies*. Cambridge University Press, Cambridge, UK. doi:10.1017/9781009492379

[90] James Pierce. 2021. In tension with progression: Grasping the frictional tendencies of speculative, critical, and other alternative designs. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, Article 617, 19 pages. doi:10.1145/3411764.3445406

[91] James Pierce, Phoebe Sengers, Tad Hirsch, Tom Jenkins, William Gaver, and Carl DiSalvo. 2015. Expanding and refining design and criticality in HCI. In *Proceedings of the 2015 CHI Conference on Human Factors in Computing Systems (CHI '15)*. Association for Computing Machinery, New York, NY, USA, 2083–2092. doi:10.1145/2702123.2702438

[92] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. In *Proceedings of the 38th International Conference on Machine Learning (PMLR, Vol. 139)*. PMLR, 8748–8763. doi:10.5555/3514013.3514101

[93] Casey Reas and Ben Fry. 2018. A Modern Prometheus: The History of Processing. Medium. Retrieved August 26, 2026 from https://medium.com/processing-foundation/a-modern-prometheus-59aed94abe85

[94] Warren Sack. 2025. Interfacing Programming Language Semantics and Pragmatics: What Does "Hello, World" Mean? *Philosophies* 10, 4 (2025), 86. doi:10.3390/philosophies10040086

[95] Jonathan Schaffer. 2003. Is There a Fundamental Level? *Noûs* 37, 3 (2003), 498–517. doi:10.1111/1468-0068.00448

[96] Hanna Schneider, 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*. Association for Computing Machinery, New York, NY, USA, 1–14. doi:10.1145/3173574.3173818

[97] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, et al. 2022. Laion-5b: An open large-scale dataset for training next generation image-text models. *Advances in neural information processing systems* 35 (2022), 25278–25294.

[98] Phoebe Sengers, Kirsten Boehner, Shay David, and Joseph 'Jofish' Kaye. 2005. Reflective design. In *Proceedings of the 4th Decennial Conference on Critical Computing: Between Sense and Sensibility*. Association for Computing Machinery, New York, NY, USA, 49–58. doi:10.1145/1094562.1094569

[99] Jorge Silvetti. 1998. The Beauty of Shadows. In *Oppositions reader: selected readings from a journal for ideas and criticism in architecture, 1973-1984*, K. Michael Hays (Ed.). Princeton Architectural Press, New York, 366–387.

[100] Anders Søgaard. 2023. Grounding the vector space of an octopus: Word meaning from raw text. *Minds and Machines* 33, 1 (2023), 33–54. doi:10.1007/s11023-023-09622-4

[101] Joel Spolsky. 2004. *Joel on Software: And on Diverse and Occasionally Related Matters*. Apress, Berkeley, CA. doi:10.1007/978-1-4302-0753-5

[102] Friedrich Steimann. 2018. Fatal abstraction. In *Proceedings of the 2018 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!)*. Association for Computing Machinery, New York, NY, USA, 125–130. doi:10.1145/3276954.3276966

[103] Hito Steyerl. 2009. In Defense of the Poor Image. *e-flux journal* 10 (2009), 1–9. https://www.e-flux.com/journal/10/61362/in-defense-of-the-poor-image

[104] Lucille A. Suchman. 2007. *Human-Machine Reconfigurations: Plans and Situated Actions* (2nd ed.). Cambridge University Press, Cambridge, UK.

[105] Kumiko Tanaka-Ishii. 2010. *Semiotics of Programming*. Cambridge University Press, Cambridge, UK. https://dl.acm.org/doi/book/10.5555/1805903

[106] Daniel Temkin. 2025. *Forty-Four Esolangs: The Art of Esoteric Code*. The MIT Press, Cambridge, MA.

[107] Giuseppe Terragni. 1936. Casa del Fascio. Building.

[108] Lee Tien. 2000. Publishing Software as a Speech Act. *Berkeley Technology Law Journal* 15, 2 (2000), 629–712. https://www.jstor.org/stable/24116702

[109] Rachel Trusty. 2025. All That Glitters is Not Gold: Viewer Understanding of Felix Gonzalez-Torres's Candy Spill Installations. *Curator: The Museum Journal* 68, 3 (2025), 484–494.

[110] Jan Van Leeuwen. 2014. On Floridi's Method of Levels of Abstraction. *Minds and Machines* 24, 1 (2014), 5–17. doi:10.1007/s11023-013-9321-7

[111] Annette Vee. 2013. Understanding Computer Programming as a Literacy. *Literacy in Composition Studies* 1, 2 (2013), 42–64. https://licsjournal.org/index.php/LiCS/article/view/794

[112] Yeesookyung. 2020. Translated Vase. The British Museum Collection, London. Museum Object ID: A_2020-3019-1. Available at https://www.britishmuseum.org/collection/object/A_2020-3019-1 (Accessed: August 26, 2026).

Received 2026-05-15; accepted 2026-06-28