Software Code as Expanded Narration

Software Code as Expressive Media

Computer programming is about algorithms, formal logic, precise reasoning, and problem solving. As such, it would appear to have little to do with what we call creative expression. The logical act of programming and the self-expressive act of creativity would seem to be mutually exclusive opposites. We think of creative expression as taking place within certain literary or visual arts, through certain kinds of communication or symbolic systems, or via certain kinds of writing, speech, discourse, language, and notation.

Some examples of expressive media and genres are poetry, storytelling, nonsense, humor, musical notation, screenplays, notes for an artwork, notes for a dance or other choreography, and architectural drawings. We do not normally think of software code as an expressive medium. When we think of software code, we think of it as being about how we handle a physical or virtual device that is to be programmed. This is a rational and calculating activity, a practical outcome of the assumptions of the "symbolic logic" of philosophers like Bertrand Russell, Friedrich Ludwig Gottlob Frege, or Noam Chomsky, and of the classical paradigm of computing theory. It is the issuing of a series of instructions or commands to a machine, to an object or mechanism that is regarded as essentially being non-alive.

What is the difference between the expressivity of the expressive act in the literary or visual arts and the series of instructions that is the computer program? We have assumed that there is an insurmountable wall between these two kinds of writing or notation. To tear down this wall, like the Berliners did to their Wall in 1989, was almost unthinkable. Only now are we ready to ask these questions. Could software code also be expressive? Does software code have to be only productive? The idea of expanded narration as applied to software code means going beyond the binary opposition of writing as being either expressive or utilitarian. The paradigm of software code as we know it is reaching its limit: the emphasis on engineering and on getting the program to do something, code as a series of instructions to a machine.

Software development or computer science should begin to concern itself with cultural codes as well as with software codes. Computer science should transform itself into a hybrid engineering and humanities discipline.

Friedrich Kittler: The Numeric Kernel is Decisive

One of the most important media theorists was the German Friedrich Kittler, who is regarded as being a poststructuralist. Kittler resisted expanding media theory to include software theory. Kittler wrote a famous essay called "There Is No Software."[665] His position is that everything in computing breaks down to the digital code of the hardware. There is no going beyond the pervasive logic of the binary. Kittler correctly points out that Turing's computing machine is a reduction of the body of real numbers extant in nature that we call chaos. Yet, in my view, Kittler was wrong because digital-binary logic is not the only possibility for computing. Digital-binary logic is not universal and forever. It is precisely software theory as an academic field, deriving from both media theory and computer science, which leads to new paradigms such as quantum computing, Creative Coding, software that operates like the reverse-engineered human brain, or software as semi-living entities (Artificial Life) rather than inert things to be manipulated by the dominating programmer subject.

In "There is No Software," Kittler fancies himself as writing about the end of history and the end of writing. In a contemporary writing and cultural scene of endlessly expanding and limitless signification, there is ironically an implosion into the no-space and no-time of microscopic computer memory. The relationship of information technology to writing, for Kittler, brings about the situation that he alleges that we do not write anymore. The idea that software code might be a form of writing, a form of *écriture* in the Derridean deconstructionist sense (an intervention or inscription into language that is more fundamental and effective than speech), never occurs to Kittler. He represses this thought and assumes that the computer must bring about the programmatic automation of reading and writing.

"This state of affairs […]," writes Kittler, "hide[s] the very act of writing… We do not write anymore." Kittler believes that writing done on a computer is not an historical act anymore because the writing tools of the computer are able to read and write by themselves.[666] He writes:

> [The] all-important property of being programmable has, in all evidence, nothing to do with software; it is an exclusive feature of hardware, more or less suited as it is to house some notation system. [667]

My view is the opposite of Kittler's. I think that software code can be the site of the re-emergence of *écriture* in Derrida's sense. There can be a shift from programming as the programmability of some device to programming as creativity, creative expression, and writing in the deepest sense of effecting change.

I disagree with Kittler's statement that *there is no software* and his belief that the numerical logic of the low-level hardware that Alan Turing and John von Neumann conceptualized in the 1930s-1940s is determining and decisive. In my view, just because the kernel or center of computing is rational and computational, does not mean that all the other layers, languages, and interfaces of the system, and which surround the kernel, must follow that logic. The education of the people we call programmers is misguided as well, since the curriculum of that training assumes that these persons must be oriented

to logical ratiocination. The education of "humanists" is wrong too – they are supposed to be the opposite of that. We need instead something hybrid or in-between, like computational aesthetics.

Kittler's Media Archaeology

The media theory of Friedrich Kittler is very successful and influential in German universities. Kittler's media archaeology and media historiography have led to the rise of the Berlin School of media theory, of which Wolfgang Ernst is at the forefront.[668] Kittler opposes the so-called discourse analysis of the study of media, which he sees as deriving its methods from hermeneutics and literary criticism. He instead advocates a technical materialism of data storage devices, data transmission, processors, and automatic writing systems, that examines what is claimed to be technologies from within. There is much to respect about Kittler's work. I understand Kittler's body of writing as a valuable contribution to posthumanism and the post-humanities – a gesture that goes beyond the anthropocentric prejudice of placing humans at the center of history and narratives of the future.

My thesis regarding the past and future of informatics is that all layers of the software above the kernel can indeed be anything. Regarding the history and the science fiction futurism of computing, it is a question of studying the past as a succession of cultural theory concepts, and therefore being open to the future of a succession of cultural theory concepts.

A technical layer of conversion between the computational-digital-binary center and the more poetic or human-language discursive applications and interfaces at the periphery and at the outer zones is possible. These mechanisms of translation can exist at a certain specific level of the network architecture. I argue against a dualistic opposition between machinic-computational and poetic-linguistic expression.

Our resistance in ideas to the freeing of software from the kernel of rational-calculating logic is paradoxically an outmoded humanist clinging to the belief in the specialness of humans – the sublime qualities of the soul and consciousness (the Cartesian *cogito*) that humans allegedly have and which we claim that machines do not have.

Derrida radicalized Saussure's semiotics when he said that there are endless chains of signification in sign systems, and not just a one-to-one static relationship between the signifier and the signified. Linguistic signs always refer to other signs, and there can never be a sign that is the endpoint of signification. One never arrives at any ultimate meaning of a word. Writing in Derrida's sense is the opposite of the system of stabilized and clear-cut definitions of words which dictionaries are intended to be. There is always an insurmountable gap between what I write or say and what my readers or listeners read or hear.

For Kittler, the software space is just virtuality or simulation. It is not possible, according to him, to establish a new relationship to the world through software programming, any aesthetically coded transformation. Art/aesthetics/design and informatics have no possible bridge between them. The miniaturization of hardware is, for Kittler,

the proper dimension of simulation, of our postmodern writing scene which is no longer a scene of writing.

Baudrillard went beyond this nostalgic position vis-à-vis technology by practicing photography as a meditation on the technical imaging media itself. He wrote of photography as *the writing of light* – a practice of an exemplary media technology as a form of writing.[669]

For Kittler, hardware always precedes and determines software. He writes:

> There are good grounds to assume the indispensability and, consequently, the priority of hardware in general… All code operations come down to absolutely local string manipulations, that is, I am afraid, *to signifiers of voltage differences*… The so-called philosophy of the so-called computer community tends systematically to obscure hardware with software, electronic signifiers with interfaces between formal and everyday languages.[670]

The combinatorial logic always wins out and the software can do nothing more than tweak bits and bytes. The software industry is one giant conspiracy to hide the machine from its user. The solution, for Kittler, is to write programs in low-level assembler code to maintain awareness that the hardware is what the program always resolves itself into in the end.

The original conception of the digital-binary computer was made in the 1930s and around the time of the Second World War by major figures in the history of ideas such as Alan Turing, John von Neumann, and Claude Shannon. Turing first conceptualized the computer in his 1936 paper "On Computable Numbers with an Application to the *Entscheidungsproblem*."[671] He developed the idea of *the Turing machine*: the mathematical model of computation where a mechanism moves above an infinitely long tape, stops over one cell, reads the symbol written in that cell, and changes the symbol to another symbol chosen from a small set of possible symbols. The control mechanism then moves to another cell to carry out the next operation, manipulating symbols according to a table of rules, simulating the logic of any algorithm that is thus proven to be computable or calculable.

The Turing machine is contemporaneous with the idea of representing instructions and data as finite sequences of binary numbers. Von Neumann is credited with the innovation of the stored-program concept.[672] Shannon achieved the breakthroughs in electrical engineering that Boolean algebra can be deployed to realize digital electronic circuitry and that binary electrical switches can support fast algebraic calculations and digital computer design.[673]

Wolfgang Hagen on Programming Languages

In his essay "Der Stil der Sourcen" ("The Style of the Source Codes"), the German media theorist Wolfgang Hagen studies the history of programming languages up until the end of the twentieth century.[674] He states his agreement with the thesis of Friedrich Kittler that "there is no software." Yet on closer examination, Hagen seems to point towards the opposite.

Hagen seeks to develop a general theory of programming languages. He begins with the thought experiment of an imaginary "library of modern source codes" which would catalogue the "Babylonian confusion" (Kittler). The library would have to include procedural, functional, declarative, object-oriented, parallel, and neuronal languages. It would have to encompass compilers, interpreters, assemblers, operating systems, and code development environments. It would be so vast that Hagen concludes that "our thought museum is a logical impossibility."[675]

Hagen wants very much to agree with Kittler (the founder of German academic "media science"). There are higher-level languages ("symbolic program texts") and "real" (Hagen's term) machine codes. What is crucial is the transition between them. When this conversion happens, it "kills the language."[676] It is a passage from "being" (software) to "nothing" (hardware). Since the program in the higher-level language must get translated into assembler or machine code to run, Hagen asserts that the higher-level software code disappears. That would come as news to Niklaus Wirth, for example, the chief designer of many programming languages and author of classic textbooks on the art of programming with data structures and algorithms. Hagen pokes fun at Wirth. To have the idea that your higher-level program is in a self-preserving relation to the machine is to believe in "a literal and illusory continuity." It is naïve idealism.

Hagen criticizes "American philosophers of the Electric Language" like Michael Heim (author of *Virtual Realism*) and Jay David Bolter (author of *Writing Space: Computers, Hypertext, and the Remediation of Print*) who claim that, according to Hagen, "computer programs and computer systems are a sophisticated collection of programmed texts that interact with each other."[677] I do not believe that the two perspectives of "hermeneutic" or deconstructionist discourse analysis and computer archaeology or materiality must be in competition against each other. They are looking at computing on two different levels. Heim and Bolter wrote about hypertext, which became massively important in the 1990s with the World Wide Web. Hypertext is an existing phenomenon. Documents which are "marked up" with hyperlinks are indeed interacting more with each other than, say, in the Gutenberg Galaxy medium of books. It is legitimate to theorize about and beyond this development. But, for Hagen, it is not the level of "real computer machines." He writes:

> If programming were a strictly deterministic process that followed fixed rules, writes the laconic Swiss [Niklaus] Wirth, programming would have been automated long ago. Or, to use Kittler's provocative words, there would be no software.[678]

What is the difference between "there is no software" and "there would be no software"? In the statement by Wirth to which Hagen refers, Wirth defends the art of programming as an open-ended activity with many possibilities. Hagen does not counter the assertion on its own terms. He dismisses it as self-evidently invalid because indeed the hardware is where the action is.

Hagen sees a historical trajectory of three evolutionary phases: the mathematical models of computability; the engineering technology of memory addressing; and the math and physics of communications engineering. If one goes back to Turing and the first decades of the computer, there certainly was no software. Does it stay that way forever? The role of software, as Hagen himself recounts, steadily increases. In the early

1950s, there are symbols in continuous connection with each other but no mathematical model. In the late 1950s, declarative languages appear. In the early 1970s in a tentative way, and then in a full-on rush in the 1980s, there is "the breakthrough of simulation."[679] Object-oriented languages on the code level correspond to the (personal) computer, on the application level, becoming a media and consumer device. The earlier media of text, image, and sound make their comeback.

Hagen then changes gears and focuses on developing a concept of programming style. The definition of style in the theory and history of rhetoric is that style is paradoxically a property of language and an effect of that very property. Can software rise to this level? He writes: "Can what gets written in software make what is written unrecognizable?" In the codex of ancient Rome that was the precursor of the book, writing became writable. That is the birth of style. In modern Europe, style freed the bourgeois author (the subject of speech and writing) from the restrictive regulations of earlier more regimented societies.

Writing style can tell a story or be interpretative. It can structure an argument. It can be creative. Despite his starting point of Kittler's "there is no software," Hagen takes a wayward turn of intuiting a future of software style: not a language, but "a climate of language."

Ten Paradigms of Informatics and Programming

In his book *Turing's Man*, J. David Bolter characterizes the information processing technique of a Universal Turing Machine as the replacement of "discrete symbols one at a time according to a finite set of rules."[680] This original logic of computing is firmly rooted in the dualism of *is* and *is not* (the long strings of binary digits or 0s and 1s). It is based on the switching of registers and signals in both storage and processing, and the alleged certainty – or identity with itself – of the conventional (pre-quantum physics) scientific object.

In the chronology of subsequent yet concurrent (in the sense that a new paradigm does not completely cause the previous paradigm to disappear) programming paradigms after Alan Turing, ideas which bear a resemblance to hypotheses which Turing "repressed" while devising the hybrid science-and-culture of the digital-binary computer return to the scene. These ideas reappear during the ensuing phases or paradigms, such as those of object-orientation and Artificial Life and Creative Coding. The fact that both program and data can be represented with binary numbers and saved on a physical storage medium is scientific. The relationship between program code and data varies from paradigm to paradigm and is cultural. In 1980s object-orientation, for example, code and data are unified into the single entity or concept of the class or the software object. This was related to the emergence of personal computers and the GUI, to the new emphasis on the computer as a media and consumer device.

I differentiate at least ten paradigms:

(1) Alan Turing's original formulation of the programming of the hardware state machine of the digital computer

(2) The 1950s cybernetics movement around figures like Heinz von Foerster and Gregory Bateson
(3) 1960s procedural programming languages such as Fortran, COBOL, and C
(4) 1970s functional programming languages such as SQL and Lisp
(5) 1980s object-oriented languages such as Smalltalk, C++ and Java, OO analysis and design, and diagrammatic modeling languages such as UML

Each of these five programming language paradigms (which I have linked roughly with the decades of the 1940s, 1950s, 1960s, 1970s, and 1980s, respectively) does not correspond in an exact way to the chronological decade with which I have associated it. The coupling of each paradigm with a given decade is an ideal type binding posited for the sake of establishing a periodic historiographical narrative. Turing wrote his groundbreaking article in 1936. I connect the procedural paradigm with the 1960s; yet Fortran first appeared in 1957, COBOL in 1959, and C in 1972. I link the functional paradigm with the 1970s; yet in fact, Lisp was created originally in 1958 and SQL in 1974. I affiliate the object-oriented paradigm with the 1980s; yet in fact, Smalltalk was introduced in 1972, C++ in 1985, and Java in 1995.

(6) Artificial Intelligence in systems of perceptrons, artificial neural networks (ANNs), machine learning and Deep Learning
(7) Artificial Life related to theoretical biology
(8) Quantum computing in software
(9) Blockchain transaction network concept and other distributed ledger technologies. A-Life, quantum computing, and blockchain architectures are examples of the posthuman worldview
(10) The Creative Coding movement

During the period of the 1950s to 1970s, there took place the rise of the academic field of computer science and the professionalization of computer programming in the corporate business world. The human computers of the 1940s (who were majority female, and who carried out the manual labor of setting into machine language the mathematical calculations specified by male scientists and engineers) were replaced by the automation (the generation of the machine code to run the program) of assembly languages followed historically by language compilers and functional and procedural programming languages such as Lisp and ALGOL. FORTRAN became the language of scientific computation. COBOL was designed as a universal business data processing language that was also closer in syntax to English.

Later came higher-level imperative languages such as Pascal and C, declarative languages such as SQL for database query, and the UNIX operating system, which was portable to almost all hardware platforms and unified academic and business computing. In the 1960s, IBM became the near-monopoly and archetypal company of the computer industry, the Massachusetts Institute of Technology (MIT) – with its close ties to business, government and the military – became the leading university for computer science, and the Association for Computing Machinery (ACM) became the leading organization setting the scientific standards for computing. The concept of the "sciences of

the artificial" (Herbert A. Simon) was developed, and the question if computer science is a science or not was raised.

The decade of the 1980s was characterized by the introduction of the personal computer to the marketplace, and its being advertised and sold to the public as a tool of personal empowerment, interactive visual design, and creative expression. During this era, the computer was also transformed from a calculation machine to a device for media consumerism and individual daily life self-administration. The consumer was encouraged to participate in the spectacle of cultural-economic activity as herself now a media producer. The Graphical User Interface – with its mouse and touchscreen input, desktop metaphor, software applications, hypertext, hypermedia, and the presentation of information as the multimedia juxtaposition of text and image – replaced the text-based command-line interface.

In the realm of computer programming, what corresponds to all of these 1980s innovations on the levels of code and software design are the event-driven model and the paradigm of object-orientation. Software development becomes a methodology for the modeling of real-world processes in preparation for their subsequent simulation, and for the creation of computer games and virtual worlds (virtualization). Object-oriented languages such as Smalltalk, C++ and Java, and diagrammatic modeling languages such as UML, need to be understood as simultaneously technical and cultural paradigms. These object-oriented languages are based on the concept of objects, which are instances of classes, both of which are design artefacts that unify data and code in a single entity. This informatics paradigm and coding culture mark a major step towards enabling the autonomy of software objects and their independence from the controlling power of the programmer-subject.

The First Hyper-Modern Computers

The first machines of computation that can be called digital-binary programmable computers were built around the time of the Second World War and during the period of the late 1940s and early 1950s. One very early digital computer – often considered to be the first – was the *Z3* designed by German engineer and businessman Konrad Zuse, who cooperated to some degree with the Nazi Party and its war effort. Zuse's invention was an electro-mechanical machine, based on an area of engineering where German industry was very strong. The first fully electronic digital computer was the *Colossus*, designed and built by British Post Office research engineer Tommy Flowers, a specialist of vacuum tubes, which took almost a year to assemble and became functional in February 1944. Eleven *Colossus* machines were deployed in the British project of cracking the code of German encryption devices used by Nazi high military command to send battlefield messages to the front lines.

The *ENIAC* computer, built for use by the U.S. Army by Herman H. Goldstine's team at the University of Pennsylvania, was a milestone achievement of design engineering and computer science. It became operational in December 1945. *ENIAC* was much faster than *Colossus* and was fully Turing-complete. It was a universal computing machine and could simulate any so-called Turing Machine, the breakthrough mathematical model which the

24-year-old British mathematician Alan Turing had formulated in his historic academic paper. The *ENIAC*, however, was still not a stored-program computer, meaning that wires and switches had to be manually inserted and set rather than the program and data being stored as software in integrated circuits. The *Manchester Baby*, which was constructed at the Victoria University of Manchester, England, and went into operation in June 1948, was the first stored-program computer which was able to store instructions in electronic memory. The theoretical insights that led to the stored-program concept were elaborated also by Turing in the same watershed 1936 paper and were more concretely fleshed out as a specification by Hungarian American mathematician John von Neumann in his 1945 "First Draft of a Report on the *EDVAC*."[681] The *EDVAC* was another early electronic computer developed under the auspices of U.S. Army ballistics research, and a successor to the *ENIAC*.

Enter Software Studies

The recent emergence of software studies (Matthew Fuller, Lev Manovich) challenges Kittler's thesis that there is no software and points to the primacy of software as a societally critical hybrid of technical and cultural patterns. In 2006 Fuller published a pioneering book on software as media and culture called *Behind the Blip*.[682] In his 2013 book *Software Takes Command*, Manovich expands media theory to include software theory. His book "is concerned with 'media software' – GUI programs such as Word, PowerPoint, Photoshop, Illustrator, After Effects, Final Cut, Firefox, Blogger, Wordpress, Google Earth, Maya, and 3D Max. These programs enable creation, publishing, sharing, and remixing of images, moving-image sequences, 3D designs, texts, maps, interactive elements…"[683]

Thinking with Manovich, one sees that a major challenge to media theory is to consider how Web sites, computer games, and web and mobile applications transform what media are. And how does software affect the design process? Is the nature of design altered by the fact that it is now everywhere carried out with the tools of simulation built on top of object-oriented design patterns? What is the relation between software design patterns and the patterns of other kinds of design – such as architectural, graphical, fashion, communication, industrial, and product design? Manovich asks: "Are there some structural features which motion graphics, graphic designs, Web sites, product designs, buildings, and video games share since they are all designed with software?"[684] What does media become after software?

Lev Manovich's theses are reminiscent of the ideas of media theorist Vilém Flusser who, in his book *Into the Universe of Technical Images*, presented the pragmatic-utopian vision of an SF society of the continuous creation and prolific exchange of high-tech images.[685]

Flusser asserts that technical images are made possible by scientific principles worked up into technologies. Particles of specific technologies (such as pixels – in the contexts of data compression and encryption algorithms) are assembled or computed into visible images. Each image technology (the photograph, the .jpg image, the VRML-programmed virtual world) is a different way of structuring particles. Technical images are reservoirs of information. Programming is a form of freedom. In the future society

of images, everyone will be empowered to envision. Everyone will be a programmer and a synthesizer of images. "There will be an ongoing dialogical programming of all apparatuses by all participants," writes Flusser in *Into the Universe of Technical Images*.[686] New-media artists and creatives should initiate a project of transforming software code into something other than what it currently is. We must go beyond the unconscious "reification" (*Verdinglichung* in German, a term of the Hungarian Marxist literary theorist György Lukács meaning the ideological operation of treating an artefact that is a specific cultural-historical construction as ahistorical or eternal) of assuming that software code as "left-brain" (the rational-calculating side of the human brain) engineers have defined it is the only possibility for software.[687] I propose starting the activity of the active transformation of software by Creative Coders who are artists, designers and thinkers: devising a new curriculum for informatics – a "right-brain" (creative and intuitive side of the human brain) informatics that builds on existing computer science yet moves it closer to art, design, sociology, philosophy, and cultural theory.

According to McLuhan and Powers in *The Global Village: Transformations in World Life and Media in the 21st Century*, reading, writing and hierarchical ordering are associated with the left brain, as are phonetic literacy and the linear concept of time.[688] The left brain is the locus of analysis, classification, and rationality. The right brain is the locus of the spatial, tactile, and musical. Awareness is when the two sides of the brain are in balance.

A key aspect of software code as expanded narration is the concept of similarities – as opposed to the discrete identities and differences of combinatorial software. Similarities is how the universe is constituted. Urgently required for software development after object-orientation is the design of relations of similarity, fractal/holographic-like patterns, and music-like resonance between the whole (the software instance) and the parts (smallest units of information or database elements) as opposed to the logic of discrete identities and differences of Turing machines. The approach that would correspond to a true breakthrough into twenty-first century science would be to identify relationships of similarity, to find samples or patterns that capture something of the vitality and complexity of the whole without breaking it down in a mechanistic way, as in the seventeenth century Cartesian method of dealing with a complex problem by breaking it down into smaller, more manageable parts, along the lines of the mechanistic relation between the whole and its parts in the archetypal car engine.

Designing a logic of similarities involves inclusion of "nonknowledge." We need to rethink science with a dose of nonknowledge, away from the obsession with knowing everything and total information. The importance of nonknowledge for science is manifest in the twentieth century sciences of quantum physics and chaos theory. It is within quantum physics that we find the idea of a vast number of states of information which are potentialities, not yet actualized realities, and which have a relationship of similarity to each other. We want to build a "quantum reservoir" of non-observable information that cannot be read or written in a visible way as in the "get" and "set" operations of programming without destroying the integrity of the data. In the quantum reservoir, we want an immensely vast number of software classes which resemble each other in subtle ways. They are invisible to the observer. The information is read and transformed. In the act of reading, the information transfers from its own quantum state to the domain of "real world" usefulness. An immense number of states should be possible, but switch-

ing actions are manageable. There is flexibility in assigning singularities to classes, and a degree of variability among the individuals of a class.

In the business world, a new software paradigm is emerging – software that handles uncertain social media data and massive volumes of data, software that is an ecosystem. New computing requirements include embedded data analytics, Linked Data, unprecedented massive volumes of data, and continuous self-learning by the software. Storage, memory, networking, and processing move closer to the data. From top-down to bottom-up: long, sequential, symbolic, scripted, ratiocinating logic gives way to short, parallel, semantic-semiotic, coupling of perception and action, immediate intelligence.

Enter Creative Coding

Creative Coding where a line of code is an aesthetic artifact and not only an instruction to the machine. Creative Coding where a new software layer opens as a performance space for music, poetry, storytelling, dance, and philosophy. Creative Coding includes the artist-oriented Integrated Development Environments (IDEs) called openFrameworks, vvvv, and Processing.[689] There is generative art – artworks which are created using an autonomous system such as a computer, a robot, an algorithm, or mathematics. There is the area of programming and music, and the growing area of programming and dance. There is the music programming language called SuperCollider, and the music programming environments called Max/MSP and Pure Data.[690] Open-source Creative Coding toolkits wrap together coding libraries for graphics, typography, computer vision, 3D modeling and audio, and image and video processing. SuperCollider is a programming language for real-time audio synthesis and algorithmic composition. It has strengths in just-in-time programming, object modeling, the sonification of linguistic data and social media data, auditory display, and microsound.

The pedagogy of instructing artists and designers to make software involves teaching them how to write code in a way that is not dry and boring for them (as the engineering approach can often be for creative-oriented students), teaching them how to design software that brings together software patterns and artistic/cultural patterns, teaching them creativity, and teaching them cultural theory so they can grasp conceptually how the paradigm of object-orientation can be pushed through to the next paradigm.

We need to unpack object-orientation philosophically into two separate streams of commodified and creative. The mainstream understanding of OO by engineering schools and the institutions for which they train programmers is philosophically naïve: they assume the existence of a "real world" and so-called "real-world" processes. Software development would be the practice of modeling these real-world processes in software. But this alleged "real world" is the realm of simulacra and simulation.[691]

Creative object-orientation neither assumes the existence of a "real world" nor does it seek to model or simulate that. Creative Coding wants to fashion a "new real," a hybrid of the familiar phenomenological environment and new Virtual Realities, new experiences of existence in a hybrid real/virtual dimension. This is the potential of software at its best.

Alan Turing: The Imitation Game and Befriending the Evil Demon

The mathematician Alan Turing has similarities with René Descartes. Like Descartes, Turing is a rationalist and a humanist. He also endeavors to go beyond Descartes in an interesting post-humanist way. Like Descartes, Turing is engaged in a struggle with an "evil demon" – yet in the realm of Artificial Intelligence and not that of "reality." Can we learn something from the Turing Test for AI to then formulate a "Turing Test for Reality"? In his seminal 1950 essay "Computing Machinery and Intelligence," Alan Turing poses the question "Can machines think?"[692] Turing's paper is widely recognized as one of the first important historical statements about Artificial Intelligence. Turing immediately replaces the question "Can machines think?" with another question which he deems to be more fruitful: "Are there imaginable digital computers which would do well in 'the Imitation Game'?"[693] With the term "digital computers," Turing implies the layers of software for natural-language processing above the hardware level. The famous Turing Test starts out life as what Turing calls the Imitation Game. Before the ability of a machine to exhibit linguistic behavior indistinguishable from that of a human comes a thought experiment about gender: the ability of a man to exhibit linguistic behavior indistinguishable from that of a woman.

There is a man (Person A), a woman (Person B), and an interrogator (Person C) whose gender is irrelevant. Person A and Person B are both not visible to Person C. Based on conversational interaction, the interrogator must decide which of the other two persons is male and which is female. The woman tells the truth, and the male deceptively pretends to be female. The interrogator does not know that Person A is the imposter. The responses are typed, so the gender identities cannot be gathered from voice. The interrogator in the Imitation Game is a lot like Descartes' rational subject. Person A is Turing's evil demon.

Yet Turing feels attracted to this deceiver or imposter. It is a certain "queering" of Person A that fascinates and seduces Turing – a queering of the evil demon. This is disclosed as he takes the next step in converting the Imitation Game from a man impersonating a woman to an AI machine impersonating a human. Turing switches sides to championing the participant in the game who is now the AI software or android. Person A goes from being the threat to rationality to the hopeful possibility of a new paradigm of informatics which Turing defends and for which he argues. The bulk of "Computing Machinery and Intelligence" consists of Turing's systematic refutation of nine rationalist arguments against AI (which he calls "Contrary Views on the Main Question"). He moves intuitively towards a paradigm shift in informatics beyond classical computer science. The behavior of the self-learning program, he asserts, will be significantly different from what is normally expected of programs. Turing wants to understand the science of AI machines which pass the Turing Test. The evil demon starts as mirror-reflection of the rational thinking subject but becomes a different intelligence.

If the tester cannot determine which of the two interlocutors is the machine, then the machine has passed the Turing Test and is deemed to be Artificially Intelligent. The Turing Test is launched into the world. It inspires science fictional posthuman narratives and philosophical reflection and questioning about the future of informatics. The Next Generation of Turing Tests is applied to androids like Rachael in *Blade Runner* and Ava in *Ex Machina*. Not only are Rachael and Ava being tested, but the human who was the

measure of all things is now also placed into question (Deckard in *Blade Runner* and Caleb in *Ex Machina*).

The machine can pass the test by simulating human intelligence. The machine does not have to think like a human or give precisely correct answers. It is enough for it to give answers which resemble the answers that a human would give. Alan Turing writes:

> In about fifty years' time it will be possible to program computers with a storage capacity of about 10^9, to make them play the imitation game so well that an average interrogator will not have more than 70 per cent chance of making the right identification after five minutes of questioning… We may hope that machines will eventually compete with men in all purely intellectual fields.[694]

Alan Turing: The Scientific and Cultural Levels of Computing

The invention of the digital-binary computer is the origination of a numeric code to implement hyperreality in microscopic detail. One way to support the reversal of the dystopia of hyperreality into a more utopian project is to make the methodological separation between the scientific and cultural dimensions of the computer in its history and future. A certain portion of computer science is scientific, and another part is cultural and is understood as changing in paradigm from decade to decade. What is scientific in the "science of the artificial" (Herbert A. Simon) of the computer is the fact that both code and data can be digitalized as numbers.[695] What is cultural is the specific relationship between code and data that prevails in given software coding paradigms which have many different historical configurations. In 1980s "object-orientation," for example, code and data are unified into the single entity or concept of "the class" or "the software object." This was related to the emergence of personal computers and the GUI, to the emphasis on the computer as a media and consumer device. The position that computer science is partly scientific and partly cultural is a more moderate approach than the "social constructivism" of the "social construction of technology" (SCOT) within the field of "Science and Technology Studies" (STS) which says that, in effect, "everything is culture."

We need a novel third knowledge framework that is neither the scientific and technological view from the inside that existing computer science has of itself nor the tendency to cultural relativism and denying of any objective validity to science that often ensues from the view from the outside that is often the research methodology in humanities-side Science and Technology Studies (STS). I highly value many academic works in Science and Technology Studies for their contributions to increasing political awareness of the power, money-making, sexist, and racist relationships which are widely operative in the institutions and cultures of scientific research and technological innovation. However, my primary goal is to develop an intellectual position which simultaneously highlights the economic, social, cultural, and institutional state of things (how power relations are maintained, and capitalist interests served) surrounding science and respects and grants validity to the rationality and special objective status of scientific knowledge that transcends historical conditions.

The thought experiment of the Turing Machine and John von Neumann's "stored-program concept" coincide with the idea of representing both instructions and data as finite sequences of binary numbers. The Universal Turing Machine is based on the switching of registers and signals in both storage and processing, and the alleged certainty – or identity with itself – of the pre-quantum physics scientific object.

What is objective and eternal as science in Alan Turing's 1936 formulation (and related formulations during the birth of computer science which soon followed) is the encoding and physical writing on temporary memory or a storage medium of both programs and code as binary numbers. The relationship between code and data changes in technological paradigm shifts in parallel with shifts in broader socio-cultural paradigms (deciding the era-specific purposes for which computers are utilized). Early computers were deployed for scientific calculations and for manipulating numbers using logical rules. The science part of the invention of computer science: (the hardware and) the algorithms and the data can all be encoded into lengthy binary strings (i.e., stored as computable numbers). The cultural part of the invention of computer science: how one does this (i.e., the relationship between the code and the data) is a cultural decision. When Turing and von Neumann ran algorithms on data for calculation, this was driven by a cultural decision, which was the institutionally needed military applications during the Second World War. They put into practice a certain precise relationship between program and data in their specific deployment of computers.

Computer science is a science in ways which are consistent with how the philosophy and the history of science have studied their objects of inquiry such as in their relation to the classical cases of astronomy, the physical sciences, and the biological sciences. Computer science is not only a set of eternally rationally decided objective truths (time- and discourse-independent properties and laws of a science) but is, in addition, a nonobjective perceptual-interpretive model and a succession of cultural paradigms which evolve and even quantum-leap from historical phase to historical phase, or decade to decade. Computer science is a designed orderly assemblage of ideas, a cognitive schema shared by a community of practitioners which has structured and organized, over a long historical arc of time, our perspective on the scientific area of software code and the computer. What the digital-binary computer has been since its inception as associated with luminaries such as Turing and von Neumann is one essential approach to the scientific field of the computer that establishes some of its principles. Other paradigms are possible which build upon and extend that approach. Alternative-supplementary frameworks of informatics are either historically identifiable in genealogical stages or extant in emerging and formative states.

The invention of the discrete logic of the on-or-off state of the bit smallest unit, or the lengthy strings of 0s and 1s, or the symbolic code or algorithms, of digitalization by Alan Turing was both a universal invention of a scientific technology *and* was embedded in Turing's allegiance to ideas of the twentieth century philosophical movement of British analytical logical positivism. Turing made certain scientific and design decisions, and some of these decisions excluded certain other architectural directions which he might have taken. It is possible to separate the scientific and the philosophical-cultural-discursive aspects. Since informatics has by now made such a deep imprint on our lives that one can point to a thoroughgoing "information-ization" and "number-ization" of hyper-

modern society, it can be said that the digital-binary computer is coupled – in an elective affinity and a prolonged historical trajectory – with certain systemic social, economic, and institutional values and goals.[696]

A long and fascinating intellectual and techno-scientific history (which, in a sense, spans all human history) led up to this quantum leap forward or scientific revolution which was the mid-twentieth century crossover from abstract ideas to the actual physical construction of the digital-binary computer. Many events in the history of mathematics, the philosophy of logic, and the design and building of successive calculation and computation machines are often chronicled as chapters in the prehistory of the computer.[697]

Jay David Bolter: Computer Science and Literary Theory

As Turing argued in "On Computable Numbers…," any specialized automaton (a precursor of the computer program) can be represented by and implemented with a finite set of binary instructions. Therefore, a universal automaton (computer hardware) can be imagined (as a thought experiment by Turing) and then built (the computer architecture of von Neumann) which would precisely mimic the desired behavior of the specialized automaton or software by cycling through those same instructions. As Herman H. Goldstine – the mathematician and computer scientist who was one of the developers of the late 1940s ENIAC (the first electronic digital computer) – explains: if the universal automaton can hypothetically run without any limitation of time, it will always execute at some juncture in its execution the desired sequence that is contained within the infinite sequence (somewhat like the proverbial monkeys who will eventually reproduce the complete works of William Shakespeare if given enough time banging away at typewriters). Turing made the mathematical proof that the specialized automaton can always be described by a sequence of discrete directives which are the code input to what would later become the physical computer. "When the instructions are fed to Turing's universal automaton," notes Goldstine, "it in turn imitates the special automaton."[698]

Jay David Bolter, professor of New Media at the Georgia Institute of Technology, undertook an interdisciplinary study of informatic technology in his books *Turing's Man: Western Culture in the Computer Age* (1984) and *Writing Space: The Computer, Hypertext, and the Remediation of Print* (2001), bringing together computer science and literary theory.[699] Yet Bolter (like Kittler) appears to have made little progress in his work in envisaging software as embodying literary, cultural, or signifying patterns. He instead stays within the scheme of assuming absolutely that programming is about numerical-combinatorial logic and the manipulation of discrete symbols. Like Kittler, Bolter limits computer thought to a sort of philosophical nominalism where the semantic and semiotic aspects of the signfying words or identifiers (whether keywords or variables named by the writer of the code) in programming languages count for nothing: "Computer thought is a sequence of operations, of fetch-and-execute cycles of the central processing unit."[700]

Bolter is concerned only with the original logic of computing of the Turing Machine as an information processing device where the symbol written at the storage location currently pointed at gets replaced by another symbol selected from a finite set of symbols according to a set of rules. This is the embodied metaphor of a physical model of re-

ality that comes with its dubiously perfect "description-language" (reminiscent of what Paul Feyerabend in *Against Method: Outline of an Anarchistic Theory of Knowledge* critiqued as "observation-language" in science[701]) – the dualism of *is* and *is not*, the philosophical-scientific assumptions behind that, and the long strings of binary digits or 0s and 1s.

The logic of identities and differences, the mathematical-philosophical axiomatic postulation that a thing is identical to itself, or that there is a one-to-one linguistic relationship between signifier and signified (the word-token and the meaning of that word) – this obsession of Western culture with reality is at the root of hyperreality. The idea of language in Aristotle and in the Noah's Arc story in the Bible is that language names the world. Both Aristotle and Noah develop classification systems of naming the animals.[702] The prominent linguist Noam Chomsky thinks that language is a universal structure of the human brain which is always the same independent of the specific languages of specific cultures.[703] This implies anthropocentrically that language essentially is the world, that language matches the world and harmonizes with the world. My view is that language is a continuous back-and-forth tension between understandings and misunderstandings, attempts at contact with the other and the confusion of the Tower of Babel. The view of language as a classification system is useful for organizing and categorizing. It overstates its claim that language is only an objective codification system that describes how everything in the world is.

Bolter states: "Every computer program is the electronic realization, the tangible proof, of a theorem in logic… Every programmer… is a logician with a theorem to prove."[704] What is certain concerning the place of electronic digital thinking in the long arc of the history of ideas, Bolter asserts, it is that the land of CPU clock cycles is a kingdom from which God, religion, meaning, and ethics are excluded. Philosophy, psychology, ecology, and literature are exiled. There is no contemplation of existence or introspection. There is no union between minds or sensuous touching between the computer and its exterior environment. "The unification of the mind with the idea of the beautiful, the true, and the good envisioned by Plato" – the ideal world of the Platonic Forms and Ideas, the beginning of Western philosophy – "the series of perfect patterns from which the imperfect objects of the material world" are derived, Bolter tells us – "has no counterpart in computerized thought."[705]

As a humanities professor who understands computer science, Bolter was eager to educate his colleagues about the logic of computing. Yet he inadvertently set up a wall between a statically conceptualized logic of computing and the thinking of the humanities and cultural studies, thus excluding contributions by the latter to the former.

Lev Manovich, *The Language of New Media*

In 2001, The MIT Press published the book *The Language of New Media* by Lev Manovich.[706] This is a milestone work in the academic theorization of new media. Manovich investigates cultural software and interfaces, visual culture and moving images, and the historical transition from film to digital video and computer games. He develops theses concerning conventions and artefacts of software applications and user experiences in these areas: interactivity, telepresence, immersion, distance and aura, digital compositing and

montage, computer animation, databases, algorithms, storing and manipulation of information, and the navigating of digital and virtual spaces. The cultural and aesthetic forms of new media are both a continuity with and a break from older media such as the cinema.

Manovich enumerates five principles which characterize new media:

(1) Numerical representation – Artefacts exist as data or can be stored as numbers
(2) Modularity – Different elements exist independently
(3) Automation – Artefacts can be created and modified by automatic processes
(4) Variability – Artefacts exist in multiple versions
(5) Transcoding – The digital-binary logic and its instances influence us culturally – from technical codes to cultural codes

New media objects are based on code, on the limitless re-programmability of the binary structure and the electronic impulses. Software Studies (Lev Manovich, Matthew Fuller, Benjamin Bratton, and other authors in the same-named MIT Press book series) in effect contests Kittler's thesis that *there is no software*.[707] Software Studies points to the primacy of software as a hybrid of technical and cultural patterns that is potentially both critical of society (*Gesellschaftskritisch*) and "designing of the future" in a pragmatic-utopian sense.

Software Studies: Coded Objects and Assemblages

In the book *Code/Space: Software and Everyday Life* (published in the MIT Press Software Studies book series), Rob Kitchin and Martin Dodge examine the explosive growth of information about ourselves, the intrusion of this information into our daily lives, and the ubiquitous availability of this data to institutions and strangers through many networked devices.[708] Their approach is to scrutinize software from the perspective of space, to research how the "production of space" (a term of the French Marxist sociologist Henri Lefebvre) in the guise of the new virtual space is implemented in a detailed way by software.[709]

Kitchin and Dodge see software as increasingly integrated into everyday life in the four domains of coded objects, coded infrastructures, coded processes, and coded assemblages. An assemblage, for Deleuze and Guattari in *A Thousand Plateaus: Capitalism and Schizophrenia*, is a unity of social-technological entities amalgamated into a configuration that is fluid, multi-functional, and complex.[710] The assemblage can combine organic and machinic components into its dynamically changing aggregation of parts and its relations with other assemblages. Assemblage theory is a systems theory for the social world.

Coded objects are, for Kitchin and Dodge, physical objects which depend on software for their functionality. Their product design implementation is made possible via software code. In the environment of the Internet of Things, computational power is embedded into many objects. There are other machine-readable objects that lack their own software but interact with external code. Coded objects are connected to distributed information and surveillance networks. Some objects develop something like an awareness

of themselves and their milieu (perhaps in Katherine Hayles' sense of the "cognitive nonconscious").[711] Their interactions with surroundings are recorded and saved on physical storage media or the cloud.

In Chapter Three "Remaking Everyday Objects," the authors study how everyday objects such as domestic appliances, handheld tools, sporting equipment, medical devices, recreational gadgets, and children's toys are made software-interface-addressable and thus available to external processes of discipline, control, and identification. The Internet of Things can become a platform against surveillance. My things or my objects belong to me, not to the government or large corporations or the semiotic consumer society.

Software Studies: The Expressivity of Code

In the book *Speaking Code: Coding as Aesthetic and Political Expression* (MIT Press Software Studies series), Geoff Cox and Alex McLean elaborate a hybrid discourse of software code writing and humanities critical theory.[712] Blending text and code, and musing on code as script and performance, they locate the signifying import and linguistic reverberations of code in its practical operations in the online networks. The study of code by Cox and McLean is an existentialist view of software programs as having open-ended possibilities, rather than the usual emphasis on their social-organizational impact of instituting fixed structures and processes. Cox and McLean examine the live-coding scene (visually displaying source code during an artistic performance) and peer production (self-organizing community efforts such as open-source software projects). They see code as an expressive and creative act, related to the two activities which have traditionally been called "art and politics."

The autonomist thinker Franco Berardi writes in his foreword to *Speaking Code*:

> If we can say that code is speaking us (pervading and formatting our action), the other way around is also true. We are speaking code in many ways… We are not always working through the effects of written code. We are escaping (or trying to escape) the automatisms implied in the written code… Hacking, free software, WikiLeaks are the names of lines of escape from the determinism of code… Linguistic excess, namely poetry, art, and desire, are conditions for the overcoming and displacement of the limits that linguistic practice presupposes.[713]

Many such projects – and more generalized in their transformation of what code is – are possible. Poetic, musical, and symbolically signifying language can reemerge within code to counteract the original historical assumption that code is a series of instructions to a machine, an exercise in formal logic, and the reduction of language to information. Text and code come together as an embodied cyborg cooperation (Katherine Hayles, Donna Haraway) or as a relation of uncertainty and indeterminacy where each partner in the human-machine exchange is reciprocally transformed. This can happen in the double frame of code as both readable as directions for the processor and as elegant expression for the human code writer.

Cox and McLean refer to the concept of "double description" as mutual causation or circularity between mind and biological evolution that was elucidated by the thinker of second-order cybernetics Gregory Bateson in his 1979 book *Mind and Nature: A Necessary Unity*.[714] Starting from this notion, the authors speak about "double coding": a composite of formal logic and linguistic creativity in Codeworks (Alan Sondheim's mixing of creative writing and code) or "pseudo-code" (informal descriptions of the steps of a program or algorithm, often a phase of software development preceding the writing of code), a hybrid articulation that is both rigorously systematic and carries the force of writing.

Vilém Flusser and Software Code

In *Into the Universe of Technical Images*, Vilém Flusser presents the pragmatic-utopian vision of an advanced utopian science fictional society of continuous creativity and permanent prolific exchange of high-tech images.[715] Flusser writes in the mode of SF theory.

Calculation and computation get added to the scientific method, and to reading and writing, as treasures of the Western cultural tradition of liberal humanist rationality and Enlightenment progress. Flusser's vision is a community of creating and sharing images.

Flusser stresses the historical continuity between the culture of written texts of the pre-digital world (which were both scientific and literary texts) and the universe of technical images. The technical image is much more an outcome of the achievements of scientific and literary texts than is usually believed. This is the opposite of what Marshall McLuhan maintained in his historical genealogy of a radical break between successive print and media cultures. Technical images are anything but natural or a return to pictorial images, as McLuhan had claimed (while calling them electronic images and saying that electronic culture retribalizes humanity). Linear texts, for both McLuhan and Flusser, have been the dominant carriers of information in human societies for four thousand years. Prior to that – for "the forty-thousand-year-period of pre-history" – pictures reigned supreme.[716] With the World Wide Web Internet that ascended in the 1990s, there is a shift from linear text to hypertext and hypermedia. Flusser diverges from McLuhan's concept of electronic images, pointing out that these images in fact, "rely on texts from which they have come and, in fact, are not surfaces but mosaics assembled from particles."[717] Technical images are a continuation of the Western culture of scientific and literary texts, a continuation by other means.

Flusser calls the traditional pictorial images of pre-history "first-degree abstractions." Those images were mimetic representations or phenomenological impressions of the physical world. The "second-degree abstractions" are texts which are, in turn, abstracted from traditional images. Technical images are "third-degree abstractions." They are abstracted from the abstraction of the abstraction (the pre-historic images) of the concrete world. Technical images can also be called post-historical.[718] Technical images are reservoirs of information.

Software programming or the writing of code is, for Flusser, a form of freedom and individual expression. In the future utopian society of images, everyone will be empowered to envision. Everyone will be a programmer and a synthesizer of images. He writes:

> The photographs, films, and television and video images that surround us at present are only a premonition of what envisioning power will be able to do in the future… All vision, imagination, and fictions of the past must pale in comparison to our images of the future.[719]

From the perspective of the present, we see more clearly the unity of the scientific and literary cultures as they were in the past and might become again in the future. Scientific and literary cultures will no longer be in opposition to each other. They are both cultures of the text.

From the Dialogical Society to Creative Coding

Flusser writes about Telematic Man and advocates for emancipatory possibilities inherent in the universe of analogue and beyond-analogue technical images as well as dialogic or advanced digital images. Flusser was a utopian thinker, similar in his moral and theological perspective to the philosopher of *I and Thou* (*Ich und Du*) Martin Buber ("I mean roughly that which Buber called *dialogic life*," writes Flusser), touched by the spirituality of Jewish Kabbalah in ways close to the historian Gershom Scholem and the Frankfurt School philosopher of critical theory Walter Benjamin.[720] Flusser brings his existentialist philosophy to bear on media and technologies. In his book *Into the Universe of Technical Images*, he contemplates the prospect of a future society that *plays* with digital-virtual images: "It will be a fabulous society, where life is radically different from our own."[721]

This utopia will not be automatically realized by new media and new technologies. The better society can only be realized when digital technologies are designed consciously with utopian values and goals. "Taking contemporary technical images as a starting point," writes Flusser, "we find two divergent trends. One moves toward a centrally programmed, totalitarian society of image receivers and image administrators, the other towards a dialogic, telematic society of image producers and image collectors."[722] Totalitarianism or liberal autonomy and democracy: the choice is up to us. The future culture of images implements "a technology of dialogue, and if the images circulated dialogically, totalitarianism would give way to a democratic structure."[723] Either we continue living in a bureaucratic social order with images controlled by a few powerful monopolies or we architect a telematic society of decentralization, empathic dialogue, mutual support, and collective authoring of the narratives of visual culture.

We also need to deepen understanding of what Flusser means by image, and how that differs from the usual meaning of the term.

There can be a coming together of Virtual Reality or computer games and stories of high literary quality – a culture of images that continues the culture of literature of the past. There will be a high level of participation in such a culture. "There will be an ongoing dialogical programming of all apparatuses by all participants."[724] It will be a playful existence, a society of *artists* in dialogue via images. Flusser refers to the notion of *Homo Ludens* of the Dutch cultural historian Johan Huizinga on the play element in culture.[725] The dialogical society, for Flusser, would envision "situations that have never been seen and could not be predicted," lived by players who would "constantly generate new rela-

tionships by playing off moves against countermoves" and write the code of previously inconceivable possibilities.[726] We need a new conscious theory and practice of images – images related to the reinvigoration of the hybrid scientific-literary culture that is the legacy of the West.

In *Does Writing Have a Future?*, Flusser envisions a path towards meaningful expressivity emerging from the metamorphosis of programming codes. Flusser anticipated the movement of generative art or Creative Coding.[727] Creative Coding is rooted in the desire and ambition of artists and creative individuals to practice software programming in a range of subcultural activities: live visuals, interactive exhibitions, choreographed dance, real-time performances, product design prototypes, and 3D printing and hybrid design-and-technical code experimentation in Maker Labs, demoscenes, and hackerspaces. Creative Coding includes projects of visual- and natural-language-centered toolkits, software poetry, and coder ethos sensitivity to the art of programming.

Flusser investigates the prospects for "writing after writing." Hope for a better society, he states, cannot be placed in those who know how to write the old way yet refuse to learn the new technological codes. Nor can hope rest with those who learn the new codes in a robotic or merely professional way (without awareness), yet remain ignorant of the value of writing, both as it was in the past and as it could be in the future. Educational institutions should teach the new codes while encouraging students to learn the history of writing and to engage in the renewal of that history. Texts will make their comeback against their suppression and replacement by computer programs, operating inside the latter to transform them, to bring text and code back within the overall flow of writing's place in history.

In *Towards a Philosophy of Photography* (1984), Vilém Flusser asserts that media technologies do not transform the world, they transform the meaning of the world.[728] They transform its symbolic dimension. We are no longer in the era of industrialization and production (of tools and machines). The photographer – who, for Flusser, is a metaphorical stand-in for all technology programmers – is not a proletarian in the classical Marxian sense. The imaging technology apparatuses do not do any work. The term photography, for Flusser, is a stand-in for all contemporary media. The structure of the gesture of photography is quantum. It is a gesture of doubt composed of point-like hesitations and point-like decisions. Photography is a post-industrial and post-ideological gesture. Photography takes information to be a "new real" in itself. It does not seek to decode the alleged meaning of that information. Creative Coding is not semiotic coding and decoding – the concept of ideology with some semiotics added – as in the Marxist television studies of Stuart Hall.[729]

The telematic society of the future – if it continues its present dystopian trajectory – will be divided into two groups: those who write computer programs and those who cannot write software code. The technocratic programmers will be pawns of the system just as much as the non-programmers will be. Their personalities will be programmed on a micro-level through each keystroke that they type: a society of programmers who are programmed. Programming can instead become the new name for what used to be called writing. Computer programming languages – as they have been until now – are structurally simple (they reduce or translate, as Friedrich Kittler says, to the digital-binary code), but not at all simple to learn and use. They are structurally simple yet functionally

complex. Programming, as it is presently constituted, leads to the automatic steering of human beings and society into a cybernetic system. Programming as we have known it is the automation of the world.

Flusser was a utopian media theorist who wrote about a future playful society of the democratic exchange of dialogical images. He investigated the place of the writing of software code in the larger context of the history and future of human writing in general. He connected photography and programming in interesting ways.

From Computer Science Code to Creative Coding Code

What is the difference between code as understood by existing computer science and code as understood by posthuman Creative Coding? Software programming languages came into existence much later than the original invention of the computer, but they are marked by the mathematical origins of computer science and the idea of a pure mathematical formal language. Each line of code is a precise unambivalent instruction. It is the opposite of human language. Human language is imbued with resonance, ambivalence, poetic qualities, subjective expression, cultural cross-references, and intertextualities. Code in its existing concept is also not visually creative in any sense of making space for singular pictographs since it consists of sequences of pre-defined symbols. The symbols laid out by the algorithm at hand are selected from a larger set of symbols of a given delineated alphabet. In existing computer science, there is a dualistic separation between the code (or the phase of code-writing) and the (time of the) executable. The activity of writing the code happens outside of the instantiated process or world which the code has set in motion. The code has a human-writable and -readable version called the source code. An interpreter or compiler converts the source code to the machine instructions required by the computer. Code is a system of rules to convert information to an alternative form to be sent over a communications channel or saved on a physical storage medium.

The following three new directions for the theory and practice of code in posthuman Creative Coding stand in the foreground:

Code and Human Language

One of the most popular application domains for projects made with the Processing Creative Coding Integrated Development Environment (IDE) is poetry generators. Software poetry embeds eloquence into purposefulness. When the center of attention of the writing of code becomes expressivity as well as functionality, the desire for programming languages which are closer to human languages grows. This tendency is already visible in the expansion of Natural Language Processing (NLP); declarative programming languages for relational databases like SQL; macro languages for lawyers to write Smart Contracts on the blockchain; the role of natural language in comments and documentation to make source code more readable for other programmers; the choosing of humanly familiar names for variables and methods; the syntax specifications of markup

languages like XML and HTML; and in the natural language input styles of AI text and image generators.

Higher level languages already evolved away from the primitive 0s and 1s towards human forms of communication and communities of understanding. Higher level languages are already closer to human language than to machine language. This trend can be extrapolated into projected further steps towards code as human language as the future unfolds. What will the practice of software development be like when its concern is both software codes and cultural codes? The "new real" emerges when designers of hybrid real-virtual environments have a toolkit available which offers building-block component options from both the real physical world and from the province of virtual three-dimensional synthetic imaging technology.

Code and the Visual

Another popular application domain for projects using Processing is music visualizers. The numerical values of the music as a data set become the input to code which converts those values to some real-time dynamically changing, or even user-interactive, graphical representation as output. Processing is especially adept at translating from one expressive media to another, as exemplified in projects that transform electromagnetic waves in the atmosphere into lively screen or VR animations; transmute dance movements into database-storable geometric forms as building blocks for future choreographies; transpose the motion-activity of children's play into music and light displays; alter weather data into three-dimensional "fuzzy" phenomena-simulating particle systems; or transfigure bodily tactile gestures into large-screen flowing clay sculptures. Processing enables generally the creation of interactive visual artworks and art installations. The digital version of Generative Design that is related to Creative Coding instantiates an algorithmic system via code which, in turn, serves as the "intelligence" that autonomously-automatically generates design or artistic output.

If Processing has migrated the attention of coding towards visual output, then the next step is for the code itself to become more visual – more artistic, intuitive, inspirational, emotional, and pictorial. The symbols available in the language's symbol set can be more malleable and expressive of the singularity of the specific expression intended in the moment. A dynamic pictogram is a flexible graphic symbol signifying its meaning through resemblance to signified likenesses evoked in the imagination. Small vector-spawned fractal icons can be phonetic letters or elements of the language. Pikto and Lightbot are examples of already existing pictographic programming languages.[730] The given "pikto" directly embodies an action or object in the game. These languages based on schematic images avoid the pitfall of the syntactic errors that vex textual languages, making them suitable for learning by children.