Author's Declaration

Mathematician: *"I swear to tell the truth, the whole truth, and nothing but the truth, so help me God."*

Programmer: *Do not take what I write and say as true. All texts, whether in this book or in my programs, are incomplete and may have contradictions, inaccuracies, or errors. Moreover, how they are perceived depends on the individual context and the reader's state of mind, on future discoveries, and so on. Consider the works of Gödel, Einstein, Heisenberg... Yet even they should not be taken solely at their word. That said, my own thoughts and intentions are pure and sincere.*



Two-faced Janus: scientist or programmer?

© 2024 Sergey Tolkachev

Introduction

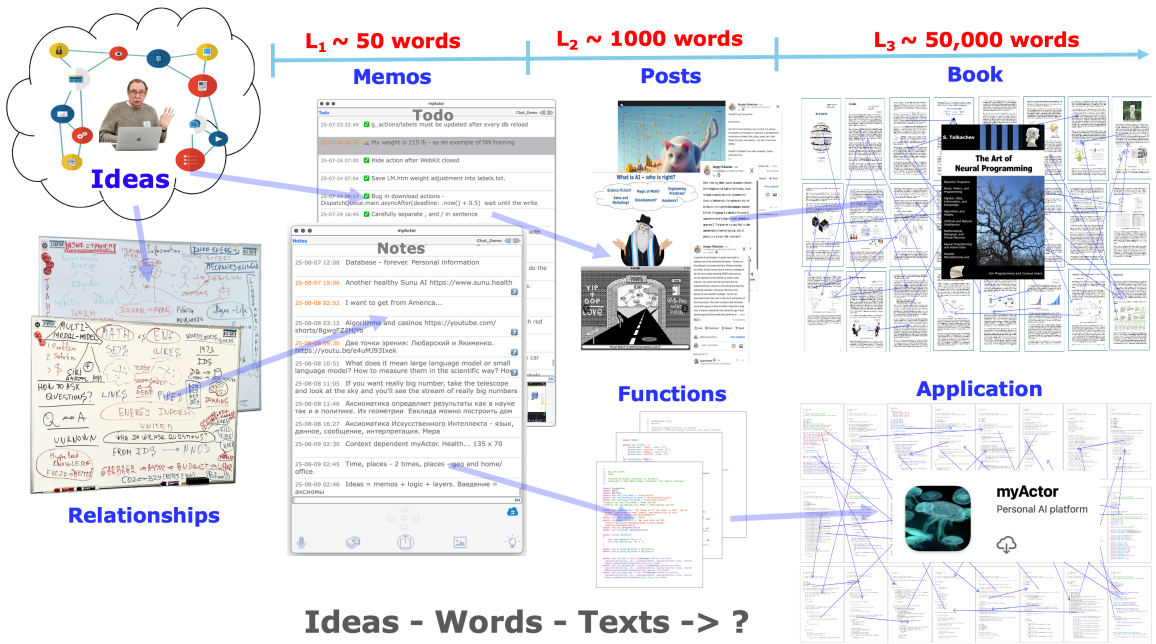
"Scientists write with painful difficulty. However, there is an even more arduous task: reading their scholarly works. And, to tell the truth, no one really takes on this matter..."

A. I. Herzen. Dilettantism in Science

"But still, there are readers of such delicacy that they will certainly want to read to the very end so as to make no mistake in their impartial judgment..."

Fyodor Dostoevsky. The Brothers Karamazov

This book consolidates two of my earlier works: "Neural Programming" and "Active Data", complemented with hundreds of posts, presentations, a large collection of memos, many conversations, and the reflections that have left countless associative traces in my memory. I am driven to share this material by my long tenure in academia, where teaching and research intertwine through lectures and discussions, and my long-standing, passionate engagement with programming. All three professions deal with words: the scientist observes and records; the educator interprets what scientists write and clarifies it for students; and the programmer translates anyone's words into actions, instructing the computer on what it must do in any given situation.



The three words in the title of this book: *art*, *programming*, and *neurons* – form a tripartite foundation around which the entire subsequent narrative is built. Curious readers might also

be interested in experimenting with a *neural exploratorium*, an evocative term that is the equivalent of an integrated development environment (IDE) in traditional programming.

Programming is an extraordinary and unusual profession. On one hand, it is filled with formal rules, routine, and bureaucracy; on the other, it is deeply intertwined with the kind of creativity characteristic of the *arts*. But what could a programmer possibly have in common with a poet or a composer? Or with a mathematician or a mechanic? At first glance, these fields of activity seem worlds apart. And this is the perfect moment to turn to that brainchild of programmers known as "Artificial Intelligence". The modern world of knowledge (including literature in all its manifestations) can no longer be imagined without the accompanying services that offer suggestions, perform checks, and answers questions. So, let's ask Google or ChatGPT to compare. The response, drawn from a vast library of tokenized texts, might be perfectly right, yet it remains cold and mechanical. It has been generated based on what was previously written by countless unnamed authors: indexed, shuffled, stitched together, and polished into prose. Yet, this response has neither soul nor beauty.

Having finished the last sentence, the author fell into thought, turning his gaze inward. What, then, are the soul and beauty? And what does programming mean to me personally? No, let us not ask AI; it is better to figure it out for ourselves. Each of us has something deeply personal, something rooted in our own feelings. I, for instance, love programming – the mother of all professions. I love it because, while working on a program, one can simultaneously be a mathematician, a poet, a musician, and even an end user, a pursuit to which one can dedicate an entire lifetime. But no, not for that, after all, one cannot love something merely through the intelligence, based on rational reasons. Who knows, when the first cell appeared on Earth, initiating Life according to the DNA program embedded within it, who was the Programmer that created this model? And how was it possible to construct such a stable, universal system for the encoding, storage, and interpretation of data and programs? Modern developers can only envy such a reliable, efficient, and elegant system of information processing. Could it all be about love? But that, of course, brings us to Aristotle and his treatises on politics and metaphysics.

Why does a programmer need the theory of relativity or Kolmogorov complexity? How can symphonic music help in creating websites, or knowledge of physics in understanding C# or JavaScript? At first glance, there is no direct connection between these concepts. However, a relatively recent idea called "*Intelligent Learning*" emerged, and one contributor, V. Vapnik, proposed that learning improves when factual knowledge is paired with ideas that seem unrelated but are linked by association. Moreover, modifying the actual data within a specific domain does not break the model, and adapting to new information happens faster and more easily!

From this perspective, we shall try to analyze current trends and apply our findings to the creation of more robust programs, programs capable of enduring and functioning across a diverse range of "platforms-stages". To this end, we will examine programming from various vantage points, including an internal perspective. Much like humans, computers will, eventually, seek to understand themselves. At the heart of this discussion lies a fundamental question: how do processes, whether within a human being or a computer,

engage in a dialogue with themselves, translate thoughts into language, and pose questions? Furthermore, *why* do they ask these questions, and to whom are they addressed? Biologists and neurophysiologists have now amassed sufficient knowledge on the anatomy and dynamics of cells, including those of the nervous system and the brain, to explore how the processes occurring within a computer fundamentally differ from those in living organisms. For programmers, the relevance of this inquiry is clear: comparing the structures and programming paradigms found in biological systems with those in computational systems can pave the way for the development of novel information models. Conversely, these models and programming methodologies may prove valuable to biologists, experts in other fields, and even practitioners in the arts.

In 1968, the first volume of Donald Knuth's monograph, "The Art of Computer Programming", was published. At that time, the number of programmers worldwide was in the tens of thousands, and one could only truly be considered a programmer if one were capable of grasping the essence of the problems presented in that book, even if one could not solve them completely. In those days, the foundation of programming consisted of mathematical methods and algorithms designed primarily for the automation of calculations, which is why the machines themselves were called "computers". Half a century later, the number of people engaged in programming has grown to over forty million, making it one of the most widespread and accessible professions. Much like mechanics in the field of mechanical engineering, programming has become a universal tool for addressing a vast array of information-related tasks; indeed, the fundamentals of programming and computer science are now even included in primary school curricula. The nature of applications has evolved, giving rise to new technologies and tools. And while most contemporary programmers may not have read Knuth's monograph, that has not prevented them from creating useful, practical solutions.

Donald Knuth has described programming as an art. While the term is mentioned only in passing within the book itself, this thesis is rooted in profound philosophical concepts that he later explored in his Turing Award lecture, a talk subsequently published in 1972 under the title "Computer Programming as an Art." This lecture ought to be on the required reading list for anyone looking to pursue a career in programming; to understand why, one need only consider a few quotes:

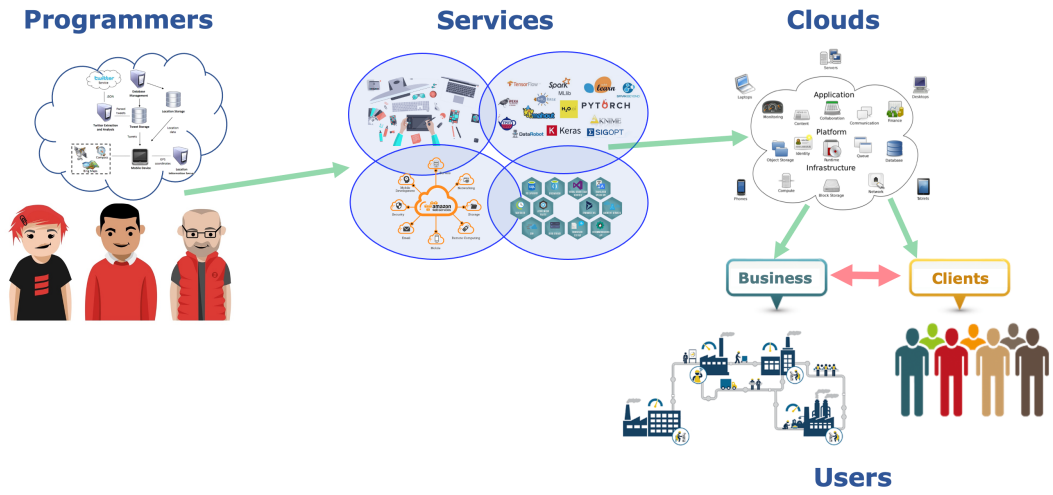
" As civilization and learning developed, the words took on more and more independent meanings, ***science*** *being used to stand for knowledge, and* ***art*** *for the application of knowledge. Thus, the science of astronomy was the basis for the art of navigation. "*

" The chief goal of my work as educator and author is to help people learn how to write ***beautiful*** *programs..."*

" My feeling is that when we prepare a program, it can be like composing ***poetry*** *or* ***music****..."*

" Well, it's true, not all programming tasks are going to be fun... It is still a pleasure to do routine jobs if we have ***beautiful*** *things to work with.*

At the beginning of his lecture, D. Knuth notes that the word *art* has multiple meanings. Yet, the term *programming* also shifts in meaning depending on the context and the era – much like the word *computer*. Originally, a *computer* referred to a person who performed mathematical calculations according to a set of rules; today, calling a human being a *computer* would be, at best, an ironic remark. Programming in the 1970s and programming in the first quarter of the 21st century represent fundamentally different technologies, requiring entirely different skill sets from programmers. Algorithmic languages (such as ALGOL, Fortran, etc.) have been superseded by new languages and tools, including those for machine learning and neural modeling. The environment in which programs now run extends far beyond traditional computers; it encompasses remote cloud services and personal clouds, where multimodal devices, intelligent edge computing, various gadgets, and many other components interact.

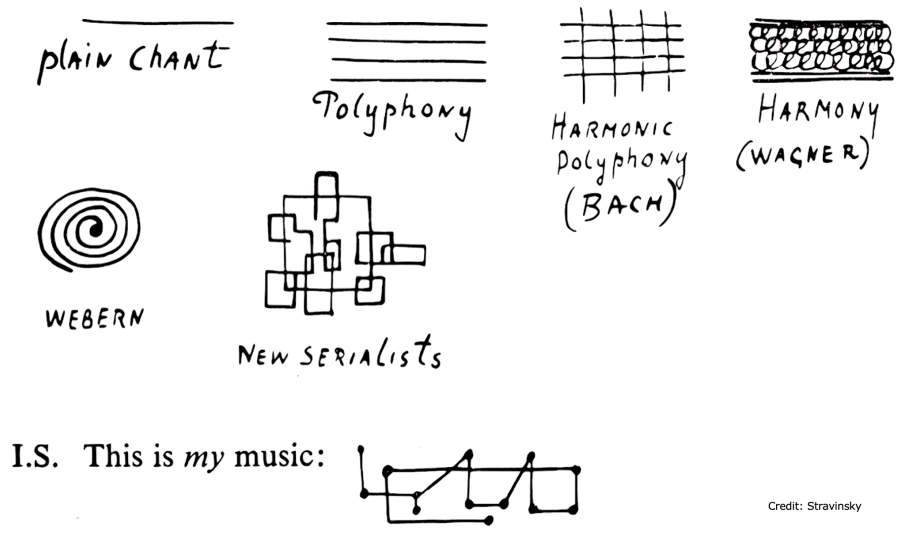


Programming 2020

Comparing programming to music and poetry serves as an example of how analogies can help us understand complex phenomena. Music and poetry share common foundations: a composer or poet records a work (a musical score or a poem) as a linear sequence of symbols (notes or letters), incorporating explicit or implicit temporal markers. Musicians and performers then execute these works, guided by *rhythmic* synchronization. Similarly, program code compiled into a linear sequence of instructions, is executed by a computer at a rhythm dictated by a clock generator. In music, the simplest composition consists of a single melody; in programming, the simplest program is a single algorithm. Yet, when multiple melodies or processes are combined into groups, linear execution evolves into something multidimensional. Multitasking – one of the most complex areas of programming, is a prime example of this multidimensionality. In mathematics, such issues are well-studied: as dimensionality increases, for instance, in objects constructed through spatial concatenation, certain properties may vanish, while others take on entirely new meanings. For example, the familiar concepts of *greater than* and *less than* do not apply to complex numbers (defined by two coordinates), though they are used when comparing the areas or volumes of geometric figures. Grasping the logic behind these properties requires a deep understanding of fundamental mathematics, as well as specialized training and expertise.

One could provide a long, detailed explanation to students regarding the technical intricacies of managing concurrent processes; yet, if they were to first attend a symphony concert, where, under the guidance of a conductor, a metronome and a baton serve to synchronize the melodies played by various instruments, they would likely find it much easier to grasp the essence and beauty of parallelism. A symphonic performance offers an excellent illustration of how the formal and universal, in terms of magnitude, speed, or other measurable attributes, can be distinguished from the informal and individual, in terms of quality or aesthetic merit. If a programmer crafts a piece of software that they find personally satisfying, one in which they perceive a harmony among interacting processes, then that software embodies the aesthetic category of *beautiful* to which Donald Knuth once referred.

One of the most striking illustrations of the analogy between music and a program is a sketch by I. Stravinsky, which demonstrates the evolution of forms of melodic harmonization.



Programming à la Stravinsky

The illustration shows how music has evolved: from monotonous drum rhythms to rhythms featuring amplitude and frequency modulation, manifested as melodies played on various instruments and, ultimately, to multidimensional harmonic polyphony. If we view this sketch through the lens of programming, replacing musical terms with technical ones, we can discern the evolution of programming itself: from classical sequential algorithms to a broad spectrum of interacting parallel processes within a networked environment. It traces a path from the Turing machine (represented by a drum or a linear interpretation) to artificial intelligence (akin to musical serialism or a philosophy of life rooted in tokens).

And herein lies the key to answering a most intriguing question: what comes next? From a formal perspective, the task is to model the continuation of this pattern, to derive an element that defines the characteristics of the next generation of musical or programmatic forms. For classical musicians and poets, this has proven to be a formidable challenge. Perhaps the

reason is that the solution extends well beyond music or poetry in their pure, isolated forms; it requires higher dimensions and the integration of various modes of artistic expression into a unified whole. In mathematics, two fundamental methods approximation and extrapolation are used to analyze and predict the behavior of datasets. Yet, while these techniques work well for numerical sequences, they are fundamentally inapplicable to sequences of graphic images, let alone the meanings that lie beneath them. Extrapolating Stravinsky's musical forms using purely mathematical methods is futile; however, thoughtful, conceptual analysis of their constituent elements and the ways in which they are combined could prove valuable for developing new architectural models.

It is common knowledge, even without the need for scientific research, that the fusion of poetry and music gave birth to theater, ballet, and opera. In the language of programmers, this combination could be described as *multimodal polyphony* – the integration and synthesis of melodies, words, movements, and stage design within a single space. Yet, beyond the author (or programmer), theater requires actors and a director. A poet once wrote, "*All the world's a stage, and all the men and women merely players*..." a line that has sparked countless interpretations and heated debates. If we were to add just a few words to this "*though sometimes they are poets and directors*" the meaning would shift in a profound and productive way. This analogy proves useful when a computer science student encounters the "Actor Model" within the context of distributed architectures. It turns out that, alongside symphonies, such a student should also read Shakespeare "Hamlet" ("*Words, words, words*...") and "A Midsummer Night's Dream" *("...and all the men and women merely players*..."), as well as Goethe's "Faust", where, in the Prologue, a poet, an actor, and a director discuss how they might coexist in harmony.

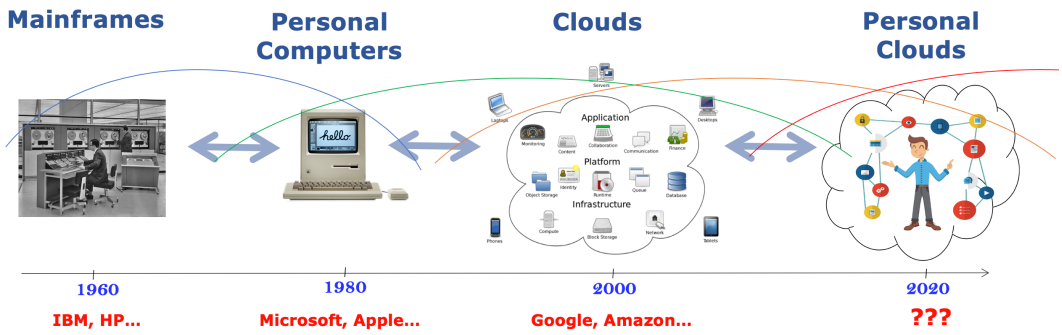
But actors are performers, devoid of freedom. For musicians and artists, improvisation is possible only to a limited extent. Performances of a Beethoven concerto by a symphony orchestra, or of a Shakespearean tragedy in the theater, allow for only very minor deviations from the author's original work. Where, then, is freedom to be found? In art, only the creator is free from all constraints. Yet there exists another spectacle – one that, since ancient times, has drawn far larger audiences than art ever has: the spectacle of sport. "*In sports, performers enjoy significantly more freedom than actors or musicians do (D. Gephard)*". This is likely why audiences enjoy watching hockey, basketball, or boxing – games in which actors, whether as part of a team or as individuals, play roles that they can improvise on the fly. However, there is a catch: the outward manifestation of their performance consists entirely of highly complex muscular movements; it is devoid of words and cannot be captured in the form of a book or a musical score. Within an athlete's mind, a sophisticated symphony of analytical processes may be unfolding, yet these remain impossible to see or comprehend. The decisions they make: where to skate, how and to whom to pass the puck, are rooted in processes of a nature entirely different from the analytical logic of the "*if… then… because…*"

Continuing our discussion on sports, we arrived at the question: how are athletes *taught*, or, more accurately, *trained*? The latter term resonates more closely with the field of artificial intelligence, offering us a useful analogy. The two concepts: *teaching* and *training* reflect distinct approaches within modern computer science. We *teach* a computer using words and algorithms, but by employing datasets, we *train* models in which the

relationships between objects possess dynamic properties. If one were to provide an athlete with a formal, written set of instructions on how to run, it is clear they would get nowhere. They must train, and train, and train again. Yet the high priests of artificial intelligence are attempting to create something similar, termed "clinical algorithms", for use by physicians. How can these two fundamentally different approaches be reconciled? Our natural language and common sense offer a clue: consciousness and the subconscious, mechanics and thermodynamics, understanding and feeling, algorithms and models, melody and symphony, good and bad, man and woman. Men tend toward understanding, women, toward feeling. Could this be yet another dimension in a harmonic polyphony? And if so, perhaps it is best not to attempt to explain one realm using the methods of another.

Why do people listen to music, engage in sports, go to the theater, and read books? It is an imprecise question, yet an interesting one. Today, as in the past, only a few people read classical literature. Music, too, is incredibly diverse: there is a vast distance or perhaps a whole spectrum between a Taylor Swift pop concert and Puccini's opera "La Bohème". A spectrum, indeed! This leads to another important conclusion for programmers: personal systems must adapt to the unique characteristics of each individual. At one time, large mainframe computers gave way to personal computers, accompanied by millions of software applications designed to tailor the device to the individual user. However, while terms like "personal computer" or "my smartphone" are widely used and understood in everyday life, the phrase "personal program" sounds unnatural, it would likely cause some confusion, not only among professionals but even for an artificial intelligence. There is a significant distinction between the term "application" (or "applied program") and "personal program." The notion that "*each of us has our own cultural program*" can help us decide whether the concept of "personal programming" makes sense.

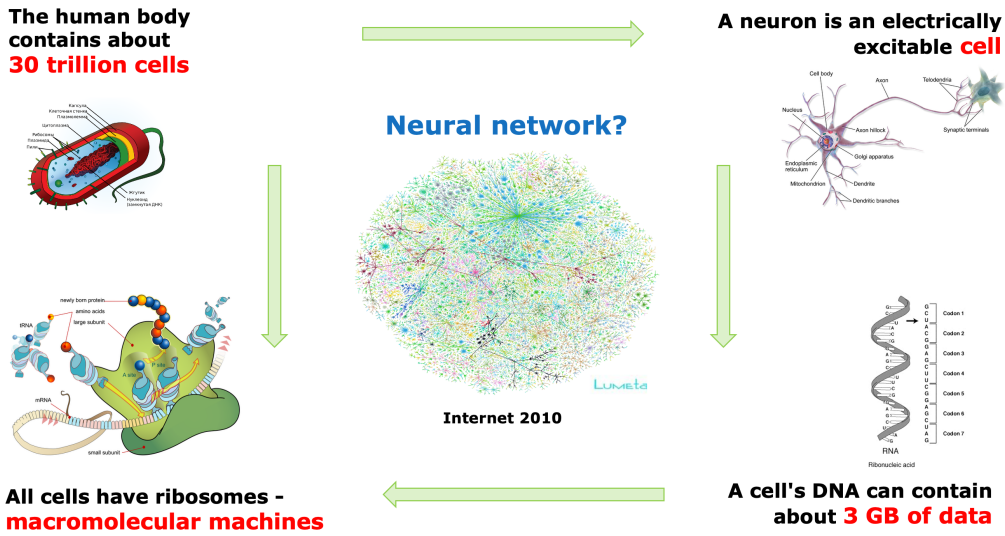
We devote so much time to art for a simple reason: programming is entering a new phase, in which the primary focus will be the integration of personal services, those surrounding an individual with external services hosted in commercial clouds. One of the key tasks for developers of personal clouds will be to expand human intellectual capabilities and protect individual identity against the mass standardization imposed by commercial systems. Art serves as an ideal prototype here: its perception is deeply individual, and its forms of expression are diverse.



The work of great composers, poets, and artists is striking for its sheer productivity, the vast amount one individual can create within their professional field, whether in the arts or

the sciences, when they approach their craft creatively rather than merely perfunctorily. Programming, too, has had, and continues to have, its own geniuses, individuals who have built large, cohesive systems almost entirely on their own. Yet, unlike music, software tends to perish along with the environment in which it was written. It is no wonder that programmers often joke that, sooner or later, every piece of software ends up in the trash bin.

A concerto composed by Mozart 300 years ago can still be heard in various performances; by contrast, it is no longer possible to continue working within HyperCard – a visual programming environment created by Bill Atkinson in the mid-1980s. HyperCard was once highly popular among professional content creators, as well as in the fields of education, science, and publishing. More than a million users built "stacks" of dynamic cards containing a wide array of material. Some even argue that it could have served as the foundation for the World Wide Web. Ultimately, however, it ended up in Apple's trash bin; today, one can only use it by running a virtual machine that emulates a Macintosh computer from that era.



Why do musical compositions endure for a long time, while software programs do not – and what can be done to change this? Paradoxically, if we compare the lifespans of various forms of notation, programming might actually come out on top. Programs written millions of years ago have recently been discovered, programs that can still be executed today on what is, in effect, the world's oldest computer: DNA and the ribosome. Setting aside the chemical aspects of cellular research, one can view the cell as an information-processing machine: DNA acts as the program and data (in machine code), while the ribosome serves as a specialized cellular processor that interprets them. This simplification is valid, provided one understands that it represents only the highest level of a structural-functional model of the cell. Furthermore, the complexity of a ribosome is comparable to that of a modern computer; likewise, the complexity and scale of the code found in, say, the iOS operating system are comparable to those of the DNA within a neuron.

If we liken the ribosome and DNA to the hardware and software of a computer, then the next level in our structural model should define the configuration of the device's functional

components and its operating system. According to Google, a single human body contains over 30 trillion cells of several types: somatic, motor, nerve, and others. For comparison, there are about 80 billion processors in the world; if we were to apply a biological classification system to computers, we would find a similar diversity of types. Beyond the processor, memory, and interfaces, every computer has its own operating system – such as Android, macOS, Linux, or Windows. Together, a computer and its operating system form a pair designed to handle a specific range of tasks. The mass production of both computers and cells results in identical copies, but what happens afterward?

Once an operating system is installed on a device and it is powered on, this symbiosis begins to lead a life of its own. With every clock cycle, and with every byte of data received or transmitted, the device becomes increasingly unique. A biological cell behaves in much the same way. Interaction with the external environment is a fundamental property of such systems: both computers and cells take in and release information and energy, a process that shapes them into distinct individuals. The terms "information" and "energy" carry a multitude of meanings, ranging from the mundane and every day to the profoundly philosophical. According to basic definitions, energy and information cannot be "transferred." They are not substances like some invisible fluid, but rather measures, akin to weight or length. These measures define a crucial property of both living and non-living systems: their internal state and their capacity to alter the state of other systems. The extent to which a given system can *heat* or *educate* another system constitutes its energy-information measure. Beneath this veneer of scientific-sounding terminology lies a model of immense significance to our entire existence: a model of harmony between the consumption and production of energy and information.

Like any system in the real world, a human being exists in a state of equilibrium with their environment. By consuming energy and information, one must return an equivalent amount. A natural way to achieve this is through work, whether productive, useful labor or some other form of activity, that entails an expenditure equivalent to what was consumed. If a machine, whether informational or physical, takes on a portion of this work, the individual must compensate for it through other work or activity. Lowering the strain on a person without offering compensation may lead to degradation. A simple illustration can be found in two school subjects: in physical education, a student might run two laps around a track; with a bicycle, they would need ten laps to match the energy used. Likewise, in mathematics, if a student uses a calculator for multiplication, they should be assigned a different logical exercise of comparable complexity.

The energy-information model makes it possible to objectively measure and compare the integrated characteristics of cells, computers, individual humans, and groups. Furthermore, drawing parallels with well-understood physical systems can assist in the analysis and synthesis of information-based analogs. Consider, for instance, the purpose of a battery: we charge it so that it can provide the energy required to perform physical work. Now, imagine an "information battery" that also requires charging. This sheds light on why people ask questions, attend the theater, or read it is a form of charging. The specific medium through which an individual acquires data, whether via books, conversation, or a computer screen, is less important than the efficient conversion of that data into useful information, which can eventually be applied to perform "information work." Concepts such as work, power,

and productivity, as defined in physical models, have direct counterparts in the realm of information. Similarly, the relationship between *mass* and *energy* expressed in Albert Einstein's equation (E ~ M) may have an analog in the relationship between *data* and *information* (I ~ D). While such hypotheses certainly require a rigorous scientific foundation, they are sufficiently intuitive for programmers to adopt as a practical, common-sense framework. However, an understanding of this model should form the basis for moral standards in any modern engineering project:

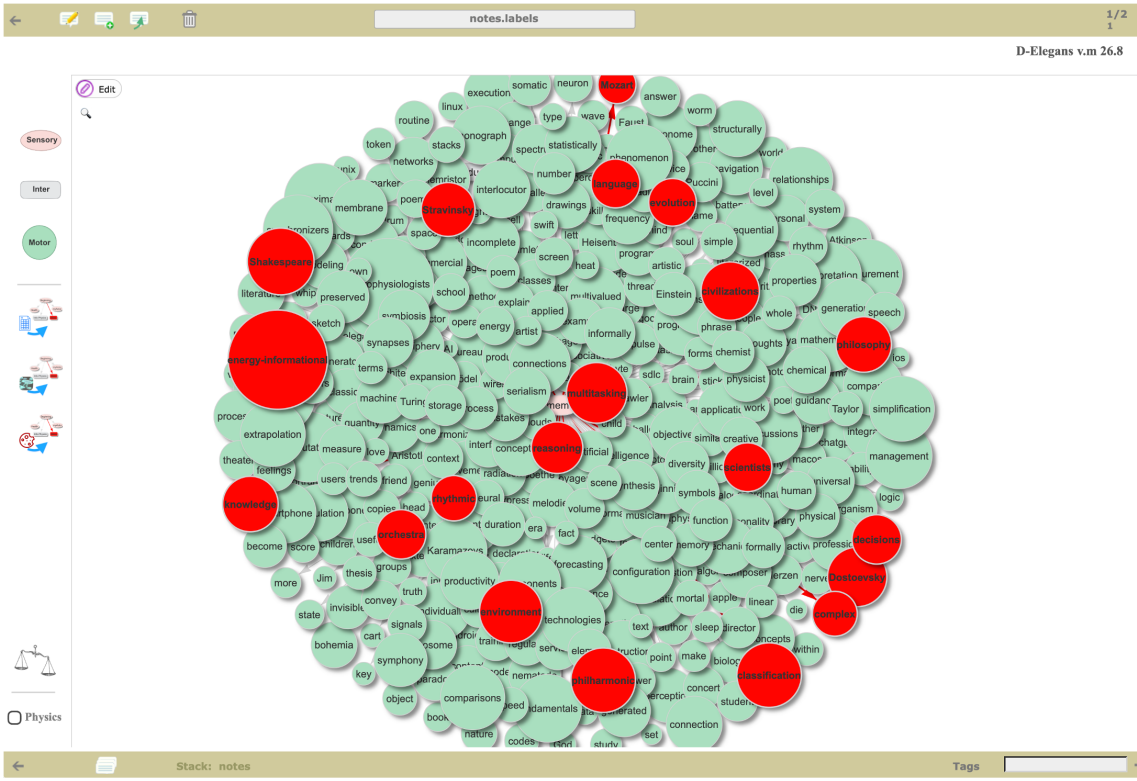
*If a person is replaced by a **machine** or **system** without being provided with a new job that requires a comparable level of informational and physical work, then such machines or systems should be regarded as weapons.*

DNA can endure for a very long time, yet all living things are mortal. This is the paradox of programming: a device along with all its contents may soon perish, while what was stored in its memory can persist. But what is worth preserving? Just the program? Just the data? Or the whole entity, as a virtual machine? And perhaps there are models even more efficient and universal than a virtual machine? What becomes of the combination of programs and data once a device begins its operation and once it concludes? It turns out that a program is not merely what the original programmer wrote; it is also everything subsequently added or modified, continuing its existence in a new form. We are all mortal, that is a fact. Yet, something remains after us: children, books, programs, memories… Children! Therein lies the answer. A "child" cell receives the initial program and data, then begins a new life: renewing itself and carrying the process forward to the next generation. Some might call this evolution; programmers call it a *Software Development Lifecycle* (SDLC).

In traditional programming, data and programs are strictly separated, a distinction that seems intuitive and follows from the mathematical foundations of functions and algorithms. A computer is a rigid structure in which connections act as "wires" with constant resistance. The role of these connections is to transmit signals without distortion; any deviation is considered an error. Biological neural networks are organized in a fundamentally different way: their connections (synapses) possess memory and are capable of altering their own transfer functions. In electronics, the memristor – a component that can change its resistance based on the current that has previously passed through it, serves as an analogue for this behavior. However, a memristor is an extremely simplified model, lacking any dynamically programmable functional properties. In contrast, a biological neuron can reconfigure its own structure, much like a programmer modifies software. Computers with such capabilities will not be available to us in the near future. Yet, programmers have an alternative: virtual neurons! Even if not numbering in the billions, a few hundred virtual neurons could be more than enough to construct useful models. The simplest biological neural model on Earth is found in the nematode "C. elegans". The worm's neural network consists of 300 neurons, and this biological computer performs a variety of useful functions.

And now, the time has come to move from words to neurons. At this point, we conclude the introduction and transition to neural programming. To do this, we will use an assistant, in our case, the application **myActor**, which will take on the roles of both programmer and secretary. Since everything in the world is interconnected, the words written above must,

in some way, be associated with what follows. On one hand, the words in the introduction constitute a complete and defined set; on the other, what comes next is an open space of potential possibilities.



First Neural Program

One of the author's tasks is to keep track of what has already been written. Thus, extracting key terms from the preceding text, organizing them into categories, creating a table of contents (the book's structure), linking these elements together, and presenting them in a convenient visual format, such an approach is quite useful. Something similar can be achieved using applied knowledge graph models; however, in our case, this graph-like visualization is actually a network of virtual neurons, functionally analogous to the neurons of C-elegans. As for the *personal program* **myActor** it acts as "My actor," performing a role according to the author's script and under the author's guidance.

Literature

1. Aristotle. *"The Basic Works of Aristotle"*
(New York: The Modern Library, 2001.)

2. John Backus. *"Can programming be liberated from the von Neumann style?"*
(Communications of the ACM., 1978.)

3. Timothy Berners-Lee. "*Information Management: A Proposal*"
(1989, May 1990)
www.w3.org/history/1989/proposal.htm

4. Leo Breiman. *"Statistical Modeling: The Two Cultures"*
(Statistical Science. Vol 16, No. 3, 2001)

5. Frederick P. Brooks, Jr. *"The Mythical Man Month."*
(Boston: Addison Wesley, 1995.)

6. Vannevar Bush. *"As We May Think"*
(1945 Endless Horizons, Washington, D.C., Public Affairs Press, 1975; originally published in Atlantic Monthly, July 1945)

7. Vladimir Cherkassky. *"Predictive Learning."*
(United States, vctextbook.com, 2013.)

8. Noam Chomsky, David W. Lightfoot *"Syntactic Structures"*
(Walter de Gruyter, 2002.)

9. Edsger W. Dijkstra. *"Denken Als Discipline"*
(Episode of the science series, Noorderlicht, 2001)

10. Dijkstra E.W. *"Cooperating Sequential Processes."*
(London: Academic Press, 1968.)

11. Albert Einstein. *"Relativity. The Special and the General Theory"*
(New York: Three Rivers Press, 1952; 1961)

12. Christiane Fellbaum (editor). *"WordNet: An Electronic Lexical Database"*
(Cambridge, MA: MIT Press; 1998.)

13. Richard P. Feynman. *"Feynman Lectures on Computation"*
(Cambridge, Mass.: Perseus Publishing, 1996; 1999)

14. James Gleick. *"The Information: A History, a Theory, a Flood."*
(New York: Pantheon Books, 2011.)

15. Kurt Gödel. *"On Formally Undecidable Propositions of Principia Mathematica and Related Systems."* (New York, Dover Publications, Inc., 1992.)

16. Danny Goodman. *"The Complete HyperCard Handbook"*
(New York: *A Bantam Book*; 1987)
https://archive.org/details/The_Complete_HyperCard_Handbook/

17. Werner Heisenberg. *"The Physical Principles of the Quantum Theory."*
(Leipzig, Dover Publications, Inc., 1949.)

18. Werner Heisenberg. *"Physics and Philosophy. The Revolution in Modern Science."*
(New York, Prometheus Books., 1999.)

19. Ingram J. *"The Notation of Time."*
(Contact Magazine, London, 1985.)

20. McCulloch, W. and Pitts, W. *"A logical calculus of the ideas immanent in nervous activity"*
(*Bulletin of Mathematical Biophysics*, 7:115 – 133; 1943)

21. Satoshi Nakamoto. *"Bitcoin: A Peer-to-Peer Electronic Cash System"*
(Archived from the original; 2014)
https://bitcoin.org/bitcoin.pdf

22. Nelson, T. H. *"The Future of Information: Ideas, Connections, and the Gods of Electronic Literature."* (Tokyo: ASCII Corporation, 1997)

23. John von Neumann. *"The Computer and the Brain"*
(Second Edition 1958, New Haven, Conn.: Yale University Press: 2000)

24. John von Neumann. *"First Draft of a Report on the EDVAC"*
(30 June 1945, Contract W-670-ORD4926, Philadelphia, PA, Morse School of Electrical Engineering, University of Pennsylvania)

25. Noran, Ovidiu S. *"Business Modelling: UML vs. IDEF."*
(PDF., 2005.)

26. Stuart Russell and Peter Norvig. *"Artificial Intelligence: A Modern Approach"*
(Upper Saddle River, NJ, Prentice Hall, 2010.)

27. Claude E. Shannon. *"A Mathematical Theory of Communication"*
(The Bell System Technical Journal, Vol. 27, July, October 1948: 379-423, 623-656)

28. J. F. Stay. *"HIPO and integrated program design."*
(IBM Systems Journal. Volume 15, Number 2, Page 143, 1976.)

29. Igor Stravinsky, Robert Craft. *"Conversations with Igor Stravinsky."*
(Berkeley: University of California Press, 1980.)

30. S. Tolkachev. *"Aggregator, filter and delivery system for online context dependent interaction, systems and methods"*
(U.S. Patent 9305050, 2016.)

31. S. Tolkachev. *"Active Data: Philosophical Programming and Business Ideas"*
(Amazon, 2022.)

32. Turing, A. M. *"COMPUTING MACHINERY AND INTELLIGENCE."*
(Mind, Volume LIX, Issue 236, October 1950.)

33. Turing, A. M. *"On Computable Numbers, with an Application to the Entscheidungsproblem."*
(Proceedings of the London Mathematical Society, 2, 42 (1), pp. 230–65, 1937.)

34. Vladimir Vapnik. *"Intelligent Learning: Similarity Control and Knowledge Transfer."*
(Yandex School of Data Analysis Conference, 2015.)

35. Watson, J. D. and Crick, F. H. C. *"Genetical implications of the structure of deoxyribonucleic acid."*
(Nature 171, May 1953.)

36. Joseph Weizenbaum *"ELIZA – a computer program for the study of natural language communication between man and machine."*
(Communications of the ACM, Volume 9 Issue 1, Jan. 1966.)

37. Terry Winograd, Fernando Flores. *"Understanding Computers and Cognition: A New Foundation for Design"*
(Addison-Wesley, 1987.)

38. Niklaus Wirth. *"Algorithms + Data Structures = Programs"*
(Englewood Cliffs: Prentice-Hall, 1976.)

39. Апатова Н. В. *"Теория информационной экономики"*
(Симферополь, ТНУ, 2005)

40. Барский А. Б. *"Логические нейронные сети."*
(Москва: НОУ «Интуит», 2016)

41. Н. Бурбаки *"ОСНОВНЫЕ СРУКТУРЫ АНАЛИЗА. ТЕОРИЯ МНОЖЕСТВ"*
(Москва: Мир, 1965.).

42. Валентинов, Н. *"Встречи с Лениным"*
(Нью Йорк, Chalidze Publ., 1981.)

43. Вернадский В. И. *"Труды по всеобщей истории науки"*
(Москва, «Наука», 1988)

44. Герман Вейль. *"Математическое мышление."*
(Москва: Наука, 1989.)

45. Винер Н. *"Кибернетика, или управление и связь в животном и машине."*
(М.: Наука, 1983.)

46. Глушков В. М. *"Основы безбумажной информатики."*
(Москва: Наука., 1987.)

47. Джермейн, Кларенс Б. *"Программирование на IBM-360."*
(Москва : Мир., 1971.)

48. Рене Декарт. *"Рассуждение о методе."*
(Москва: АН СССР., 1953.)

49. Заде Л. *"Понятие лингвистической переменной и его применение к принятию приближенных решений"*
(Москва: Мир., 1976.)

50. Кнут Д. Э. *"Искусство программирования. Том 1. Основные алгоритмы"*
(Москва: Вильямс, 2002.)

51. Колмогоров А. Н. *"Теория информации и теория алгоритмов."*
(М.: Наука, 1987.)

52. Леонтьев В. К., Гордеев Э. Н. *"Комбинаторные аспекты теории информации."*
(Москва: МФТИ, 2019.)

53. Николлс Дж. Г., Мартин А. Р., Валлас Б. Дж., Фукс П.А. *"От нейрона к мозгу"*
(Москва: УРСС, 2019)

54. Платон. *"Диалоги"*
(Издательство Эксмо, 2016.)

55. Пушкин А. С. *"Разговор книгопродавца с поэтом."*
(Москва: АН СССР., 1953.)

56. Разова Е. Г. *"Из истории нотной записи."*
(Новосибирск: Муниципальное бюджетное учреждение, 2015.)

57. Раушенбах Б. В. *"Пространственные построения в древнерусской живописи"*
(Москва: Издательство Наука., 1975.)

58. Раушенбах Б. В. *"Геометрия картины и зрительное восприятие"* (СПб: Азбука-классика., 2002.)

59. Альфред Реньи. *"Диалоги о математике."* (Москва: Мир., 1969.)

60. Леонид Сабанеев *"Музыка Речи."* (Москва: Работник просвещения, 1923.)

61. Савельев С. В., Негашева М. А. *"Практикум по анатомии мозга человека."* (Москва: ВЕДИ, 2001)

62. *"Сборник научных подпрограмм на ФОРТРАНе."* (Москва: Статистика, 1974)

63. Сеченов И. М. *"Рефлексы головного мозга"* (В кн.: Сеченов И. М., Павлов И. П., Введенский Н. Е. Физиология нервной системы. Москва, Государственное издательство медицинской литературы, 1952.)

64. Аркадий и Борис Стругацкие. *" Понедельник начинается в субботу."* (Москва: Детская литература., 1965.)

65. Толкачев С. *"Нейронное программирование диалоговых систем"* (Издательство Корона-Век, 2006.)

66. Л. Н. Толстой *"Воспитание и образование. Собрание сочинений в 22 т."* (Москва: Художественная литература, 1986.)

67. Л. Н. Толстой *"Война и мир. Собрание сочинений в восьми томах."* (Москва: Лексика, 1996.)

68. Успенский В. А. *"К семиотическим посланиям Андрея Николаевича Колмогорова."* (НЛО, 24.1978.)

69. Хижняков Ю. Н. *"Алгоритмы нечеткого, нейронного и нейро-нечеткого управления в системах реального времени."* (Пермь: Нац. исслед. политехн. университет, 2013)

70. К. Шеннон. *"Работы по теории информации и кибернетике."* (Москва, Изд. Иностранной литературы, 1963.)