جزوه

زبان تخصصی کامپیوتر

دوره کارشناسی

آموزشکده فنی حرفه‌ای

پاییز ۹۷

Computer

From Wikipedia, the free encyclopedia

Computer

Computers and computing devices from different eras

A **computer** is a device that can be instructed to carry out sequences of arithmetic or logical operations automatically via computer programming. Modern computers have the ability to follow generalized sets of operations, called *programs*. These programs enable computers to perform an extremely wide range of tasks.

Computers are used as control systems for a wide variety of industrial and consumer devices. This includes simple special purpose devices like microwave ovens and remote controls, factory devices such as industrial robots and computer-aided design, and also general purpose devices like personal computers and mobile devices such as smartphones.

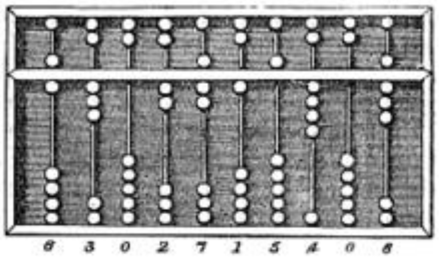
Early computers were only conceived as calculating devices. Since ancient times, simple manual devices like the abacus aided people in doing calculations. Early in the Industrial Revolution, some mechanical devices were built to automate long tedious tasks, such as guiding patterns for looms. More sophisticated electrical machines did specialized analog calculations in the early 20th century. The first digital electronic calculating machines were developed during World War II. The speed, power, and versatility of computers have been increasing dramatically ever since then.

Conventionally, a modern computer consists of at least one processing element, typically a central processing unit (CPU), and some form of memory. The processing element carries out arithmetic and logical operations, and a sequencing and control unit can change the order of operations in response to stored information. Peripheral devices include input devices (keyboards, mice, joystick, etc.), output devices (monitor screens, printers, etc.), and input/output devices that perform both functions (e.g., the 2000s-era touchscreen). Peripheral devices allow information to be retrieved from an external source and they enable the result of operations to be saved and retrieved.

Etymology

A female computer, with microscope and calculator, 1952

According to the *Oxford English Dictionary*, the first known use of the word "computer" was in 1613 in a book called *The Yong Mans Gleanings* by English writer Richard Braithwait: "I haue [sic] read the truest computer of Times, and the best Arithmetician that euer [sic] breathed, and he reduceth thy dayes into a short number." This usage of the term referred to a human computer, a person who carried out calculations or computations. The word continued with the same meaning until the middle of the 20th century. From the end of the 19th century, the word began to take on its more familiar meaning, a machine that carries out computations.[1]



The *Online Etymology Dictionary* gives the first attested use of "computer" in the "1640s, [meaning] "one who calculates,"; this is an "... agent noun from compute (v.)". The *Online Etymology Dictionary* states that the use of the term to mean "calculating machine" (of any type) is from 1897." The *Online Etymology Dictionary* indicates that the "modern use" of the term, to mean "programmable digital electronic computer" dates from "... 1945 under this name; [in a] theoretical [sense] from 1937, as Turing machine".[2]

History

Pre-20th century

The Ishango bone

Devices have been used to aid computation for thousands of years, mostly using one-to-one correspondence with fingers. The earliest counting device was probably a form of tally stick. Later record keeping aids throughout the Fertile Crescent included calculi (clay spheres, cones, etc.) which represented counts of items, probably livestock or grains, sealed in hollow unbaked clay containers.[3][4] The use of counting rods is one example.

The Chinese Suanpan (算盘) (the number represented on this abacus is 6,302,715,408)

The abacus was initially used for arithmetic tasks. The Roman abacus was developed from devices used in Babylonia as early as 2400 BC. Since then, many other forms of reckoning boards or tables have been invented. In a medieval European counting house, a checkered cloth would be placed on a table, and markers moved around on it according to certain rules, as an aid to calculating sums of money.

The ancient Greek-designed Antikythera mechanism, dating between 150 and 100 BC, is the world's oldest analog computer.

The Antikythera mechanism is believed to be the earliest mechanical analog "computer", according to Derek J. de Solla Price.[5] It was designed to calculate astronomical positions. It was discovered in 1901 in the Antikythera wreck off the Greek island of Antikythera, between Kythera and Crete, and has been dated to *circa* 100 BC. Devices of a level of complexity comparable to that of the Antikythera mechanism would not reappear until a thousand years later.

Many mechanical aids to calculation and measurement were constructed for astronomical and navigation use. The planisphere was a star chart invented by Abū Rayhān al-Bīrūnī in the early 11th century.[6] The astrolabe was invented in the Hellenistic world either in the 1st or 2nd centuries BC and is often attributed to Hipparchus. A combination of the planisphere and dioptra, the astrolabe was effectively an analog computer capable of working out several different kinds of problems in spherical astronomy. An astrolabe incorporating a mechanical calendar computer[7][8] and gear-wheels was invented by Abi Bakr of Isfahan, Persia in 1235.[9] Abū Rayhān al-Bīrūnī invented the first mechanical geared lunisolar calendar astrolabe,[10] an early fixed-wired knowledge processing machine[11] with a gear train and gear-wheels,[12] *circa* 1000 AD.

The sector, a calculating instrument used for solving problems in proportion, trigonometry, multiplication and division, and for various functions, such as squares and cube roots, was developed in the late 16th century and found application in gunnery, surveying and navigation.

The planimeter was a manual instrument to calculate the area of a closed figure by tracing over it with a mechanical linkage.

A slide rule

The slide rule was invented around 1620–1630, shortly after the publication of the concept of the logarithm. It is a hand-operated analog computer for doing multiplication and division. As slide rule development progressed, added scales provided reciprocals, squares and square roots, cubes and cube roots, as well as transcendental functions such as logarithms and exponentials, circular and hyperbolic trigonometry and other functions. Slide rules with special scales are still used for quick performance of routine calculations, such as the E6B circular slide rule used for time and distance calculations on light aircraft.

In the 1770s, Pierre Jaquet-Droz, a Swiss watchmaker, built a mechanical doll (automata) that could write holding a quill pen. By switching the number and order of its internal wheels different letters, and hence different messages, could be produced. In effect, it could be mechanically "programmed" to read instructions. Along with two other complex machines, the doll is at the Musée d'Art ET d'Histoire of Neuchâtel, Switzerland, and still operates.[13]

The tide-predicting machine invented by Sir William Thomson in 1872 was of great utility to navigation in shallow waters. It used a system of pulleys and wires to automatically calculate predicted tide levels for a set period at a particular location.

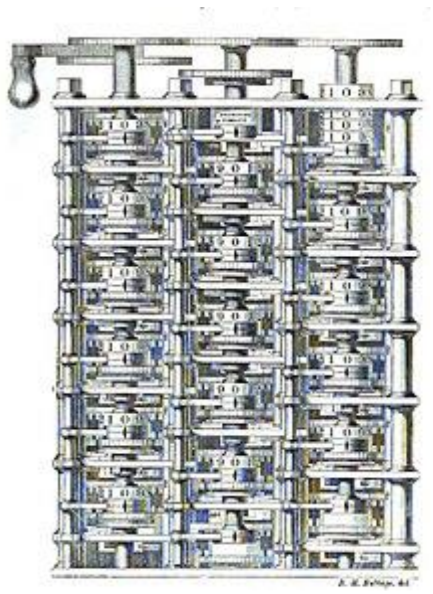
The differential analyser, a mechanical analog computer designed to solve differential equations by integration, used wheel-and-disc mechanisms to perform the integration. Lord Kelvin had already discussed the possible construction of such calculators In 1876, but he had been stymied by the limited output torque of the ball-and-disk integrators.[14] In a differential analyzer, the output of one integrator drove the input of the next integrator, or a graphing output. The torque amplifier was the advance that allowed these machines to work. Starting in the 1920s, Vannevar Bush and others developed mechanical differential analyzers.

First computing device

A portion of Babbage's Difference engine.

Charles Babbage, an English mechanical engineer and polymath, originated the concept of a programmable computer. Considered the "father of the computer", [15] he conceptualized and invented the first mechanical computer in the early 19th century. After working on his revolutionary difference engine, designed to aid in navigational calculations, in 1833 he realized that a much more general design, an Analytical Engine, was possible. The input of programs and data was to be provided to the machine via punched cards, a method being used at the time to direct mechanical looms such as the Jacquard loom. For output, the machine would have a printer, a curve plotter and a bell. The machine would also be able to punch numbers onto cards to be read in later. The Engine incorporated an arithmetic logic unit, control flow in the form of conditional branching and loops, and integrated memory, making it the first design for a general-purpose computer that could be described in modern terms as Turing-complete.[16] [17]

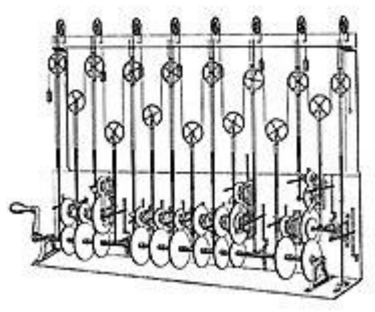
The machine was about a century ahead of its time. All the parts for his machine had to be made by hand – this was a major problem for a device with thousands of parts. Eventually, the project was dissolved with the decision of the British Government to cease funding. Babbage's failure to complete the analytical engine can be chiefly attributed to difficulties not only of politics and financing, but also to his desire to develop an increasingly sophisticated computer and to move ahead faster than anyone else could follow. Nevertheless, his son, Henry Babbage, completed a simplified version of the analytical engine's computing unit (the *mill*) in 1888. He gave a successful demonstration of its use in computing tables in 1906.



Analog computers

Sir William Thomson's third tide-predicting machine design, 1879–81

During the first half of the 20th century, many scientific computing needs were met by increasingly sophisticated analog computers, which used a direct mechanical or electrical model of the problem as a basis for computation. However, these were not programmable and generally lacked the versatility and accuracy of modern digital computers.[18] The first modern analog computer was a tide-predicting machine, invented by Sir William Thomson in 1872. The differential analyser, a mechanical analog computer designed to solve differential equations by integration using wheel-and-disc mechanisms, was conceptualized in 1876 by James Thomson, the brother of the more famous Lord Kelvin.[14]



The art of mechanical analog computing reached its zenith with the differential analyzer, built by H. L. Hazen and Vannevar Bush at MIT starting in 1927. This built on the mechanical integrators of James Thomson and the torque amplifiers invented by H. W. Nieman. A dozen of these devices were built before their obsolescence became obvious. By the 1950s, the success of digital electronic computers had spelled the end for most analog computing machines, but analog computers remained in use during the 1950s in some specialized applications such as education (control systems) and aircraft (slide rule).

Digital computers

It has been suggested that this section be split out into another article titled *Digital computer*. (Discuss) *(May 2017)*

Electromechanical

By 1938, the United States Navy had developed an electromechanical analog computer small enough to use aboard a submarine. This was the Torpedo Data Computer, which used trigonometry to solve the problem of firing a torpedo at a moving target. During World War II, similar devices were developed in other countries as well.

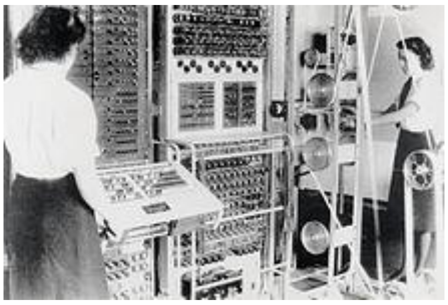
Replica of Zuse's Z3, the first fully automatic, digital (electromechanical) computer.

Early digital computers were electromechanical; electric switches drove mechanical relays to perform the calculation. These devices had a low operating speed and were eventually superseded by much faster all-electric computers, originally using vacuum tubes. The Z2, created by German engineer Konrad Zuse in 1939, was one of the earliest examples of an electromechanical relay computer.[19]

In 1941, Zuse followed his earlier machine up with the Z3, the world's first working electromechanical programmable, fully automatic digital computer.[20][21] The Z3 was built with 2000 relays, implementing a 22 bit word length that operated at a clock frequency of about 5–10 Hz.[22] Program code was supplied on punched film while data could be stored in 64 words of memory or supplied from the keyboard. It was quite similar to modern machines in some respects, pioneering numerous advances such as floating point numbers. Rather than the harder-to-implement decimal system (used in Charles Babbage's earlier design), using a binary system meant that Zuse's machines were easier to build and potentially more reliable, given the technologies available at that time.[23] The Z3 was Turing complete.[24][25]

Vacuum tubes and digital electronic circuits

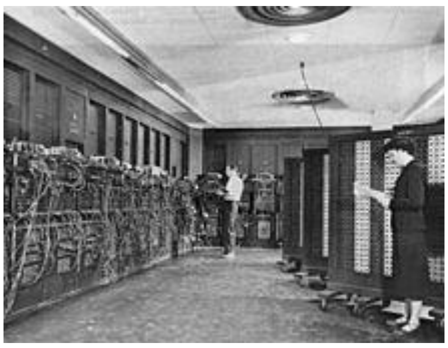
Purely electronic circuit elements soon replaced their mechanical and electromechanical equivalents, at the same time that digital calculation replaced analog. The engineer Tommy Flowers, working at the Post Office Research Station in London in the 1930s, began to explore the possible use of electronics for the telephone exchange. Experimental equipment that he built in 1934 went into operation five years later, converting a portion of the telephone exchange network into an electronic data processing system, using thousands of vacuum tubes.[18] In the US, John Vincent Atanasoff and Clifford E. Berry of Iowa State University developed and tested the Atanasoff–Berry Computer (ABC) in 1942,[26] the first "automatic electronic digital computer".[27] This design was also all-electronic and used about 300 vacuum tubes, with capacitors fixed in a mechanically rotating drum for memory.[28]



Colossus, the first electronic digital programmable computing device, was used to break German ciphers during World War II.

During World War II, the British at Bletchley Park achieved a number of successes at breaking encrypted German military communications. The German encryption machine, Enigma, was first attacked with the help of the electro-mechanical bombes. To crack the more sophisticated German Lorenz SZ 40/42 machine, used for high-level Army communications, Max Newman and his colleagues commissioned Flowers to build the Colossus.[28] He spent eleven months from early February 1943 designing and building the first Colossus.[29] After a functional test in December 1943, Colossus was shipped to Bletchley Park, where it was delivered on 18 January 1944[30] and attacked its first message on 5 February.[28]

Colossus was the world's first electronic digital programmable computer.[18] It used a large number of valves (vacuum tubes). It had paper-tape input and was capable of being configured to perform a variety of Boolean logical operations on its data, but it was not Turing-complete. Nine Mk II Colossi were built (The Mk I was converted to an Mk II making ten machines in total). Colossus Mark I contained 1,500 thermionic valves (tubes), but Mark II with 2,400 valves, was both 5 times faster and simpler to operate than Mark I, greatly speeding the decoding process.[31][32]



ENIAC was the first electronic, Turing-complete device, and performed ballistics trajectory calculations for the United States Army.

The U.S.-built ENIAC [33] (Electronic Numerical Integrator and Computer) was the first electronic programmable computer built in the

US. Although the ENIAC was similar to the Colossus, it was much faster, more flexible, and it was Turing-complete. Like the Colossus, a "program" on the ENIAC was defined by the states of its patch cables and switches, very different from the stored program electronic machines that came later. Once a program was written, it had to be mechanically set into the machine with manual resetting of plugs and switches.

It combined the high speed of electronics with the ability to be programmed for many complex problems. It could add or subtract 5000 times a second, a thousand times faster than any other machine. It also had modules to multiply, divide, and square root. High-speed memory was limited to 20 words (about 80 bytes). Built under the direction of John Mauchly and J. Presper Eckert at the University of Pennsylvania, ENIAC's development and construction lasted from 1943 to full operation at the end of 1945. The machine was huge, weighing 30 tons, using 200 kilowatts of electric power and contained over 18,000 vacuum tubes, 1,500 relays, and hundreds of thousands of resistors, capacitors, and inductors.[34]

Modern computers

Concept of modern computer

The principle of the modern computer was proposed by Alan Turing in his seminal 1936 paper, [35] *On Computable Numbers*. Turing proposed a simple device that he called "Universal Computing machine" and that is now known as a universal Turing machine. He proved that such a machine is capable of computing anything that is computable by executing instructions (program) stored on tape, allowing the machine to be programmable. The fundamental concept of Turing's design is the stored program, where all the instructions for computing are stored in memory. Von Neumann acknowledged that the central concept of the modern computer was due to this paper.[36] Turing machines are to this day a central object of study in theory of computation. Except for the limitations imposed by their finite memory stores, modern computers are said to be Turing-complete, which is to say, they have algorithm execution capability equivalent to a universal Turing machine.

Stored programs

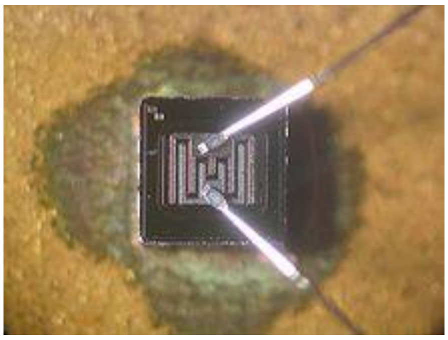
A section of the Manchester Baby, the first electronic stored-program computer

Early computing machines had fixed programs. Changing its function required the re-wiring and re-structuring of the machine.[28] With the proposal of the stored-program computer this changed. A stored-program computer includes by design an instruction set and can store in memory a set of instructions (a program) that details the computation. The theoretical basis for the stored-program computer was laid by Alan Turing in his 1936 paper. In 1945, Turing joined the National Physical Laboratory and began work on developing an electronic stored-program digital computer. His 1945 report "Proposed Electronic Calculator" was the first specification for such a device. John von Neumann at the University of Pennsylvania also circulated his *First Draft of a Report on the EDVAC* in 1945.[18]

The Manchester Baby was the world's first stored-program computer. It was built at the Victoria University of Manchester by Frederic C. Williams, Tom Kilburn and Geoff Tootill, and ran its first program on 21 June 1948.[37] It was designed as a testbed for the Williams tube, the first random-access digital storage device.[38] Although the computer was considered "small and primitive" by the standards of its time, it was the first working machine to contain all of the elements essential to a modern electronic computer.[39] As soon as the Baby had demonstrated the feasibility of its design, a project was initiated at the university to develop it into a more usable computer, the Manchester Mark 1.

The Mark 1 in turn quickly became the prototype for the Ferranti Mark 1, the world's first commercially available general-purpose computer.[40] Built by Ferranti, it was delivered to the University of Manchester in February 1951. At least seven of these later machines were delivered between 1953 and 1957, one of them to Shell labs in Amsterdam.[41] In October 1947, the directors of British catering company J. Lyons & Company decided to take an active role in promoting the commercial development of computers. The LEO I computer became operational in April 1951[42] and ran the world's first regular routine office computer job.

Transistors


A bipolar junction transistor

The bipolar transistor was invented in 1947. From 1955 onwards transistors replaced vacuum tubes in computer designs, giving rise to the "second generation" of computers. Compared to vacuum tubes, transistors have many advantages: they are smaller, and require less power than vacuum tubes, so give off less heat. Silicon junction transistors were much more reliable than vacuum tubes and had longer, indefinite, service life. Transistorized computers could contain tens of thousands of binary logic circuits in a relatively compact space.

At the University of Manchester, a team under the leadership of Tom Kilburn designed and built a machine using the newly developed transistors instead of valves.[43] Their first transistorised computer and the first in the world, was operational by 1953, and a second version was completed there in April 1955. However, the machine did make use of valves to generate its 125 kHz clock waveforms and in the circuitry to read and write on its magnetic drum memory, so it was not the first completely transistorized computer. That distinction goes to the Harwell CADET of 1955,[44] built by the electronics division of the Atomic Energy Research Establishment at Harwell.[44][45]

Integrated circuits

The next great advance in computing power came with the advent of the integrated circuit. The idea of the integrated circuit was first conceived by a radar scientist working for the Royal Radar Establishment of the Ministry of Defence, Geoffrey W.A. Dummer. Dummer presented the first public description of an integrated circuit at the Symposium on Progress in Quality Electronic Components in Washington, D.C. on 7 May 1952.[46]

The first practical ICs were invented by Jack Kilby at Texas Instruments and Robert Noyce at Fairchild Semiconductor.[47] Kilby recorded his initial ideas concerning the integrated circuit in July 1958, successfully demonstrating the first working integrated example on 12 September 1958.[48] In his patent application of 6 February 1959, Kilby described his new device as "a body of semiconductor material ... wherein all the components of the electronic circuit are completely integrated".[49][50] Noyce also came up with his own idea of an integrated circuit half a year later than Kilby.[51] His chip solved many practical problems that Kilby's had not. Produced at Fairchild Semiconductor, it was made of silicon, whereas Kilby's chip was made of germanium.

This new development heralded an explosion in the commercial and personal use of computers and led to the invention of the microprocessor. While the subject of exactly which device was the first microprocessor is contentious, partly due to lack of agreement on the exact definition of the term "microprocessor", it is largely undisputed that the first single-chip microprocessor was the Intel 4004,[52] designed and realized by Ted Hoff, Federico Faggin, and Stanley Mazor at Intel.[53]

Mobile computers

With the continued miniaturization of computing resources, and advancements in portable battery life, portable computers grew in popularity in the 2000s.[54] The same developments that spurred the growth of laptop computers and other portable computers allowed manufacturers to integrate computing resources into cellular phones. These so-called smartphones and tablets run on a variety of operating systems and have become the dominant computing device on the market, with manufacturers reporting having shipped an estimated 237 million devices in 2Q 2013.[55]

Types

Computers are typically classified based on their uses:

Based on uses

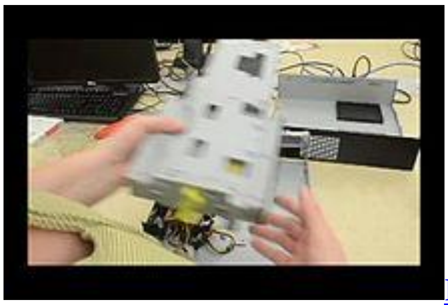
- Analog computer
- Digital computer
- Hybrid computer

Based on sizes

- Smartphone
- Microcomputer
- Workstation
- Personal computer
- Laptop
- Minicomputer
- Mainframe computer
- Supercomputer

Hardware

Main articles: Computer hardware, Personal computer hardware, Central processing unit, and Microprocessor


Play media
Video demonstrating the standard components of a "slimline" computer

The term *hardware* covers all of those parts of a computer that are tangible physical objects. Circuits, computer chips, graphic cards, sound cards, memory (RAM), motherboard, displays, power supplies, cables, keyboards, printers and "mice" input devices are all hardware.

History of computing hardware

Main article: History of computing hardware

First generation (mechanical/electromechanical)	Calculators	Pascal's calculator, Arithmometer, Difference engine, Quevedo's analytical machines
	Programmable devices	Jacquard loom, Analytical engine, IBM ASCC/Harvard Mark I, Harvard Mark II, IBM SSEC, Z1, Z2, Z3
Second generation (vacuum tubes)	Calculators	Atanasoff–Berry Computer, IBM 604, UNIVAC 60, UNIVAC 120
	Programmable devices	Colossus, ENIAC, Manchester Baby, EDSAC, Manchester Mark 1, Ferranti Pegasus, Ferranti Mercury, CSIRAC, EDVAC, UNIVAC I, IBM 701, IBM 702, IBM 650, Z22
Third generation (discrete transistors and SSI, MSI, LSI integrated circuits)	Mainframes	IBM 7090, IBM 7080, IBM System/360, BUNCH
	Minicomputer	HP 2116A, IBM System/32, IBM System/36, LINC, PDP-8, PDP-11
	Desktop Computer	Programma 101, HP 9100
Fourth generation (VLSI integrated circuits)	Minicomputer	VAX, IBM System i
	4-bit microcomputer	Intel 4004, Intel 4040
	8-bit microcomputer	Intel 8008, Intel 8080, Motorola 6800, Motorola 6809, MOS Technology 6502, Zilog Z80
	16-bit microcomputer	Intel 8088, Zilog Z8000, WDC 65816/65802
	32-bit microcomputer	Intel 80386, Pentium, Motorola 68000, ARM
	64-bit microcomputer[56]	Alpha, MIPS, PA-RISC, PowerPC, SPARC, x86-64, ARMv8-A
	Embedded computer	Intel 8048, Intel 8051
	Personal computer	Desktop computer, Home computer, Laptop computer, Personal digital assistant (PDA), Portable computer, Tablet PC, Wearable computer
Theoretical/experimental	Quantum computer, Chemical computer, DNA computing, Optical computer, Spintronics-based computer	

Other hardware topics

Peripheral device (input/output)	Input	Mouse, keyboard, joystick, image scanner, webcam, graphics tablet, microphone
	Output	Monitor, printer, loudspeaker
	Both	Floppy disk drive, hard disk drive, optical disc drive, teleprinter
Computer buses	Short range	RS-232, SCSI, PCI, USB
	Long range (computer networking)	Ethernet, ATM, FDDI

A general purpose computer has four main components: the arithmetic logic unit (ALU), the control unit, the memory, and the input and output devices (collectively termed I/O). These parts are interconnected by buses, often made of groups of wires. Inside each of these parts are thousands to trillions of small electrical circuits, which can be turned, off or on by means of an electronic switch. Each circuit represents a bit (binary digit) of information so that when the circuit is on it represents a "1", and when off it represents a "0" (in positive logic representation). The circuits are arranged in logic gates so that one or more of the circuits may control the state of one or more of the other circuits.

Input devices

When unprocessed data is sent to the computer with the help of input devices, the data is processed and sent to output devices. The input devices may be hand-operated or automated. The act of processing is mainly regulated by the CPU. Some examples of input devices are:

- Computer keyboard
- Digital camera
- Digital video
- Graphics tablet
- Image scanner
- Joystick
- Microphone
- Mouse
- Overlay keyboard
- Real-time clock
- Trackball
- Touchscreen

Output devices

The means through which computer gives output are known as output devices. Some examples of output devices are:

- Computer monitor
- Printer
- PC speaker
- Projector
- Sound card
- Video card

Control unit

Main articles: CPU design and Control unit

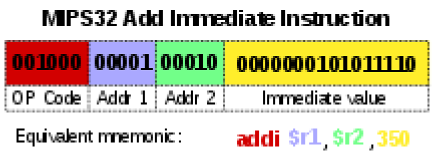


Diagram showing how a particular MIPS architecture instruction would be decoded by the control system

The control unit (often called a control system or central controller) manages the computer's various components; it reads and interprets (decodes) the program instructions, transforming them into control signals that activate other parts of the computer.[57] Control systems in advanced computers may change the order of execution of some instructions to improve performance.

A key component common to all CPUs is the program counter; a special memory cell (a register) that keeps track of which location in memory the next instruction is to be read from.[58]

The control system's function is as follows—note that this is a simplified description, and some of these steps may be performed concurrently or in a different order depending on the type of CPU:

1. Read the code for the next instruction from the cell indicated by the program counter.
2. Decode the numerical code for the instruction into a set of commands or signals for each of the other systems.
3. Increment the program counter so it points to the next instruction.
4. Read whatever data the instruction requires from cells in memory (or perhaps from an input device). The location of this required data is typically stored within the instruction code.
5. Provide the necessary data to an ALU or register.
6. If the instruction requires an ALU or specialized hardware to complete, instruct the hardware to perform the requested operation.
7. Write the result from the ALU back to a memory location or to a register or perhaps an output device.
8. Jump back to step (1).

Since the program counter is (conceptually) just another set of memory cells, it can be changed by calculations done in the ALU. Adding 100 to the program counter would cause the next instruction to be read from a place 100 locations further down the program. Instructions that modify the program counter are often known as "jumps" and allow for loops (instructions that are repeated by the computer) and often-conditional instruction execution (both examples of control flow).

The sequence of operations that the control unit goes through to process an instruction is in itself like a short computer program, and indeed, in some more complex CPU designs, there is another yet smaller computer called a microsequencer, which runs a microcode program that causes all of these events to happen.

Central processing unit (CPU)

The control unit, ALU, and registers are collectively known as a central processing unit (CPU). Early CPUs were composed of many separate components but since the mid-1970s CPUs have typically been constructed on a single integrated circuit called a *microprocessor*.

Arithmetic logic unit (ALU)

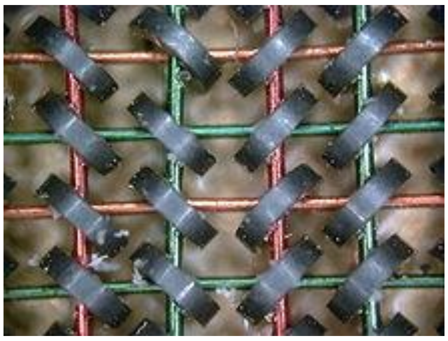
Main article: Arithmetic logic unit

The ALU is capable of performing two classes of operations: arithmetic and logic.[59] The set of arithmetic operations that a particular ALU supports may be limited to addition and subtraction, or might include multiplication, division, trigonometry functions such as sine, cosine, etc., and square roots. Some can only operate on whole numbers (integers) while others use floating point to represent real numbers, albeit with limited precision. However, any computer that is capable of performing just the simplest operations can be programmed to break down the more complex operations into simple steps that it can perform. Therefore, any computer can be programmed to perform any arithmetic operation—although it will take more time to do so if its ALU does not directly support the operation. An ALU may also compare numbers and return boolean truth values (true or false) depending on whether one is equal to, greater than or less than the other ("is 64 greater than 65?"). Logic operations involve Boolean logic: AND, OR, XOR, and NOT. These can be useful for creating complicated conditional statements and processing boolean logic.

Superscalar computers may contain multiple ALUs, allowing them to process several instructions simultaneously.[60] Graphics processors and computers with SIMD and MIMD features often contain ALUs that can perform arithmetic on vectors and matrices.

Memory

Main article: Computer data storage



Magnetic core memory was the computer memory of choice throughout the 1960s, until it was replaced by semiconductor memory.

A computer's memory can be viewed as a list of cells into which numbers can be placed or read. Each cell has a numbered "address" and can store a single number. The computer can be instructed to "put the number 123 into the cell numbered 1357" or to "add the number that is in cell 1357 to the number that is in cell 2468 and put the answer into cell 1595." The information stored in memory may represent practically anything. Letters, numbers, even computer instructions can be placed into memory with equal ease. Since the CPU does not differentiate between different types of information, it is the software's responsibility to give significance to what the memory sees as nothing but a series of numbers.

In almost all modern computers, each memory cell is set up to store binary numbers in groups of eight bits (called a byte). Each byte is able to represent 256 different numbers ($2^8 = 256$); either from 0 to 255 or −128 to +127. To store larger numbers, several consecutive bytes may be used (typically, two, four or eight). When negative numbers are required, they are usually stored in two's complement notation. Other arrangements are possible, but are usually not seen outside of specialized applications or historical contexts. A computer can store any kind of information in memory if it can be represented numerically. Modern computers have billions or even trillions of bytes of memory.

The CPU contains a special set of memory cells called registers that can be read and written to much more rapidly than the main memory area. There are typically between two and one hundred registers depending on the type of CPU. Registers are used for the most frequently needed data items to avoid having to access main memory every time data is needed. As data is constantly being worked on, reducing the need to access main memory (which is often slow compared to the ALU and control units) greatly increases the computer's speed.

Computer main memory comes in two principal varieties:

- random-access memory or RAM
- read-only memory or ROM

RAM can be read and written to anytime the CPU commands it, but ROM is preloaded with data and software that never changes, therefore the CPU can only read from it. ROM is typically used to store the computer's initial start-up instructions. In general, the contents of RAM are erased when the power to the computer is turned off, but ROM retains its data indefinitely. In a PC, the ROM contains a specialized program called the BIOS that orchestrates loading the computer's operating system from the hard disk drive into RAM whenever the computer is turned on or reset. In embedded computers, which frequently do not have disk drives, all of the required software may be stored in ROM. Software stored in ROM is often called firmware, because it is notionally more like hardware than software. Flash memory blurs the distinction between ROM and RAM, as it retains its data when turned off but is also rewritable. It is typically much slower than conventional ROM and RAM however, so its use is restricted to applications where high speed is unnecessary.[61]

In more sophisticated computers there may be one or more RAM cache memories, which are slower than registers but faster than main memory. Generally computers with this sort of cache are designed to move frequently needed data into the cache automatically, often without the need for any intervention on the programmer's part.

Input/output (I/O)

Main article: Input/output

Hard disk drives are common storage devices used with computers.

I/O is the means by which a computer exchanges information with the outside world.[62] Devices that provide input or output to the computer are called peripherals.[63] On a typical personal computer, peripherals include input devices like the keyboard and mouse, and output devices such as the display and printer. Hard disk drives, floppy disk drives and optical disc drives serve as both input and output devices. Computer networking is another form of I/O. I/O devices are often complex computers in their own right, with their own CPU and memory. A graphics processing unit might contain fifty or more tiny computers that perform the calculations necessary to display 3D graphics.[*citation needed*] Modern desktop computers contain many smaller computers that assist the main CPU in performing I/O. A 2016-era flat screen display contains its own computer circuitry.

Multitasking

Main article: Computer multitasking

While a computer may be viewed as running one gigantic program stored in its main memory, in some systems it is necessary to give the appearance of running several programs simultaneously. This is achieved by multitasking i.e. having the computer switch rapidly between running each program in turn.[64] One means by which this is done is with a special signal called an interrupt, which can periodically cause the computer to stop executing instructions where it was and do something else instead. By remembering where it was executing prior to the interrupt, the computer can return to that task later. If several programs are running "at the same time". then the interrupt generator might be causing several hundred interrupts per second, causing a program switch each time. Since modern computers typically execute instructions several orders of magnitude faster than human perception, it may appear that many programs are running at the same time even though only one is ever executing in any given instant. This method of multitasking is sometimes termed "time-sharing" since each program is allocated a "slice" of time in turn.[65]

Before the era of inexpensive computers, the principal use for multitasking was to allow many people to share the same computer. Seemingly, multitasking would cause a computer that is switching between several programs to run more slowly, in direct proportion to the number of programs it is running, but most programs spend much of their time waiting for slow input/output devices to complete their tasks. If a program is waiting for the user to click on the mouse or press a key on the keyboard, then it will not take a "time slice" until the event it is waiting for has occurred. This frees up time for other programs to execute so that many programs may be run simultaneously without unacceptable speed loss.

Multiprocessing

Main article: Multiprocessing

Cray designed many supercomputers that used multiprocessing heavily.

Some computers are designed to distribute their work across several CPUs in a multiprocessing configuration, a technique once employed only in large and powerful machines such as supercomputers, mainframe computers and servers. Multiprocessor and multi-core (multiple CPUs on a single integrated circuit) personal and laptop computers are now widely available, and are being increasingly used in lower-end markets as a result.

Supercomputers in particular often have highly unique architectures that differ significantly from the basic stored-program architecture and from general purpose computers.[66] They often feature thousands of CPUs, customized high-speed interconnects, and specialized computing hardware. Such designs tend to be useful only for specialized tasks due to the large scale of program organization required to successfully utilize most of the available resources at once. Supercomputers usually see usage in large-scale simulation, graphics rendering, and cryptography applications, as well as with other so-called "embarrassingly parallel" tasks.

Software

Main article: Computer software

Software refers to parts of the computer which do not have a material form, such as programs, data, protocols, etc. Software is that part of a computer system that consists of encoded information or computer instructions, in contrast to the physical hardware from which the system is built. Computer software includes computer programs, libraries and related non-executable data, such as online documentation or digital media. It is often divided into system software and application software]] Computer hardware and software require each other and neither can be realistically used on its own. When software is stored in hardware that cannot easily be modified, such as with BIOS ROM in an IBM PC compatible computer, it is sometimes called "firmware".

Operating system /System Software	Unix and BSD	UNIX System V, IBM AIX, HP-UX, Solaris (SunOS), IRIX, List of BSD operating systems
	GNU/Linux	List of Linux distributions, Comparison of Linux distributions
	Microsoft Windows	Windows 95, Windows 98, Windows NT, Windows 2000, Windows ME, Windows XP, Windows Vista, Windows 7, Windows 8, Windows 8.1, Windows 10
	DOS	86-DOS (QDOS), IBM PC DOS, MS-DOS, DR-DOS, FreeDOS
	Macintosh	Classic Mac OS, macOS (previously OS X and Mac OS X)

	operating systems	
	Embedded and real-time	List of embedded operating systems
	Experimental	Amoeba, Oberon/Bluebottle, Plan 9 from Bell Labs
Library	Multimedia	DirectX, OpenGL, OpenAL, Vulkan (API)
	Programming library	C standard library, Standard Template Library
Data	Protocol	TCP/IP, Kermit, FTP, HTTP, SMTP
	File format	HTML, XML, JPEG, MPEG, PNG
User interface	Graphical user interface (WIMP)	Microsoft Windows, GNOME, KDE, QNX Photon, CDE, GEM, Aqua
	Text-based user interface	Command-line interface, Text user interface
Application Software	Office suite	Word processing, Desktop publishing, Presentation program, Database management system, Scheduling & Time management, Spreadsheet, Accounting software
	Internet Access	Browser, Email client, Web server, Mail transfer agent, Instant messaging
	Design and manufacturing	Computer-aided design, Computer-aided manufacturing, Plant management, Robotic manufacturing, Supply chain management
	Graphics	Raster graphics editor, Vector graphics editor, 3D modeler, Animation editor, 3D computer graphics, Video editing, Image processing
	Audio	Digital audio editor, Audio playback, Mixing, Audio synthesis, Computer music
	Software engineering	Compiler, Assembler, Interpreter, Debugger, Text editor, Integrated development environment, Software performance analysis, Revision control, Software configuration management
	Educational	Edutainment, Educational game, Serious game, Flight simulator
	Games	Strategy, Arcade, Puzzle, Simulation, First-person shooter, Platform, Massively multiplayer, Interactive fiction
	Misc	Artificial intelligence, Antivirus software, Malware scanner, Installer/Package management systems, File manager

Languages

There are thousands of different programming languages—some intended to be general purpose, others useful only for highly specialized applications.

Programming languages	
Lists of programming languages	Timeline of programming languages, List of programming languages by category, Generational list of programming languages, List of programming languages, Non-English-based programming languages
Commonly used assembly languages	ARM, MIPS, x86
Commonly used high-	Ada, BASIC, C, C++, C#, COBOL, Fortran, PL/1, REXX, Java, Lisp, Pascal,

level programming languages	Object Pascal
Commonly used scripting languages	Bourne script, JavaScript, Python, Ruby, PHP, Perl

Application Software

Programs

The defining feature of modern computers which distinguishes them from all other machines is that they can be programmed. That is to say that some type of instructions (the program) can be given to the computer, and it will process them. Modern computers based on the von Neumann architecture often have machine code in the form of an imperative programming language. In practical terms, a computer program may be just a few instructions or extend to many millions of instructions, as do the programs for word processors and web browsers for example. A typical modern computer can execute billions of instructions per second (gigaflops) and rarely makes a mistake over many years of operation. Large computer programs consisting of several million instructions may take teams of programmers years to write, and due to the complexity of the task almost certainly contain errors.

Stored program architecture

Main articles: Computer program and Computer programming

Replica of the Manchester Baby, the world's first electronic stored-program computer, at the Museum of Science and Industry in Manchester, England

This section applies to most common RAM machine–based computers.

In most cases, computer instructions are simple: add one number to another, move some data from one location to another, send a message to some external device, etc. These instructions are read from the computer's memory and are generally carried out (executed) in the order they were given. However, there are usually specialized instructions to tell the computer to jump ahead or backwards to some other place in the program and to carry on executing from there. These are called "jump" instructions (or branches). Furthermore, jump instructions may be made to happen conditionally so that different sequences of instructions may be used depending on the result of some previous calculation or some external event. Many computers directly support subroutines by providing a type of jump that "remembers" the location it jumped from and another instruction to return to the instruction following that jump instruction.

Program execution might be likened to reading a book. While a person will normally read each word and line in sequence, they may at times jump back to an earlier place in the text or skip sections that are not of interest. Similarly, a computer may sometimes go back and repeat the instructions in some section of the program over and over again until some internal condition is met. This is called the flow of control within the program and it is what allows the computer to perform tasks repeatedly without human intervention.

Comparatively, a person using a pocket calculator can perform a basic arithmetic operation such as adding two numbers with just a few button presses. But to add together all of the numbers from 1 to 1,000 would take thousands of button presses and a lot of time, with a near certainty of making a mistake. On the other hand, a computer may be programmed to do this with just a few simple instructions. The following example is written in the MIPS assembly language:

```
begin:
addi $8, $0, 0          # initialize sum to 0
addi $9, $0, 1          # set first number to add = 1
loop:
slti $10, $9, 1000      # check if the number is less than 1000
beq $10, $0, finish     # if odd number is greater than n then exit
add $8, $8, $9          # update sum
addi $9, $9, 1          # get next number
j loop                  # repeat the summing process
finish:
add $2, $8, $0          # put sum in output register
```

Once told to run this program, the computer will perform the repetitive addition task without further human intervention. It will almost never make a mistake and a modern PC can complete the task in a fraction of a second.

Machine code

In most computers, individual instructions are stored as machine code with each instruction being given a unique number (its operation code or opcode for short). The command to add two numbers together would have one opcode; the command to multiply them would have a different opcode, and so on. The simplest computers are able to perform any of a handful of different instructions; the more complex computers have several hundred to choose from, each with a unique numerical code. Since the computer's memory is able to store numbers, it can also store the instruction codes. This leads to the important fact that entire programs (which are just lists of these instructions) can be represented as lists of numbers and can themselves be manipulated inside the computer in the same way as numeric data. The fundamental concept of storing programs in the computer's memory alongside the data they operate on is the crux of the von Neumann, or stored program[*citation needed*], architecture. In some cases, a computer might store some or all of its program in memory that is kept separate from the data it operates on. This is called the Harvard architecture after the Harvard Mark I computer. Modern von Neumann computers display some traits of the Harvard architecture in their designs, such as in CPU caches.

While it is possible to write computer programs as long lists of numbers (machine language) and while this technique was used with many early computers,[67] it is extremely tedious and potentially error-prone to do so in practice, especially for complicated programs. Instead, each basic instruction can be given a short name that is indicative of its function and easy to remember – a mnemonic such as ADD, SUB, MULT or JUMP. These mnemonics are collectively known as a computer's assembly language. Converting programs written in assembly language into something the computer can actually understand (machine language) is usually done by a computer program called an assembler.

A 1970s punched card containing one line from a Fortran program. The card reads: "Z(1) = Y + W(1)" and is labeled "PROJ039" for identification purposes.

Programming language

Main article: Programming language

Programming languages provide various ways of specifying programs for computers to run. Unlike natural languages, programming languages are designed to permit no ambiguity and to be concise. They are purely written languages and are often difficult to read aloud. They are generally either translated into machine code by a compiler or an assembler before being run, or translated directly at run time by an interpreter. Sometimes programs are executed by a hybrid method of the two techniques.

Low-level languages

Main article: Low-level programming language

Machine languages and the assembly languages that represent them (collectively termed *low-level programming languages*) tend to be unique to a particular type of computer. For instance, an ARM architecture computer (such as may be found in a smartphone or a hand-held videogame) cannot understand the machine language of an x86 CPU that might be in a PC.[68]

High-level languages/third generation language

Main article: High-level programming language

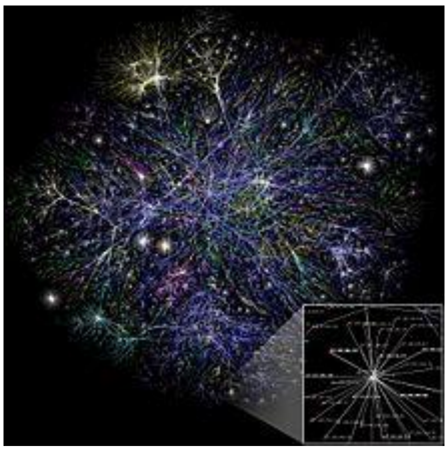
Although considerably easier than in machine language, writing long programs in assembly language is often difficult and is also error prone. Therefore, most practical programs are written in more abstract high-level programming languages that are able to express the needs of the programmer more conveniently (and thereby help reduce programmer error). High level languages are usually "compiled" into machine language (or sometimes into assembly language and then into machine language) using another computer program called a compiler.[69] High level languages are less related to the workings of the target computer than assembly language, and more related to the language and structure of the problem(s) to be solved by the final program. It is therefore often possible to use different compilers to translate the same high level language program into the machine language of many different types of computer. This is part of the means by which software like video games may be made available for different computer architectures such as personal computers and various video game consoles.

Fourth-generation languages

Main article: Fourth-generation programming language

Fourth-generation languages (4GL) are less procedural than 3G languages. The benefit of 4GL is that they provide ways to obtain information without requiring the direct help of a programmer.

Program design



Program design of small programs is relatively simple and involves the analysis of the problem, collection of inputs, using the programming constructs within languages, devising or using established procedures and algorithms, providing data for output devices and solutions to the problem as applicable. As problems become larger and more complex, features such as subprograms, modules, formal documentation, and new paradigms such as object-oriented programming are encountered. Large programs involving thousands of line of code and more require formal software methodologies. The task of developing large software systems presents a significant intellectual challenge. Producing software with an acceptably high reliability within a predictable schedule and budget has historically been difficult; the academic and professional discipline of software engineering concentrates specifically on this challenge.

Bugs

Main article: Software bug

The actual first computer bug, a moth found trapped on a relay of the Harvard Mark II computer

Errors in computer programs are called "bugs". They may be benign and not affect the usefulness of the program, or have only subtle effects. But in some cases, they may cause the program or the entire system to "hang", becoming unresponsive to input such as mouse clicks or keystrokes, to completely fail, or to crash. Otherwise benign bugs may sometimes be harnessed for malicious intent by an unscrupulous user writing an exploit, code designed to take advantage of a bug and disrupt a computer's proper execution. Bugs are usually not the fault of the computer. Since computers merely execute the instructions they are given, bugs are nearly always the result of programmer error or an oversight made in the program's design.[70] Admiral Grace Hopper, an American computer scientist and developer of the first compiler, is credited for having first used the term "bugs" in computing after a dead moth was found shorting a relay in the Harvard Mark II computer in September 1947.[71]

Firmware

Firmware is the technology which has the combination of both hardware and software such as BIOS chip inside a computer. This chip (hardware) is located on the motherboard and has the BIOS set up (software) stored in it.

Networking and the Internet

Main articles: Computer networking and Internet

Visualization of a portion of the routes on the Internet

Computers have been used to coordinate information between multiple locations since the 1950s. The U.S. military's SAGE system was the first large-scale example of such a system, which led to a number of special-purpose commercial systems such as Sabre.[72] In the 1970s, computer engineers at research institutions throughout the United States began to link their computers together using telecommunications technology. The effort was funded by ARPA (now DARPA), and the computer network that resulted was called the ARPANET.[73] The technologies that made the Arpanet possible spread and evolved.

In time, the network spread beyond academic and military institutions and became known as the Internet. The emergence of networking involved a redefinition of the nature and boundaries of the computer. Computer operating systems and applications were modified to include the ability to define and access the resources of other computers on the network, such as peripheral devices, stored information, and the like, as extensions of the resources of an individual computer. Initially these facilities were available primarily to people working in high-tech environments, but in the 1990s the spread of applications like e-mail and the World Wide Web, combined with the development of cheap, fast networking technologies like Ethernet and ADSL saw computer networking become almost ubiquitous. In fact, the number of computers that are networked is growing phenomenally. A very large proportion of personal computers regularly connect to the Internet to communicate and receive information. "Wireless" networking, often utilizing mobile phone networks, has meant networking is becoming increasingly ubiquitous even in mobile computing environments.

Unconventional computers

Main article: Human computer

See also: Harvard Computers

A computer does not need to be electronic, nor even have a processor, nor RAM, nor even a hard disk. While popular usage of the word "computer" is synonymous with a personal electronic computer, the modern[74] definition of a computer is literally: "*A device that computes*, especially a programmable [usually] electronic machine that performs high-speed mathematical or logical operations or that assembles, stores, correlates, or otherwise processes information."[75] Any device which *processes information* qualifies as a computer, especially if the processing is purposeful.[*citation needed*]

Unconventional computing

Further information: Unconventional computing

Historically, computers evolved from mechanical computers and eventually from vacuum tubes to transistors. However, conceptually computational systems as flexible as a personal computer can be built out of almost anything. For example, a computer can be made out of billiard balls (billiard ball computer); an often quoted example.[*citation needed*] More realistically, modern computers are made out of transistors made of photolithographed semiconductors.

Future

There is active research to make computers out of many promising new types of technology, such as optical computers, DNA computers, neural computers, and quantum computers. Most computers are universal, and are able to calculate any computable function, and are limited only by their memory capacity and operating speed. However different designs of computers can give very different performance for particular problems; for example quantum computers can potentially break some modern encryption algorithms (by quantum factoring) very quickly.

Computer architecture paradigms

There are many types of computer architectures:

- Quantum computer vs. Chemical computer
- Scalar processor vs. Vector processor
- Non-Uniform Memory Access (NUMA) computers
- Register machine vs. Stack machine
- Harvard architecture vs. von Neumann architecture
- Cellular architecture

Of all these abstract machines, a quantum computer holds the most promise for revolutionizing computing.[76] Logic gates are a common abstraction which can apply to most of the above digital or analog paradigms. The ability to store and execute lists of instructions called programs makes computers extremely versatile, distinguishing them from calculators. The Church–Turing thesis is a mathematical statement of this versatility: any computer with a minimum capability (being Turing-complete) is, in principle, capable of performing the same tasks that any other computer can perform. Therefore, any type of computer (netbook, supercomputer, cellular automaton, etc.) is able to perform the same computational tasks, given enough time and storage capacity.

Artificial intelligence

A computer will solve problems in exactly the way it is programmed to, without regard to efficiency, alternative solutions, possible shortcuts, or possible errors in the code. Computer programs that learn and adapt are part of the emerging field of artificial intelligence and machine learning. Artificial intelligence based products generally fall into two major categories: rule based systems and pattern recognition systems. Rule based systems attempt to represent the rules used by human experts and tend to be expensive to develop. Pattern based systems use data about a problem to generate conclusions. Examples of pattern based systems include voice recognition, font recognition, translation and the emerging field of on-line marketing.

Professions and organizations

As the use of computers has spread throughout society, there are an increasing number of careers involving computers.

Computer-related professions	
Hardware-related	Electrical engineering, Electronic engineering, Computer engineering, Telecommunications engineering, Optical engineering, Nanoengineering
Software-related	Computer science, Computer engineering, Desktop publishing, Human–computer interaction, Information technology, Information systems, Computational science, Software engineering, Video game industry, Web design

The need for computers to work well together and to be able to exchange information has spawned the need for many standards organizations, clubs and societies of both a formal and informal nature.

Organizations	
Standards groups	ANSI, IEC, IEEE, IETF, ISO, W3C
Professional societies	ACM, AIS, IET, IFIP, BCS
Free/open source software groups	Free Software Foundation, Mozilla Foundation, Apache Software Foundation

Programming language

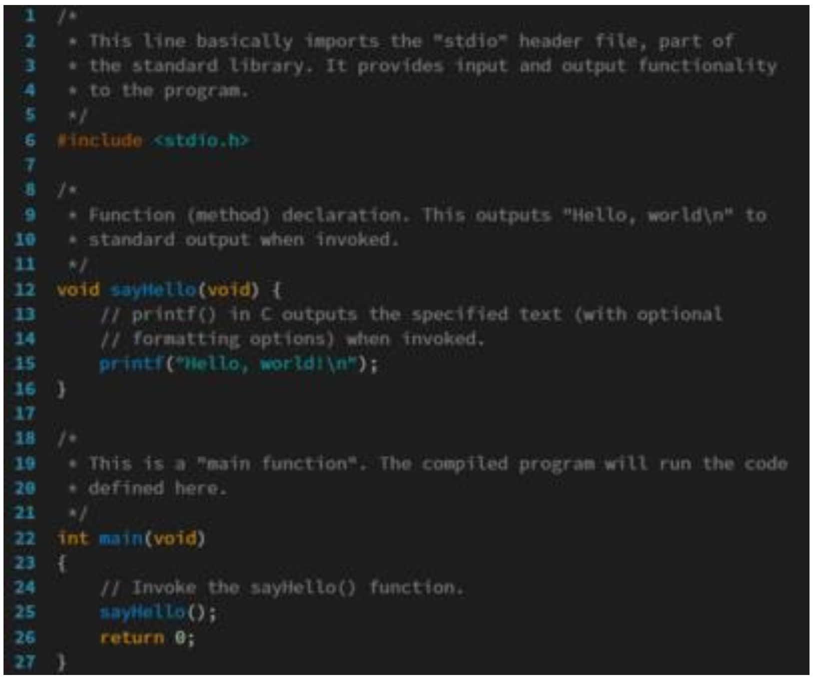
From Wikipedia, the free encyclopedia

Jump to navigationJump to search

The source code for a simple computer program written in the C programming language. When compiled and run, it would give the output "Hello, world!".

A **programming language** is a formal language, which comprises a set of instructions used to produce various kinds of output. Programming languages are used to create programs that implement specific algorithms.

Most programming languages consist of instructions for computers, although there are programmable machines that use a limited set of specific instructions, rather than the general programming languages of modern computers. Early ones preceded the invention of the digital computer, the first probably being the automatic flute player described in the 9th century by the brothers Musa in Baghdad, during the Islamic Golden Age.[1] From the early 1800s, programs were used to direct the behavior of machines such as Jacquard looms, music boxes and player pianos.[2] However, their programs (such as a player piano's scrolls) could not produce different behavior in response to some input or condition.



```c
/*
 * This line basically imports the "stdio" header file, part of
 * the standard library. It provides input and output functionality
 * to the program.
 */
#include <stdio.h>

/*
 * Function (method) declaration. This outputs "Hello, world\n" to
 * standard output when invoked.
 */
void sayHello(void) {
    // printf() in C outputs the specified text (with optional
    // formatting options) when invoked.
    printf("Hello, world!\n");
}

/*
 * This is a "main function". The compiled program will run the code
 * defined here.
 */
int main(void)
{
    // Invoke the sayHello() function.
    sayHello();
    return 0;
}
```

Thousands of different programming languages have been created, mainly in the computer field, and many more still are being created every year. Many programming languages require computation to be specified in an imperative form (i.e., as a sequence of operations to perform) while other languages use other forms of program specification such as the declarative form (i.e. the desired result is specified, not how to achieve it).

The description of a programming language is usually split into the two components of syntax (form) and semantics (meaning). Some languages are defined by a specification document (for example, the C programming language is specified by an ISO Standard) while other languages (such as Perl) have a dominant implementation that is treated as a reference. Some languages have both, with the basic language defined by a standard and extensions taken from the dominant implementation being common.

Definitions:

A programming language is a notation for writing programs, which are specifications of a computation or algorithm.[3] Some, but not all, authors restrict the term "programming language" to those languages that can express *all* possible algorithms.[3][4] Traits often considered important for what constitutes a programming language include:

Function and target

A *computer programming language* is a language used to write computer programs, which involves a computer performing some kind of computation[5] or algorithm and possibly control external devices such as printers, disk drives, robots,[6] and so on. For example, PostScript programs are frequently created by another program to control a computer printer or display. More generally, a programming language may describe computation on some, possibly abstract, machine. It is generally accepted that a complete specification for a programming language includes a description, possibly idealized, of a machine or processor for that language.[7] In most practical contexts, a programming language involves a computer; consequently, programming languages are usually defined and studied this way.[8] Programming languages differ from natural languages in that natural languages are only used for interaction between people, while programming languages also allow humans to communicate instructions to machines.

Abstractions

Programming languages usually contain abstractions for defining and manipulating data structures or controlling the flow of execution. The practical necessity that a programming language support adequate abstractions is expressed by the abstraction principle;[9] this principle is sometimes formulated as a recommendation to the programmer to make proper use of such abstractions.[10]

Expressive power

The theory of computation classifies languages by the computations they are capable of expressing. All Turing complete languages can implement the same set of algorithms. ANSI/ISO SQL-92 and Charity are examples of languages that are not Turing complete, yet often called programming languages.[11][12]

Markup languages like XML, HTML, or troff, which define structured data, are not usually considered programming languages.[13][14][15] Programming languages may, however, share the syntax with markup languages if a computational semantics is defined. XSLT, for example, is a Turing complete language entirely using XML syntax.[16][17][18] Moreover, LaTeX, which is mostly used for structuring documents, also contains a Turing complete subset.[19][20]

The term *computer language* is sometimes used interchangeably with programming language.[21] However, the usage of both terms varies among authors, including the exact scope of each. One usage describes programming languages as a subset of computer languages.[22] In this vein, languages used in computing that have a different goal than expressing computer programs are generically designated computer languages. For instance, markup languages are sometimes referred to as computer languages to emphasize that they are not meant to be used for programming.[23]

Another usage regards programming languages as theoretical constructs for programming abstract machines, and computer languages as the subset thereof that runs on physical computers, which have finite hardware resources.[24] John C. Reynolds emphasizes that formal specification languages are just as much programming languages as are the languages intended for execution. He also argues that textual and even graphical input formats that affect the behavior of a computer are programming languages, despite the fact they are commonly not Turing-complete, and remarks that ignorance of programming language concepts is the reason for many flaws in input formats.[25]

History:

Main article: History of programming languages

Early developments:

Very early computers, such as Colossus, were programmed without the help of a stored program, by modifying their circuitry or setting banks of physical controls.

Slightly later, programs could be written in machine language, where the programmer writes each instruction in a numeric form the hardware can execute directly. For example, the instruction to add the value in two memory location might consist of 3 numbers: a "opcode" that selects the "add" operation, and two memory locations. The programs, in decimal or binary form, were read in from punched cards or magnetic tape or toggled in on switches on the front panel of the computer. Machine languages were later termed *first-generation programming languages* (1GL).

The next step was development of so-called *second-generation programming languages* (2GL) or assembly languages, which were still closely tied to the instruction set architecture of the specific computer. These served to make the program much more human-readable and relieved the programmer of tedious and error-prone address calculations.

The first *high-level programming languages*, or *third-generation programming languages* (3GL), were written in the 1950s. An early high-level programming language to be designed for a computer was

Plankalkül, developed for the German Z3 by Konrad Zuse between 1943 and 1945. However, it was not implemented until 1998 and 2000.[26]

John Mauchly's Short Code, proposed in 1949, was one of the first high-level languages ever developed for an electronic computer.[27] Unlike machine code, Short Code statements represented mathematical expressions in understandable form. However, the program had to be translated into machine code every time it ran, making the process much slower than running the equivalent machine code.

At the University of Manchester, Alick Glennie developed Autocode in the early 1950s. A programming language, it used a compiler to automatically convert the language into machine code. The first code and compiler was developed in 1952 for the Mark 1 computer at the University of Manchester and is considered to be the first compiled high-level programming language.[28][29]

The second autocode was developed for the Mark 1 by R. A. Brooker in 1954 and was called the "Mark 1 Autocode". Brooker also developed an autocode for the Ferranti Mercury in the 1950s in conjunction with the University of Manchester. The version for the EDSAC 2 was devised by D. F. Hartley of University of Cambridge Mathematical Laboratory in 1961. Known as EDSAC 2 Autocode, it was a straight development from Mercury Autocode adapted for local circumstances and was noted for its object code optimisation and source-language diagnostics which were advanced for the time. A contemporary but separate thread of development, Atlas Autocode was developed for the University of Manchester Atlas 1 machine.

In 1954, FORTRAN was invented at IBM by John Backus. It was the first widely used high-level general purpose programming language to have a functional implementation, as opposed to just a design on paper.[30][31] It is still a popular language for high-performance computing[32] and is used for programs that benchmark and rank the world's fastest supercomputers.[33]

Another early programming language was devised by Grace Hopper in the US, called FLOW-MATIC. It was developed for the UNIVAC I at Remington Rand during the period from 1955 until 1959. Hopper found that business data processing customers were uncomfortable with mathematical notation, and in early 1955, she and her team wrote a specification for an English programming language and implemented a prototype.[34] The FLOW-MATIC compiler became publicly available in early 1958 and was substantially complete in 1959.[35] Flow-Matic was a major influence in the design of COBOL, since only it and its direct descendant AIMACO were in actual use at the time.[36]

Refinement:

The increased use of high-level languages introduced a requirement for *low-level programming languages* or *system programming languages*. These languages, to varying degrees, provide facilities between assembly languages and high-level languages and can be used to perform tasks which require direct access to hardware facilities but still provide higher-level control structures and error-checking.

The period from the 1960s to the late 1970s brought the development of the major language paradigms now in use:

- APL introduced *array programming* and influenced functional programming.[37]
- ALGOL refined both *structured procedural programming* and the discipline of language specification; the "Revised Report on the Algorithmic Language ALGOL 60" became a model for how later language specifications were written.
- Lisp, implemented in 1958, was the first dynamically typed *functional programming* language
- In the 1960s, Simula was the first language designed to support *object-oriented programming*; in the mid-1970s, Smalltalk followed with the first "purely" object-oriented language.
- C was developed between 1969 and 1973 as a system programming language for the Unix operating system and remains popular.[38]

- Prolog, designed in 1972, was the first *logic programming* language.
- In 1978, ML built a polymorphic type system on top of Lisp, pioneering *statically typed functional programming* languages.

Each of these languages spawned descendants, and most modern programming languages count at least one of them in their ancestry.

The 1960s and 1970s also saw considerable debate over the merits of *structured programming*, and whether programming languages should be designed to support it.[39] Edsger Dijkstra, in a famous 1968 letter published in the Communications of the ACM, argued that GOTO statements should be eliminated from all "higher level" programming languages.[40]

Consolidation and growth:

A selection of textbooks that teach programming, in languages both popular and obscure. These are only a few of the thousands of programming languages and dialects that have been designed in history.

The 1980s were years of relative consolidation. C++ combined object-oriented and systems programming. The United States government standardized Ada, a systems programming language derived from Pascal and intended for use by defense contractors. In Japan and elsewhere, vast sums were spent investigating so-called "fifth generation" languages that incorporated logic programming constructs.[41] The functional languages community moved to standardize ML and Lisp. Rather than inventing new paradigms, all of these movements elaborated upon the ideas invented in the previous decades.

One important trend in language design for programming large-scale systems during the 1980s was an increased focus on the use of *modules* or large-scale organizational units of code. Modula-2, Ada, and ML all developed notable module systems in the 1980s, which were often wedded to generic programming constructs.[42]

The rapid growth of the Internet in the mid-1990s created opportunities for new languages. Perl, originally a Unix scripting tool first released in 1987, became common in dynamic websites. Java came to be used for server-side programming, and bytecode virtual machines became popular again in commercial settings with their promise of "Write once, run anywhere" (UCSD Pascal had been popular for a time in the early 1980s). These developments were not fundamentally novel, rather they were refinements of many existing languages and paradigms (although their syntax was often based on the C family of programming languages).

Programming language evolution continues, in both industry and research. Current directions include security and reliability verification, new kinds of modularity (mixins, delegates, aspects), and database integration such as Microsoft's LINQ.

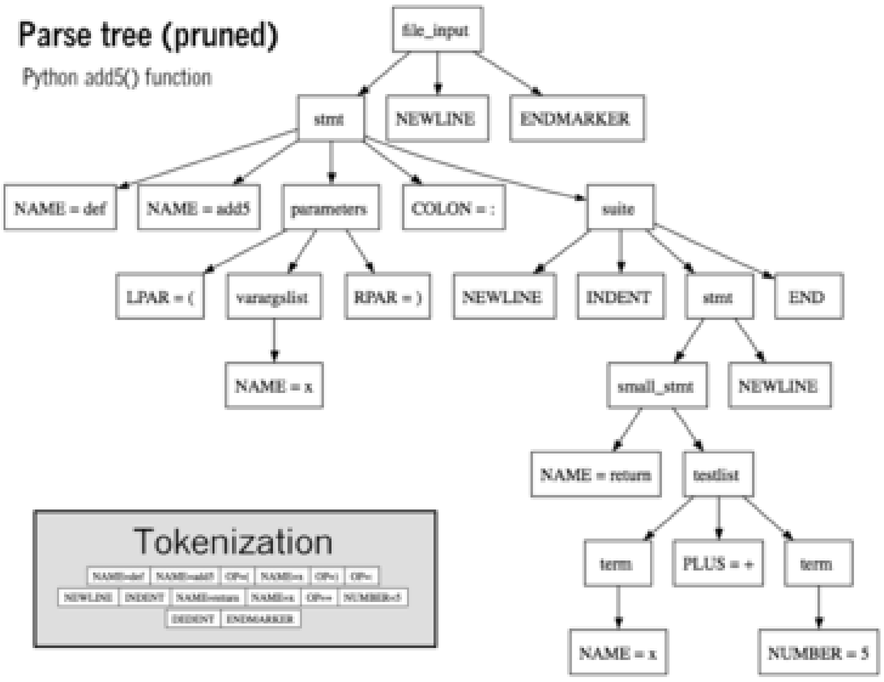
Fourth-generation programming languages (4GL) are computer programming languages which aim to provide a higher level of abstraction of the internal computer hardware details than 3GLs. *Fifth generation programming languages* (5GL) are programming languages based on solving problems using constraints given to the program, rather than using an algorithm written by a programmer.

Elements:

All programming languages have some primitive building blocks for the description of data and the processes or transformations applied to them (like the addition of two numbers or the selection of an item from a collection). These primitives are defined by syntactic and semantic rules which describe their structure and meaning respectively.

Syntax:

Main article: Syntax (programming languages)



Parse tree of Python code with inset tokenization

```
def add5(x):
    return x+5

def dotwrite(ast):
    nodename = getNodename()
    label=symbol.sym_name.get(int(ast[0]),ast[0])
    print '    %s [label="%s' % (nodename, label),
    if isinstance(ast[1], str):
        if ast[1].strip():
            print '= %s"];' % ast[1]
        else:
            print '"]'
    else:
        print '"];'
        children = []
        for  in n, childenumerate(ast[1:]):
            children.append(dotwrite(child))
        print ,'    %s -> {' % nodename
        for  in :namechildren
            print '%s' % name,
```

Syntax highlighting is often used to aid programmers in recognizing elements of source code. The language above is Python.

A programming language's surface form is known as its syntax. Most programming languages are purely textual; they use sequences of text including words, numbers, and punctuation, much like written natural languages. On the other hand, there are some programming languages which are more graphical in nature, using visual relationships between symbols to specify a program.

The syntax of a language describes the possible combinations of symbols that form a syntactically correct program. The meaning given to a combination of symbols is handled by semantics (either formal or hard-coded in a reference implementation). Since most languages are textual, this article discusses textual syntax.

Programming language syntax is usually defined using a combination of regular expressions (for lexical structure) and Backus–Naur form (for grammatical structure). Below is a simple grammar, based on Lisp:

```
expression ::= atom | list
atom       ::= number | symbol
number     ::= [+-]?['0'-'9']+
symbol     ::= ['A'-'Z''a'-'z'].*
list       ::= '(' expression* ')'
```

This grammar specifies the following:

- an *expression* is either an *atom* or a *list*;
- an *atom* is either a *number* or a *symbol*;
- a *number* is an unbroken sequence of one or more decimal digits, optionally preceded by a plus or minus sign;
- a *symbol* is a letter followed by zero or more of any characters (excluding whitespace); and
- a *list* is a matched pair of parentheses, with zero or more *expressions* inside it.

The following are examples of well-formed token sequences in this grammar: `12345`, `()` and `(a b c232 (1))`.

Not all syntactically correct programs are semantically correct. Many syntactically correct programs are nonetheless ill-formed, per the language's rules; and may (depending on the language specification and the soundness of the implementation) result in an error on translation or execution. In some cases, such programs may exhibit undefined behavior. Even when a program is well-defined within a language, it may still have a meaning that is not intended by the person who wrote it.

Using natural language as an example, it may not be possible to assign a meaning to a grammatically correct sentence or the sentence may be false:

- "Colorless green ideas sleep furiously." is grammatically well-formed but has no generally accepted meaning.
- "John is a married bachelor." is grammatically well-formed but expresses a meaning that cannot be true.

The following C language fragment is syntactically correct, but performs operations that are not semantically defined (the operation `*p >> 4` has no meaning for a value having a complex type and `p->im` is not defined because the value of `p` is the null pointer):

```
complex *p = NULL;
complex abs_p = sqrt(*p >> 4 + p->im);
```

If the type declaration on the first line were omitted, the program would trigger an error on compilation, as the variable "p" would not be defined. But the program would still be syntactically correct since type declarations provide only semantic information.

The grammar needed to specify a programming language can be classified by its position in the Chomsky hierarchy. The syntax of most programming languages can be specified using a Type-2 grammar, i.e., they are context-free grammars.[43] Some languages, including Perl and Lisp, contain constructs that allow execution during the parsing phase. Languages that have constructs that allow the programmer to alter the behavior of the parser make syntax analysis an undecidable problem, and generally blur the distinction between parsing and execution.[44] In contrast to Lisp's macro system and Perl's `BEGIN` blocks, which may contain general computations, C macros are merely string replacements and do not require code execution.[45]

Semantics:

The term *semantics* refers to the meaning of languages, as opposed to their form (syntax).

Static semantics:

The static semantics defines restrictions on the structure of valid texts that are hard or impossible to express in standard syntactic formalisms.[3] For compiled languages, static semantics essentially include those semantic rules that can be checked at compile time. Examples include checking that every identifier is declared before it is used (in languages that require such declarations) or that the labels on the arms of a case statement are distinct.[46] Many important restrictions of this type, like checking that identifiers are used in the appropriate context (e.g. not adding an integer to a function name), or that subroutine calls have the appropriate number and type of arguments, can be enforced by defining them as rules in a logic called a type system. Other forms of static analyses like data flow analysis may also be part of static semantics. Newer programming languages like Java and C# have definite assignment analysis, a form of data flow analysis, as part of their static semantics.

Dynamic semantics:

Main article: Semantics of programming languages

Once data has been specified, the machine must be instructed to perform operations on the data. For example, the semantics may define the strategy by which expressions are evaluated to values, or the manner in which control structures conditionally execute statements. The *dynamic semantics* (also known as *execution semantics*) of a language defines how and when the various constructs of a language should produce a program behavior. There are many ways of defining execution semantics. Natural language is often used to specify the execution semantics of languages commonly used in practice. A significant amount of academic research went into formal semantics of programming languages, which allow execution semantics to be specified in a formal manner. Results from this field of research have seen limited application to programming language design and implementation outside academia.

Type system

Main articles: Data type, Type system, and Type safety

A type system defines how a programming language classifies values and expressions into *types*, how it can manipulate those types and how they interact. The goal of a type system is to verify and usually enforce a certain level of correctness in programs written in that language by detecting certain incorrect operations. Any decidable type system involves a trade-off: while it rejects many incorrect programs, it can also prohibit some correct, albeit unusual programs. In order to bypass this downside, a number of languages have *type loopholes*, usually unchecked casts that may be used by the programmer to explicitly allow a normally disallowed operation between different types. In most typed languages, the type system is used only to type check programs, but a number of languages, usually functional ones, infer types, relieving the programmer from the need to write type annotations. The formal design and study of type systems is known as *type theory*.

Typed versus untyped languages:

A language is *typed* if the specification of every operation defines types of data to which the operation is applicable, with the implication that it is not applicable to other types.[47] For example, the data represented by `"this text between the quotes"` is a string, and in many programming languages dividing a number by a string has no meaning and will be rejected by the compilers. The invalid operation may be detected when the program is compiled ("static" type checking) and will be rejected by the compiler with a compilation error message, or it may be detected when the program is run ("dynamic" type checking), resulting in a run-time exception. Many languages allow a function called an exception handler to be written to handle this exception and, for example, always return "-1" as the result.

A special case of typed languages are the *single-type* languages. These are often scripting or markup languages, such as REXX or SGML, and have only one data type[*dubious – discuss*]—most commonly character strings which are used for both symbolic and numeric data.

In contrast, an *untyped language*, such as most assembly languages, allows any operation to be performed on any data, which are generally considered to be sequences of bits of various lengths.[47] High-level languages which are untyped include BCPL, Tcl, and some varieties of Forth.

In practice, while few languages are considered typed from the point of view of type theory (verifying or rejecting *all* operations), most modern languages offer a degree of typing.[47] Many production languages provide means to bypass or subvert the type system, trading type-safety for finer control over the program's execution (see casting).

Static versus dynamic typing:

In *static typing*, all expressions have their types determined prior to when the program is executed, typically at compile-time. For example, 1 and (2+2) are integer expressions; they cannot be passed to a function that expects a string, or stored in a variable that is defined to hold dates.[47]

Statically typed languages can be either *manifestly typed* or *type-inferred*. In the first case, the programmer must explicitly write types at certain textual positions (for example, at variable declarations). In the second case, the compiler *infers* the types of expressions and declarations based on context. Most mainstream statically typed languages, such as C++, C# and Java, are manifestly typed. Complete type inference has traditionally been associated with less mainstream languages, such as Haskell and ML. However, many manifestly typed languages support partial type inference; for example, C++, Java and C# all infer types in certain limited cases.[48] Additionally, some programming languages allow for some types to be automatically converted to other types; for example, an int can be used where the program expects a float.

Dynamic typing, also called *latent typing*, determines the type-safety of operations at run time; in other words, types are associated with *run-time values* rather than *textual expressions*.[47] As with type-inferred languages, dynamically typed languages do not require the programmer to write explicit type annotations on expressions. Among other things, this may permit a single variable to refer to values of different types at different points in the program execution. However, type errors cannot be automatically detected until a piece of code is actually executed, potentially making debugging more difficult. Lisp, Smalltalk, Perl, Python, JavaScript, and Ruby are all examples of dynamically typed languages.

Weak and strong typing:

Weak typing allows a value of one type to be treated as another, for example treating a string as a number.[47] This can occasionally be useful, but it can also allow some kinds of program faults to go undetected at compile time and even at run time.

Strong typing prevents the above. An attempt to perform an operation on the wrong type of value raises an error.[47] Strongly typed languages are often termed *type-safe* or *safe*.

An alternative definition for "weakly typed" refers to languages, such as Perl and JavaScript, which permit a large number of implicit type conversions. In JavaScript, for example, the expression `2 * x` implicitly converts `x` to a number, and this conversion succeeds even if `x` is `null`, `undefined`, an `Array`, or a string of letters. Such implicit conversions are often useful, but they can mask programming errors. *Strong* and *static* are now generally considered orthogonal concepts, but usage in the literature differs. Some use the term *strongly typed* to mean *strongly, statically typed*, or, even more confusingly, to mean simply *statically typed*. Thus C has been called both strongly typed and weakly, statically typed.[49][50]

It may seem odd to some professional programmers that C could be "weakly, statically typed". However, notice that the use of the generic pointer, the **void*** pointer, does allow for casting of pointers to other pointers without needing to do an explicit cast. This is extremely similar to somehow casting an array of bytes to any kind of datatype in C without using an explicit cast, such as `(int)` or `(char)`.

Standard library and run-time system:

Main article: Standard library

Most programming languages have an associated core library (sometimes known as the 'standard library', especially if it is included as part of the published language standard), which is conventionally made available by all implementations of the language. Core libraries typically include definitions for commonly used algorithms, data structures, and mechanisms for input and output.

The line between a language and its core library differs from language to language. In some cases, the language designers may treat the library as a separate entity from the language. However, a language's core library is often treated as part of the language by its users, and some language specifications even require that this library be made available in all implementations. Indeed, some languages are designed so that the meanings of certain syntactic constructs cannot even be described without referring to the core library. For example, in Java, a string literal is defined as an instance of the `java.lang.String` class; similarly, in Smalltalk, an anonymous function expression (a "block") constructs an instance of the library's `BlockContext` class. Conversely, Scheme contains multiple coherent subsets that suffice to construct the rest of the language as library macros, and so the language designers do not even bother to say which portions of the language must be implemented as language constructs, and which must be implemented as parts of a library.

Design and implementation:

Programming languages share properties with natural languages related to their purpose as vehicles for communication, having a syntactic form separate from its semantics, and showing *language families* of related languages branching one from another.[51][52] But as artificial constructs, they also differ in fundamental ways from languages that have evolved through usage. A significant difference is that a programming language can be fully described and studied in its entirety, since it has a precise and finite definition.[53] By contrast, natural languages have changing meanings given by their users in different communities. While constructed languages are also artificial languages designed from the ground up with a specific purpose, they lack the precise and complete semantic definition that a programming language has.

Many programming languages have been designed from scratch, altered to meet new needs, and combined with other languages. Many have eventually fallen into disuse. Although there have been attempts to design one "universal" programming language that serves all purposes, all of them have failed to be generally accepted as filling this role.[54] The need for diverse programming languages arises from the diversity of contexts in which languages are used:

- Programs range from tiny scripts written by individual hobbyists to huge systems written by hundreds of programmers.
- Programmers range in expertise from novices who need simplicity above all else, to experts who may be comfortable with considerable complexity.
- Programs must balance speed, size, and simplicity on systems ranging from microcontrollers to supercomputers.
- Programs may be written once and not change for generations, or they may undergo continual modification.

- Programmers may simply differ in their tastes: they may be accustomed to discussing problems and expressing them in a particular language.

One common trend in the development of programming languages has been to add more ability to solve problems using a higher level of abstraction. The earliest programming languages were tied very closely to the underlying hardware of the computer. As new programming languages have developed, features have been added that let programmers express ideas that are more remote from simple translation into underlying hardware instructions. Because programmers are less tied to the complexity of the computer, their programs can do more computing with less effort from the programmer. This lets them write more functionality per time unit.[55]

Natural language programming has been proposed as a way to eliminate the need for a specialized language for programming. However, this goal remains distant and its benefits are open to debate. Edsger W. Dijkstra took the position that the use of a formal language is essential to prevent the introduction of meaningless constructs, and dismissed natural language programming as "foolish".[56] Alan Perlis was similarly dismissive of the idea.[57] Hybrid approaches have been taken in Structured English and SQL.

A language's designers and users must construct a number of artifacts that govern and enable the practice of programming. The most important of these artifacts are the language *specification* and *implementation*.

Specification:

Main article: Programming language specification

The specification of a programming language is an artifact that the language users and the implementors can use to agree upon whether a piece of source code is a valid program in that language, and if so what its behavior shall be.

A programming language specification can take several forms, including the following:

- An explicit definition of the syntax, static semantics, and execution semantics of the language. While syntax is commonly specified using a formal grammar, semantic definitions may be written in natural language (e.g., as in the C language), or a formal semantics (e.g., as in Standard ML[58] and Scheme[59] specifications).
- A description of the behavior of a translator for the language (e.g., the C++ and Fortran specifications). The syntax and semantics of the language have to be inferred from this description, which may be written in natural or a formal language.
- A *reference or model* implementation, sometimes written in the language being specified (e.g., Prolog or ANSI REXX[60]). The syntax and semantics of the language are explicit in the behavior of the reference implementation.

Implementation:

Main article: Programming language implementation

An *implementation* of a programming language provides a way to write programs in that language and execute them on one or more configurations of hardware and software. There are, broadly, two approaches to programming language implementation: *compilation* and *interpretation*. It is generally possible to implement a language using either technique.

The output of a compiler may be executed by hardware or a program called an interpreter. In some implementations that make use of the interpreter approach there is no distinct boundary between compiling and interpreting. For instance, some implementations of BASIC compile and then execute the source a line at a time.

Programs that are executed directly on the hardware usually run several orders of magnitude faster than those that are interpreted in software.[*citation needed*]

One technique for improving the performance of interpreted programs is just-in-time compilation. Here the virtual machine, just before execution, translates the blocks of bytecode which are going to be used to machine code, for direct execution on the hardware.

Proprietary languages:

Although most of the most commonly used programming languages have fully open specifications and implementations, many programming languages exist only as proprietary programming languages with the implementation available only from a single vendor, which may claim that such a proprietary language is their intellectual property. Proprietary programming languages are commonly domain specific languages or internal scripting languages for a single product; some proprietary languages are used only internally within a vendor, while others are available to external users.

Some programming languages exist on the border between proprietary and open; for example, Oracle Corporation asserts proprietary rights to some aspects of the Java programming language,[61] and Microsoft's C# programming language, which has open implementations of most parts of the system, also has Common Language Runtime (CLR) as a closed environment.[62]

Many proprietary languages are widely used, in spite of their proprietary nature; examples include MATLAB, VBScript, and Wolfram Language. Some languages may make the transition from closed to open; for example, Erlang was originally an Ericsson's internal programming language.[63]

Use:

Thousands of different programming languages have been created, mainly in the computing field.[64] Software is commonly built with 5 programming languages or more.[65]

Programming languages differ from most other forms of human expression in that they require a greater degree of precision and completeness. When using a natural language to communicate with other people, human authors and speakers can be ambiguous and make small errors, and still expect their intent to be understood. However, figuratively speaking, computers "do exactly what they are told to do", and cannot "understand" what code the programmer intended to write. The combination of the language definition, a program, and the program's inputs must fully specify the external behavior that occurs when the program is executed, within the domain of control of that program. On the other hand, ideas about an algorithm can be communicated to humans without the precision required for execution by using pseudocode, which interleaves natural language with code written in a programming language.

A programming language provides a structured mechanism for defining pieces of data, and the operations or transformations that may be carried out automatically on that data. A programmer uses the abstractions present in the language to represent the concepts involved in a computation. These concepts are represented as a collection of the simplest elements available (called primitives).[66] *Programming* is the process by which programmers combine these primitives to compose new programs, or adapt existing ones to new uses or a changing environment.

Programs for a computer might be executed in a batch process without human interaction, or a user might type commands in an interactive session of an interpreter. In this case the "commands" are simply programs, whose execution is chained together. When a language can run its commands through an interpreter (such as a Unix shell or other command-line interface), without compiling, it is called a scripting language.[67]

Measuring language usage

Main article: Measuring programming language popularity

It is difficult to determine which programming languages are most widely used, and what usage means varies by context. One language may occupy the greater number of programmer hours, a different one have more lines of code, and a third may consume the most CPU time. Some languages are very popular for particular kinds of applications. For example, COBOL is still strong in the corporate data center, often on large mainframes;[68][69] Fortran in scientific and engineering applications; Ada in aerospace, transportation, military, real-time and embedded applications; and C in embedded applications and operating systems. Other languages are regularly used to write many different kinds of applications.

Various methods of measuring language popularity, each subject to a different bias over what is measured, have been proposed:

- counting the number of job advertisements that mention the language[70]
- the number of books sold that teach or describe the language[71]
- estimates of the number of existing lines of code written in the language – which may underestimate languages not often found in public searches[72]
- counts of language references (i.e., to the name of the language) found using a web search engine.

Combining and averaging information from various internet sites, stackify.com reported the ten most popular programming languages as (in descending order by overall popularity): Java, C, C++, Python, C#, JavaScript, VB .NET, R, PHP, and MATLAB.[73]

Dialects, flavors and implementations:

A **dialect** of a programming language or a data exchange language is a (relatively small) variation or extension of the language that does not change its intrinsic nature. With languages such as Scheme and Forth, standards may be considered insufficient, inadequate or illegitimate by implementors, so often they will deviate from the standard, making a new dialect. In other cases, a dialect is created for use in a domain-specific language, often a subset. In the Lisp world, most languages that use basic S-expression syntax and Lisp-like semantics are considered Lisp dialects, although they vary wildly, as do, say, Racket and Clojure. As it is common for one language to have several dialects, it can become quite difficult for an inexperienced programmer to find the right documentation. The BASIC programming language has many dialects.

The explosion of Forth dialects led to the saying "If you've seen one Forth... you've seen *one* Forth."

Taxonomies:

Further information: Categorical list of programming languages

There is no overarching classification scheme for programming languages. A given programming language does not usually have a single ancestor language. Languages commonly arise by combining the elements of several predecessor languages with new ideas in circulation at the time. Ideas that originate in one language will diffuse throughout a family of related languages, and then leap suddenly across familial gaps to appear in an entirely different family.

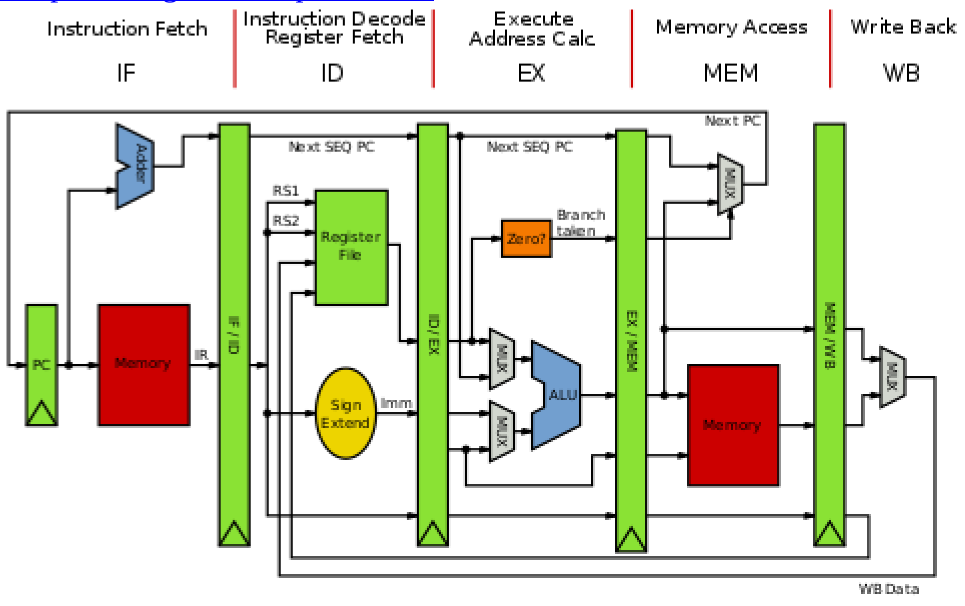
The task is further complicated by the fact that languages can be classified along multiple axes. For example, Java is both an object-oriented language (because it encourages object-oriented organization) and a concurrent language (because it contains built-in constructs for running multiple threads in parallel). Python is an object-oriented scripting language.

In broad strokes, programming languages divide into *programming paradigms* and a classification by *intended domain of use*, with general-purpose programming languages distinguished from domain-specific programming languages. Traditionally, programming languages have been regarded as describing computation in terms of imperative sentences, i.e. issuing commands. These are generally called imperative programming languages. A great deal of research in programming languages has been aimed at blurring the distinction between a program as a set of instructions and a program as an assertion about the desired answer, which is the main feature of declarative programming.[74] More refined paradigms include procedural programming, object-oriented programming, functional programming, and logic programming; some languages are hybrids of paradigms or multi-paradigmatic. An assembly language is not so much a paradigm as a direct model of an underlying machine architecture. By purpose, programming languages might be considered general purpose, system programming languages, scripting languages, domain-specific languages, or concurrent/distributed languages (or a combination of these).[75] Some general purpose languages were designed largely with educational goals.[76]

A programming language may also be classified by factors unrelated to programming paradigm. For instance, most programming languages use English language keywords, while a minority do not. Other languages may be classified as being deliberately esoteric or not.

Computer architecture

From Wikipedia, the free encyclopedia
Jump to navigationJump to search



A pipelined implementation of the MIPS architecture. Pipelining is a key concept in computer architecture.

In computer engineering, **computer architecture** is a set of rules and methods that describe the functionality, organization, and implementation of computer systems. Some definitions of architecture define it as describing the capabilities and programming model of a computer but not a particular implementation.[1] In other definitions computer architecture involves instruction set architecture design, microarchitecture design, logic design, and implementation.[2]

History:

The first documented computer architecture was in the correspondence between Charles Babbage and Ada Lovelace, describing the analytical engine. When building the computer Z1 in 1936, Konrad Zuse described in two patent applications for his future projects that machine instructions could be stored in the same storage used for data, i.e. the stored-program concept.[3][4] Two other early and important examples are:

- John von Neumann's 1945 paper, First Draft of a Report on the EDVAC, which described an organization of logical elements;[5] and
- Alan Turing's more detailed *Proposed Electronic Calculator* for the Automatic Computing Engine, also 1945 and which cited John von Neumann's paper.[6]

The term “architecture” in computer literature can be traced to the work of Lyle R. Johnson and Frederick P. Brooks, Jr., members of the Machine Organization department in IBM’s main research center in 1959. Johnson had the opportunity to write a proprietary research communication about the Stretch, an IBM-developed supercomputer for Los Alamos National Laboratory (at the time known as Los Alamos Scientific Laboratory). To describe the level of detail for discussing the luxuriously embellished computer, he noted that his description of formats, instruction types, hardware parameters, and speed enhancements were at the level of “system architecture” – a term that seemed more useful than “machine organization.”[7]

Subsequently, Brooks, a Stretch designer, started Chapter 2 of a book (Planning a Computer System: Project Stretch, ed. W. Buchholz, 1962) by writing,[8]

Computer architecture, like other architecture, is the art of determining the needs of the user of a structure and then designing to meet those needs as effectively as possible within economic and technological constraints.

Brooks went on to help develop the IBM System/360 (now called the IBM zSeries) line of computers, in which “architecture” became a noun defining “what the user needs to know”.[9] Later, computer users came to use the term in many less-explicit ways.[10]

The earliest computer architectures were designed on paper and then directly built into the final hardware form.[11] Later, computer architecture prototypes were physically built in the form of a transistor–transistor logic (TTL) computer—such as the prototypes of the 6800 and the PA-RISC—tested, and tweaked, before committing to the final hardware form. As of the 1990s, new computer architectures are typically "built", tested, and tweaked—inside some other computer architecture in a computer architecture simulator; or inside a FPGA as a soft microprocessor; or both—before committing to the final hardware form.[12]

Subcategories:

The discipline of computer architecture has three main subcategories:[13]

1. *Instruction Set Architecture*, or ISA. The ISA defines the machine code that a processor reads and acts upon as well as the word size, memory address modes, processor registers, and data type.
2. *Microarchitecture*, or *computer organization* describes how a particular processor will implement the ISA.[14] The size of a computer's CPU cache for instance, is an issue that generally has nothing to do with the ISA.
3. *System Design* includes all of the other hardware components within a computing system. These include:
   1. Data processing other than the CPU, such as direct memory access (DMA)
   2. Other issues such as virtualization, multiprocessing, and software features.

There are other types of computer architecture. The following types are used in bigger companies like Intel, and count for 1% of all of computer architecture

- Macroarchitecture: architectural layers more abstract than microarchitecture
- Assembly Instruction Set Architecture (ISA): A smart assembler may convert an abstract assembly language common to a group of machines into slightly different machine language for different implementations
- Programmer Visible Macroarchitecture: higher level language tools such as compilers may define a consistent interface or contract to programmers using them, abstracting differences between underlying ISA, UISA, and microarchitectures. E.g. the C, C++, or Java standards define different Programmer Visible Macroarchitecture.
- UISA (Microcode Instruction Set Architecture)—a group of machines with different hardware level microarchitectures may share a common microcode architecture, and hence a UISA.[*citation needed*]
- Pin Architecture: The hardware functions that a microprocessor should provide to a hardware platform, e.g., the x86 pins A20M, FERR/IGNNE or FLUSH. Also, messages that the processor should emit so that external caches can be invalidated (emptied). Pin architecture functions are more flexible than ISA functions because external hardware can adapt to new encodings, or change from a pin to a message. The term "architecture" fits, because the functions must be provided for compatible systems, even if the detailed method changes.

Roles:

Definition:

The purpose is to design a computer that maximizes performance while keeping power consumption in check, costs low relative to the amount of expected performance, and is also very reliable. For this, many aspects are to be considered which includes instruction set design, functional organization, logic design, and implementation. The implementation involves integrated circuit design, packaging, power, and cooling. Optimization of the design requires familiarity with compilers, operating systems to logic design, and packaging.[15]

Instruction set architecture:

Main article: Instruction set architecture

An instruction set architecture (ISA) is the interface between the computer's software and hardware and also can be viewed as the programmer's view of the machine. Computers do not understand high-level programming languages such as Java, C++, or most programming languages used. A processor only understands instructions encoded in some numerical fashion, usually as binary numbers. Software tools, such as compilers, translate those high level languages into instructions that the processor can understand.

Besides instructions, the ISA defines items in the computer that are available to a program—e.g. data types, registers, addressing modes, and memory. Instructions locate these available items with register indexes (or names) and memory addressing modes.

The ISA of a computer is usually described in a small instruction manual, which describes how the instructions are encoded. Also, it may define short (vaguely) mnemonic names for the instructions. The names can be recognized by a software development tool called an assembler. An assembler is a computer program that translates a human-readable form of the ISA into a computer-readable form. Disassemblers are also widely available, usually in debuggers and software programs to isolate and correct malfunctions in binary computer programs.

ISAs vary in quality and completeness. A good ISA compromises between programmer convenience (how easy the code is to understand), size of the code (how much code is required to do a specific action), cost of the computer to interpret the instructions (more complexity means more hardware needed to decode and execute the instructions), and speed of the computer (with more complex decoding hardware comes longer decode time). Memory organization defines how instructions interact with the memory, and how memory interacts with itself.

During design emulation software (emulators) can run programs written in a proposed instruction set. Modern emulators can measure size, cost, and speed to determine if a particular ISA is meeting its goals.

Computer organization:

Main article: Microarchitecture

Computer organization helps optimize performance-based products. For example, software engineers need to know the processing power of processors. They may need to optimize software in order to gain the most performance for the lowest price. This can require quite detailed analysis of the computer's organization. For example, in a SD card, the designers might need to arrange the card so that the most data can be processed in the fastest possible way.

Computer organization also helps plan the selection of a processor for a particular project. Multimedia projects may need very rapid data access, while virtual machines may need fast interrupts. Sometimes certain tasks need additional components as well. For example, a computer capable of running a virtual machine needs virtual memory hardware so that the memory of different virtual computers can be kept separated. Computer organization and features also affect power consumption and processor cost.

Implementation:

Main article: Implementation

Once an instruction set and micro-architecture are designed, a practical machine must be developed. This design process is called the *implementation*. Implementation is usually not considered architectural design, but rather hardware design engineering. Implementation can be further broken down into several steps:

- **Logic Implementation** designs the circuits required at a logic gate level
- **Circuit Implementation** does transistor-level designs of basic elements (gates, multiplexers, latches etc.) as well as of some larger blocks (ALUs, caches etc.) that may be implemented at the log gate level, or even at the physical level if the design calls for it.
- **Physical Implementation** draws physical circuits. The different circuit components are placed in a chip floorplan or on a board and the wires connecting them are created.
- **Design Validation** tests the computer as a whole to see if it works in all situations and all timings. Once the design validation process starts, the design at the logic level are tested using logic emulators. However, this is usually too slow to run realistic test. So, after making corrections based on the first test, prototypes are constructed using Field-Programmable Gate-Arrays (FPGAs). Most hobby projects stop at this stage. The final step is to test prototype integrated circuits. Integrated circuits may require several redesigns to fix problems.

For CPUs, the entire implementation process is organized differently and is often referred to as CPU design.

Design goals[edit]

The exact form of a computer system depends on the constraints and goals. Computer architectures usually trade off standards, power versus performance, cost, memory capacity, latency (latency is the amount of time that it takes for information from one node to travel to the source) and throughput. Sometimes other considerations, such as features, size, weight, reliability, and expandability are also factors.

The most common scheme does an in depth power analysis and figures out how to keep power consumption low, while maintaining adequate performance.

Performance[edit]

Modern computer performance is often described in IPC (instructions per cycle). This measures the efficiency of the architecture at any clock frequency. Since a faster rate can make a faster computer, this is a useful measurement. Older computers had IPC counts as low as 0.1 instructions per cycle. Simple modern processors easily reach near 1. Superscalar processors may reach three to five IPC by executing several instructions per clock cycle.

Counting machine language instructions would be misleading because they can do varying amounts of work in different ISAs. The "instruction" in the standard measurements is not a count of the ISA's actual machine language instructions, but a unit of measurement, usually based on the speed of the VAX computer architecture.

Many people used to measure a computer's speed by the clock rate (usually in MHz or GHz). This refers to the cycles per second of the main clock of the CPU. However, this metric is somewhat misleading, as a machine with a higher clock rate may not necessarily have greater performance. As a result, manufacturers have moved away from clock speed as a measure of performance.

Other factors influence speed, such as the mix of functional units, bus speeds, available memory, and the type and order of instructions in the programs.

There are two main types of speed: latency and throughput. Latency is the time between the start of a process and its completion. Throughput is the amount of work done per unit time. Interrupt latency is the guaranteed maximum response time of the system to an electronic event (like when the disk drive finishes moving some data).

Performance is affected by a very wide range of design choices — for example, pipelining a processor usually makes latency worse, but makes throughput better. Computers that control machinery usually need low interrupt latencies. These computers operate in a real-time environment and fail if an operation is not completed in a specified amount of time. For example, computer-controlled anti-lock brakes must begin braking within a predictable, short time after the brake pedal is sensed or else failure of the brake will occur.

Benchmarking takes all these factors into account by measuring the time a computer takes to run through a series of test programs. Although benchmarking shows strengths, it shouldn't be how you choose a computer. Often the measured machines split on different measures. For example, one system might handle scientific applications quickly, while another might render video games more smoothly. Furthermore, designers may target and add special features to their products, through hardware or software, that permit a specific benchmark to execute quickly but don't offer similar advantages to general tasks.

Power efficiency[edit]

Main article: Low-power electronics

Power efficiency is another important measurement in modern computers. A higher power efficiency can often be traded for lower speed or higher cost. The typical measurement when referring to power consumption in computer architecture is MIPS/W (millions of instructions per second per watt).

Modern circuits have less power required per transistor as the number of transistors per chip grows.[16] This is because each transistor that is put in a new chip requires its own power supply and requires new pathways to be built to power it. However the number of transistors per chip is starting to increase at a slower rate. Therefore, power efficiency is starting to become as important, if not more important than fitting more and more transistors into a single chip. Recent processor designs have shown this emphasis as they put more focus on power efficiency rather than cramming as many transistors into a single chip as possible.[17] In the world of embedded computers, power efficiency has long been an important goal next to throughput and latency.

Shifts in market demand[edit]

Increases in publicly released refresh rates have grown slowly over the past few years, with respect to vast leaps in power consumption reduction and miniaturization demand. This has led to a new demand for longer battery life and reductions in size due to the mobile technology being produced at a greater rate. This change in focus from greater refresh rates to power consumption and miniaturization can be shown by the significant reductions in power consumption, as much as 50%, that were reported by Intel in their release of the Haswell microarchitecture; where they dropped their power consumption benchmark from 30-40 watts down to 10-20 watts.[18] Comparing this to the processing speed increase of 3 GHz to 4 GHz (2002 to 2006)[19] it can be seen that the focus in research and development are shifting away from refresh rates and moving towards consuming less power and taking up less space.

Computer security

From Wikipedia, the free encyclopedia
Jump to navigationJump to search

Computer security, **cybersecurity**[1], or **IT security** is the protection of computer systems from theft or damage to their hardware, software or electronic data, as well as from disruption or misdirection of the services they provide.

The field is of growing importance due to increasing reliance on computer systems, the Internet[2] and wireless networks such as Bluetooth and Wi-Fi, and due to the growth of "smart" devices, including smartphones, televisions and the various tiny devices that constitute the Internet of Things. Due to its complexity, both in terms of politics and technology, it is also one of the major challenges of contemporary world.[3]

Vulnerabilities and attacks:

Main article: Vulnerability (computing)

A vulnerability is a weakness in design, implementation, operation or internal control. Most of the vulnerabilities that have been discovered are documented in the Common Vulnerabilities and Exposures (CVE) database.

An *exploitable* vulnerability is one for which at least one working attack or "exploit" exists.[4] Vulnerabilities are often hunted or exploited with the aid of automated tools or manually using customized scripts.

To secure a computer system, it is important to understand the attacks that can be made against it, and these threats can typically be classified into one of these categories below:

Backdoor:

A backdoor in a computer system, a cryptosystem or an algorithm, is any secret method of bypassing normal authentication or security controls. They may exist for a number of reasons, including by original design or from poor configuration. They may have been added by an authorized party to allow some legitimate access, or by an attacker for malicious reasons; but regardless of the motives for their existence, they create a vulnerability.

Denial-of-service attacks:

Denial of service attacks (DoS) are designed to make a machine or network resource unavailable to its intended users.[5] Attackers can deny service to individual victims, such as by deliberately entering a wrong password enough consecutive times to cause the victims account to be locked, or they may overload the capabilities of a machine or network and block all users at once. While a network attack from a single IP address can be blocked by adding a new firewall rule, many forms of Distributed denial of service (DDoS) attacks are possible, where the attack comes from a large number of points – and defending is much more difficult. Such attacks can originate from the zombie computers of a botnet, but a range of other techniques are possible including reflection and amplification attacks, where innocent systems are fooled into sending traffic to the victim.

Direct-access attacks:

An unauthorized user gaining physical access to a computer is most likely able to directly copy data from it. They may also compromise security by making operating system modifications, installing software worms, keyloggers, covert listening devices or using wireless mice.[6] Even when the system is protected by standard security measures, these may be able to be by-passed by booting another operating system or tool from a CD-ROM or other bootable media. Disk encryption and Trusted Platform Module are designed to prevent these attacks.

Eavesdropping:

Eavesdropping is the act of surreptitiously listening to a private conversation, typically between hosts on a network. For instance, programs such as Carnivore and NarusInSight have been used by the FBI and NSA to eavesdrop on the systems of internet service providers. Even machines that operate as a closed system (i.e., with no contact to the outside world) can be eavesdropped upon via monitoring the faint electro-magnetic transmissions generated by the hardware; TEMPEST is a specification by the NSA referring to these attacks.

Multivector, polymorphic attacks:

Surfacing in 2017, a new class of multi-vector,[7] polymorphic[8] cyber threats surfaced that combined several types of attacks and changed form to avoid cybersecurity controls as they spread. These threats have been classified as fifth generation cyberattacks.[9]

Phishing:

Phishing is the attempt to acquire sensitive information such as usernames, passwords, and credit card details directly from users.[10] Phishing is typically carried out by email spoofing or instant messaging, and it often directs users to enter details at a fake website whose look and feel are almost identical to the legitimate one. Preying on a victim's trust, phishing can be classified as a form of social engineering.

Privilege escalation:

Privilege escalation describes a situation where an attacker with some level of restricted access is able to, without authorization, elevate their privileges or access level. For example, a standard computer user may be able to fool the system into giving them access to restricted data; or even to "become root" and have full unrestricted access to a system.

Social engineering:

Main article: Social engineering (security)

Social engineering aims to convince a user to disclose secrets such as passwords, card numbers, etc. by, for example, impersonating a bank, a contractor, or a customer.[11]

A common scam involves fake CEO emails sent to accounting and finance departments. In early 2016, the FBI reported that the scam has cost US businesses more than $2bn in about two years.[12]

In May 2016, the Milwaukee Bucks NBA team was the victim of this type of cyber scam with a perpetrator impersonating the team's president Peter Feigin, resulting in the handover of all the team's employees' 2015 W-2 tax forms.[13]

Spoofing:

Main article: Spoofing attack

Spoofing is the act of masquerading as a valid entity through falsification of data (such as an IP address or username), in order to gain access to information or resources that one is otherwise unauthorized to obtain.[14][15] There are several types of spoofing, including:

- Email spoofing, where an attacker forges the sending (*From*, or source) address of an email.
- IP address spoofing, where an attacker alters the source IP address in a network packet to hide their identity or impersonate another computing system.
- MAC spoofing, where an attacker modifies the Media Access Control (MAC) address of their network interface to pose as a valid user on a network.
- Biometric spoofing, where an attacker produces a fake biometric sample to pose as another user.[16]

Tampering:

Tampering describes a malicious modification of products. So-called "Evil Maid" attacks and security services planting of surveillance capability into routers[17] are examples.

Information security culture:

Employee behavior can have a big impact on information security in organizations. Cultural concepts can help different segments of the organization work effectively or work against effectiveness towards information security within an organization. "Exploring the Relationship between Organizational Culture and Information Security Culture″ provides the following definition of information security culture: ″ISC is the totality of patterns of behavior in an organization that contribute to the protection of information of all kinds."[18]

Andersson and Reimers (2014) found that employees often do not see themselves as part of the organization Information Security "effort" and often take actions that ignore organizational Information Security best interests.[19] Research shows Information security culture needs to be improved continuously. In ″Information Security Culture from Analysis to Change″, authors commented, ″It′s a never ending process, a cycle of evaluation and change or maintenance.″ To manage the information security culture, five steps should be taken: Pre-evaluation, strategic planning, operative planning, implementation, and post-evaluation.[20]

- Pre-Evaluation: to identify the awareness of information security within employees and to analyze the current security policy.
- Strategic Planning: to come up with a better awareness program, clear targets need to be set. Clustering[*definition needed*] people is helpful to achieve it.
- Operative Planning: a good security culture can be established based on internal communication, management-buy-in, and security awareness and a training program.[20]
- Implementation: four stages should be used to implement the information security culture. They are:

1. Commitment of the management
2. Communication with organizational members
3. Courses for all organizational members
4. Commitment of the employees[20]

Systems at risk:

The growth in the number of computer systems, and the increasing reliance upon them of individuals, businesses, industries and governments means that there are an increasing number of systems at risk.

Financial systems:

The computer systems of financial regulators and financial institutions like the U.S. Securities and Exchange Commission, SWIFT, investment banks, and commercial banks are prominent hacking targets for cybercriminals interested in manipulating markets and making illicit gains.[21] Web sites and apps that accept or store credit card numbers, brokerage accounts, and bank account information are also prominent hacking targets, because of the potential for immediate financial gain from transferring money, making purchases, or selling the information on the black market.[22] In-store payment systems and ATMs have also been tampered with in order to gather customer account data and PINs.

Utilities and industrial equipment:

Computers control functions at many utilities, including coordination of telecommunications, the power grid, nuclear power plants, and valve opening and closing in water and gas networks. The Internet is a potential attack vector for such machines if connected, but the Stuxnet worm demonstrated that even equipment controlled by computers not connected to the Internet can be vulnerable. In 2014, the Computer Emergency Readiness Team, a division of the Department of Homeland Security, investigated 79 hacking incidents at energy companies.[23] Vulnerabilities in smart meters (many of which use local radio or cellular communications) can cause problems with billing fraud.[24]

Aviation:

The aviation industry is very reliant on a series of complex systems which could be attacked.[25] A simple power outage at one airport can cause repercussions worldwide,[26] much of the system relies on radio transmissions which could be disrupted,[27] and controlling aircraft over oceans is especially dangerous because radar surveillance only extends 175 to 225 miles offshore.[28] There is also potential for attack from within an aircraft.[29]

In Europe, with the (Pan-European Network Service)[30] and NewPENS,[31] and in the US with the NextGen program,[32] air navigation service providers are moving to create their own dedicated networks.

The consequences of a successful attack range from loss of confidentiality to loss of system integrity, air traffic control outages, loss of aircraft, and even loss of life.

Consumer devices:

Desktop computers and laptops are commonly targeted to gather passwords or financial account information, or to construct a botnet to attack another target. Smartphones, tablet computers, smart watches, and other mobile devices such as quantified self devices like activity trackers have sensors such as cameras, microphones, GPS receivers, compasses, and accelerometers which could be exploited, and may collect personal information, including sensitive health information. Wifi, Bluetooth, and cell phone networks on any of these devices could be used as attack vectors, and sensors might be remotely activated after a successful breach.[33]

The increasing number of home automation devices such as the Nest thermostat are also potential targets.[33]

Large corporations:

Large corporations are common targets. In many cases this is aimed at financial gain through identity theft and involves data breaches such as the loss of millions of clients' credit card details by Home Depot,[34] Staples,[35] Target Corporation,[36] and the most recent breach of Equifax.[37]

Some cyberattacks are ordered by foreign governments, these governments engage in cyberwarfare with the intent to spread their propaganda, sabotage, or spy on their targets. Many people believe the Russian government played a major role in the US presidential election of 2016 by using Twitter and Facebook to affect the results of the election.[38]

Medical records have been targeted for use in general identify theft, health insurance fraud, and impersonating patients to obtain prescription drugs for recreational purposes or resale.[39] Although cyber threats continue to increase, 62% of all organizations did not increase security training for their business in 2015.[40][41]

Not all attacks are financially motivated however; for example security firm HBGary Federal suffered a serious series of attacks in 2011 from hacktivist group Anonymous in retaliation for the firm's CEO claiming to have infiltrated their group,[42][43] and in the Sony Pictures attack of 2014 the motive appears to have been to embarrass with data leaks, and cripple the company by wiping workstations and servers.[44][45]

Automobiles:

See also: Autonomous car § Potential disadvantages, Automated driving system § Risks and liabilities, and Automotive hacking

Vehicles are increasingly computerized, with engine timing, cruise control, anti-lock brakes, seat belt tensioners, door locks, airbags and advanced driver-assistance systems on many models. Additionally, connected cars may use WiFi and Bluetooth to communicate with onboard consumer devices and the cell phone network.[46] Self-driving cars are expected to be even more complex.

All of these systems carry some security risk, and such issues have gained wide attention.[47][48][49] Simple examples of risk include a malicious compact disc being used as an attack vector,[50] and the car's onboard microphones being used for eavesdropping. However, if access is gained to a car's internal controller area network, the danger is much greater[46] – and in a widely publicised 2015 test, hackers remotely carjacked a vehicle from 10 miles away and drove it into a ditch.[51][52]

Manufacturers are reacting in a number of ways, with Tesla in 2016 pushing out some security fixes "over the air" into its cars' computer systems.[53]

In the area of autonomous vehicles, in September 2016 the United States Department of Transportation announced some initial safety standards, and called for states to come up with uniform policies.[54][55]

Government:

Government and military computer systems are commonly attacked by activists[56][57][58][59] and foreign powers.[60][61][62][63] Local and regional government infrastructure such as traffic light controls, police and intelligence agency communications, personnel records, student records,[64] and financial systems are also potential targets as they are now all largely computerized. Passports and government ID cards that control access to facilities which use RFID can be vulnerable to cloning.

Internet of things and physical vulnerabilities:

The Internet of things (IoT) is the network of physical objects such as devices, vehicles, and buildings that are embedded with electronics, software, sensors, and network connectivity that enables them to collect and exchange data[65] – and concerns have been raised that this is being developed without appropriate consideration of the security challenges involved.[66][67]

While the IoT creates opportunities for more direct integration of the physical world into computer-based systems,[68][69] it also provides opportunities for misuse. In particular, as the Internet of Things spreads widely, cyber attacks are likely to become an increasingly physical (rather than simply virtual) threat.[70] If a front door's lock is connected to the Internet, and can be locked/unlocked from a phone, then a criminal could enter the home at the press of a button from a stolen or hacked phone. People could stand to lose much more than their credit card numbers in a world controlled by IoT-enabled devices. Thieves have also used electronic means to circumvent non-Internet-connected hotel door locks.[71]

Medical systems:

See also: Medical device hijack and Medical data breach

Medical devices have either been successfully attacked or had potentially deadly vulnerabilities demonstrated, including both in-hospital diagnostic equipment[72] and implanted devices including pacemakers[73] and insulin pumps.[74] There are many reports of hospitals and hospital organizations getting hacked, including ransomware attacks,[75][76][77][78] Windows XP exploits,[79][80] viruses,[81][82][83] and data breaches of sensitive data stored on hospital servers.[84][76][85][86][87] On 28 December 2016 the US Food and Drug Administration released its recommendations for how medical device manufacturers should maintain the security of Internet-connected devices – but no structure for enforcement.[88][89]

Energy sector:

In distributed generation systems, the risk of cyber attacks is real, according to *Daily Energy Insider*. An attack could cause a loss of power in a large area for a long period of time, and such an attack could have just as severe consequences as a natural disaster. The District of Columbia is considering creating a Distributed Energy Resources (DER) Authority within the city, with the goal being for customers to have more insight into their own energy use and giving the local electric utility, Pepco, the chance to better estimate energy demand. The D.C. proposal, however, would "allow third-party vendors to create numerous points of energy distribution, which could potentially create more opportunities for cyberattackers to threaten the electric grid."[90]

Impact of security breaches:

Serious financial damage has been caused by security breaches, but because there is no standard model for estimating the cost of an incident, the only data available is that which is made public by the organizations involved. "Several computer security consulting firms produce estimates of total worldwide losses attributable to virus and worm attacks and to hostile digital acts in general. The 2003 loss estimates by these firms range from $13 billion (worms and viruses only) to $226 billion (for all forms of covert attacks). The reliability of these estimates is often challenged; the underlying methodology is basically anecdotal."[91] Security breaches continue to cost businesses billions of dollars but a survey revealed that 66% of security staffs do not believe senior leadership takes cyber precautions as a strategic priority.[40][*third-party source needed*]

However, reasonable estimates of the financial cost of security breaches can actually help organizations make rational investment decisions. According to the classic Gordon-Loeb Model analyzing the optimal investment level in information security, one can conclude that the amount a firm spends to protect information should generally be only a small fraction of the expected loss (i.e., the expected value of the loss resulting from a cyber/information security breach).[92]

Attacker motivation:

As with physical security, the motivations for breaches of computer security vary between attackers. Some are thrill-seekers or vandals, some are activists, others are criminals looking for financial gain. State-sponsored attackers are now common and well resourced, but started with amateurs such as Markus Hess who hacked for the KGB, as recounted by Clifford Stoll, in *The Cuckoo's Egg*.

A standard part of threat modelling for any particular system is to identify what might motivate an attack on that system, and who might be motivated to breach it. The level and detail of precautions will vary depending on the system to be secured. A home personal computer, bank, and classified military network face very different threats, even when the underlying technologies in use are similar.

Computer protection (countermeasures):

In computer security a countermeasure is an action, device, procedure, or technique that reduces a threat, a vulnerability, or an attack by eliminating or preventing it, by minimizing the harm it can cause, or by discovering and reporting it so that corrective action can be taken.[93][94][95]

Some common countermeasures are listed in the following sections:

Security by design:

Main article: Secure by design

Security by design, or alternately secure by design, means that the software has been designed from the ground up to be secure. In this case, security is considered as a main feature.

Some of the techniques in this approach include:

- The principle of least privilege, where each part of the system has only the privileges that are needed for its function. That way even if an attacker gains access to that part, they have only limited access to the whole system.
- Automated theorem proving to prove the correctness of crucial software subsystems.
- Code reviews and unit testing, approaches to make modules more secure where formal correctness proofs are not possible.
- Defense in depth, where the design is such that more than one subsystem needs to be violated to compromise the integrity of the system and the information it holds.
- Default secure settings, and design to "fail secure" rather than "fail insecure" (see fail-safe for the equivalent in safety engineering). Ideally, a secure system should require a deliberate, conscious, knowledgeable and free decision on the part of legitimate authorities in order to make it insecure.
- Audit trails tracking system activity, so that when a security breach occurs, the mechanism and extent of the breach can be determined. Storing audit trails remotely, where they can only be appended to, can keep intruders from covering their tracks.
- Full disclosure of all vulnerabilities, to ensure that the "window of vulnerability" is kept as short as possible when bugs are discovered.

Security architecture:

The Open Security Architecture organization defines IT security architecture as "the design artifacts that describe how the security controls (security countermeasures) are positioned, and how they relate to the overall information technology architecture. These controls serve the purpose to maintain the system's quality attributes: confidentiality, integrity, availability, accountability and assurance services".[96]

Techopedia defines security architecture as "a unified security design that addresses the necessities and potential risks involved in a certain scenario or environment. It also specifies when and where to apply security controls. The design process is generally reproducible." The key attributes of security architecture are:[97]

- the relationship of different components and how they depend on each other.
- the determination of controls based on risk assessment, good practice, finances, and legal matters.
- the standardization of controls.

Security measures:

A state of computer "security" is the conceptual ideal, attained by the use of the three processes: threat prevention, detection, and response. These processes are based on various policies and system components, which include the following:

- User account access controls and cryptography can protect systems files and data, respectively.
- Firewalls are by far the most common prevention systems from a network security perspective as they can (if properly configured) shield access to internal network services, and block certain kinds of attacks through packet filtering. Firewalls can be both hardware- or software-based.
- Intrusion Detection System (IDS) products are designed to detect network attacks in-progress and assist in post-attack forensics, while audit trails and logs serve a similar function for individual systems.
- "Response" is necessarily defined by the assessed security requirements of an individual system and may cover the range from simple upgrade of protections to notification of legal authorities, counter-

attacks, and the like. In some special cases, a complete destruction of the compromised system is favored, as it may happen that not all the compromised resources are detected.

Today, computer security comprises mainly "preventive" measures, like firewalls or an exit procedure. A firewall can be defined as a way of filtering network data between a host or a network and another network, such as the Internet, and can be implemented as software running on the machine, hooking into the network stack (or, in the case of most UNIX-based operating systems such as Linux, built into the operating system kernel) to provide real time filtering and blocking. Another implementation is a so-called "physical firewall", which consists of a separate machine filtering network traffic. Firewalls are common amongst machines that are permanently connected to the Internet.

Some organizations are turning to big data platforms, such as Apache Hadoop, to extend data accessibility and machine learning to detect advanced persistent threats.[98][99]

However, relatively few organisations maintain computer systems with effective detection systems, and fewer still have organised response mechanisms in place. As a result, as Reuters points out: "Companies for the first time report they are losing more through electronic theft of data than physical stealing of assets".[100] The primary obstacle to effective eradication of cyber crime could be traced to excessive reliance on firewalls and other automated "detection" systems. Yet it is basic evidence gathering by using packet capture appliances that puts criminals behind bars.[*citation needed*]

Vulnerability management:

Main article: Vulnerability management

Vulnerability management is the cycle of identifying, and remediating or mitigating vulnerabilities,[101] especially in software and firmware. Vulnerability management is integral to computer security and network security.

Vulnerabilities can be discovered with a vulnerability scanner, which analyzes a computer system in search of known vulnerabilities,[102] such as open ports, insecure software configuration, and susceptibility to malware.

Beyond vulnerability scanning, many organisations contract outside security auditors to run regular penetration tests against their systems to identify vulnerabilities. In some sectors this is a contractual requirement.[103]

Reducing vulnerabilities:

While formal verification of the correctness of computer systems is possible,[104][105] it is not yet common. Operating systems formally verified include seL4,[106] and SYSGO's PikeOS[107][108] – but these make up a very small percentage of the market.

Two factor authentication is a method for mitigating unauthorized access to a system or sensitive information. It requires "something you know"; a password or PIN, and "something you have"; a card, dongle, cellphone, or other piece of hardware. This increases security as an unauthorized person needs both of these to gain access.

Social engineering and direct computer access (physical) attacks can only be prevented by non-computer means, which can be difficult to enforce, relative to the sensitivity of the information. Training is often involved to help mitigate this risk, but even in a highly disciplined environments (e.g. military organizations), social engineering attacks can still be difficult to foresee and prevent.

Enoculation, derived from inoculation theory, seeks to prevent social engineering and other fraudulent tricks or traps by instilling a resistance to persuasion attempts through exposure to similar or related attempts.[109]

It is possible to reduce an attacker's chances by keeping systems up to date with security patches and updates, using a security scanner[*definition needed*] or/and hiring competent people responsible for security.(This statement is ambiguous. Even systems developed by "competent" people get penetrated) The effects of data loss/damage can be reduced by careful backing up and insurance.

Hardware protection mechanisms:

See also: Computer security compromised by hardware failure

While hardware may be a source of insecurity, such as with microchip vulnerabilities maliciously introduced during the manufacturing process,[110][111] hardware-based or assisted computer security also offers an alternative to software-only computer security. Using devices and methods such as dongles, trusted platform modules, intrusion-aware cases, drive locks, disabling USB ports, and mobile-enabled access may be considered more secure due to the physical access (or sophisticated backdoor access) required in order to be compromised. Each of these is covered in more detail below.

- USB dongles are typically used in software licensing schemes to unlock software capabilities,[112] but they can also be seen as a way to prevent unauthorized access to a computer or other device's software. The dongle, or key, essentially creates a secure encrypted tunnel between the software application and the key. The principle is that an encryption scheme on the dongle, such as Advanced Encryption Standard (AES) provides a stronger measure of security, since it is harder to hack and replicate the dongle than to simply copy the native software to another machine and use it. Another security application for dongles is to use them for accessing web-based content such as cloud software or Virtual Private Networks (VPNs).[113] In addition, a USB dongle can be configured to lock or unlock a computer.[114]
- Trusted platform modules (TPMs) secure devices by integrating cryptographic capabilities onto access devices, through the use of microprocessors, or so-called computers-on-a-chip. TPMs used in conjunction with server-side software offer a way to detect and authenticate hardware devices, preventing unauthorized network and data access.[115]
- Computer case intrusion detection refers to a device, typically a push-button switch, which detects when a computer case is opened. The firmware or BIOS is programmed to show an alert to the operator when the computer is booted up the next time.
- Drive locks are essentially software tools to encrypt hard drives, making them inaccessible to thieves.[116] Tools exist specifically for encrypting external drives as well.[117]
- Disabling USB ports is a security option for preventing unauthorized and malicious access to an otherwise secure computer. Infected USB dongles connected to a network from a computer inside the firewall are considered by the magazine Network World as the most common hardware threat facing computer networks.
- Mobile-enabled access devices are growing in popularity due to the ubiquitous nature of cell phones. Built-in capabilities such as Bluetooth, the newer Bluetooth low energy (LE), Near field communication (NFC) on non-iOS devices and biometric validation such as thumb print readers, as well as QR code reader software designed for mobile devices, offer new, secure ways for mobile phones to connect to access control systems. These control systems provide computer security and can also be used for controlling access to secure buildings.[118]

Secure operating systems:

Main article: Security-evaluated operating system

One use of the term "computer security" refers to technology that is used to implement secure operating systems. In the 1980s the United States Department of Defense (DoD) used the "Orange Book"[119] standards, but the current international standard ISO/IEC 15408, "Common Criteria" defines a number of progressively more stringent Evaluation Assurance Levels. Many common operating systems meet the EAL4 standard of being "Methodically Designed, Tested and Reviewed", but the formal verification required for the highest levels means that they are uncommon. An example of an EAL6 ("Semiformally Verified Design and Tested") system is Integrity-178B, which is used in the Airbus A380[120] and several military jets.[121]

Secure coding:

Main article: Secure coding

In software engineering, secure coding aims to guard against the accidental introduction of security vulnerabilities. It is also possible to create software designed from the ground up to be secure. Such systems are "secure by design". Beyond this, formal verification aims to prove the correctness of the algorithms underlying a system;[122] important for cryptographic protocols for example.

Capabilities and access control lists:

Main articles: Access control list and Capability (computers)

Within computer systems, two of many security models capable of enforcing privilege separation are access control lists (ACLs) and capability-based security. Using ACLs to confine programs has been proven to be insecure in many situations, such as if the host computer can be tricked into indirectly allowing restricted file access, an issue known as the confused deputy problem. It has also been shown that the promise of ACLs of giving access to an object to only one person can never be guaranteed in practice. Both of these problems are resolved by capabilities. This does not mean practical flaws exist in all ACL-based systems, but only that the designers of certain utilities must take responsibility to ensure that they do not introduce flaws.[*citation needed*]

Capabilities have been mostly restricted to research operating systems, while commercial OSs still use ACLs. Capabilities can, however, also be implemented at the language level, leading to a style of programming that is essentially a refinement of standard object-oriented design. An open source project in the area is the E language.

End user security training:

Repeated education/training in security "best practices" can have a marked effect on compliance with good end user network security habits—which particularly protect against phishing, ransomware and other forms of malware which have a social engineering aspect.[19]

Response to breaches:

Responding forcefully to attempted security breaches (in the manner that one would for attempted physical security breaches) is often very difficult for a variety of reasons:

- Identifying attackers is difficult, as they are often in a different jurisdiction to the systems they attempt to breach, and operate through proxies, temporary anonymous dial-up accounts, wireless connections, and other anonymising procedures which make backtracing difficult and are often located in yet another jurisdiction. If they successfully breach security, they are often able to delete logs to cover their tracks.
- The sheer number of attempted attacks is so large that organisations cannot spend time pursuing each attacker (a typical home user with a permanent (e.g., cable modem) connection will be attacked at

least several times per day, so more attractive targets could be presumed to see many more). Note however, that most of the sheer bulk of these attacks are made by automated vulnerability scanners and computer worms.

- Law enforcement officers are often unfamiliar with information technology, and so lack the skills and interest in pursuing attackers. There are also budgetary constraints. It has been argued that the high cost of technology, such as DNA testing, and improved forensics mean less money for other kinds of law enforcement, so the overall rate of criminals not getting dealt with goes up as the cost of the technology increases. In addition, the identification of attackers across a network may require logs from various points in the network and in many countries, the release of these records to law enforcement (with the exception of being voluntarily surrendered by a network administrator or a system administrator) requires a search warrant and, depending on the circumstances, the legal proceedings required can be drawn out to the point where the records are either regularly destroyed, or the information is no longer relevant.
- The United States government spends the largest amount of money every year on cyber security. The United States has a yearly budget of 28 billion dollars. Canada has the 2nd highest annual budget at 1 billion dollars. Australia has the third highest budget with only 70 million dollars.[123]

Types of security and privacy:

- Access control
- Anti-keyloggers
- Anti-malware
- Anti-spyware
- Anti-subversion software
- Anti-tamper software
- Antivirus software
- Cryptographic software
- Computer-aided dispatch (CAD)
- Firewall
- Intrusion detection system (IDS)
- Intrusion prevention system (IPS)
- Log management software
- Records management
- Sandbox
- Security information management
- SIEM
- Anti-theft
- Parental control
- Software and operating system updating

Notable attacks and breaches:

Further information: List of cyber-attacks and List of data breaches

Some illustrative examples of different types of computer security breaches are given below.

Robert Morris and the first computer worm:

Main article: Morris worm

In 1988, only 60,000 computers were connected to the Internet, and most were mainframes, minicomputers and professional workstations. On 2 November 1988, many started to slow down, because they were running a malicious code that demanded processor time and that spread itself to other computers – the first internet "computer worm".[124] The software was traced back to 23-year-old Cornell University graduate student Robert Tappan Morris, Jr. who said 'he wanted to count how many machines were connected to the Internet'.[124]

Rome Laboratory:

In 1994, over a hundred intrusions were made by unidentified crackers into the Rome Laboratory, the US Air Force's main command and research facility. Using trojan horses, hackers were able to obtain unrestricted access to Rome's networking systems and remove traces of their activities. The intruders were able to obtain classified files, such as air tasking order systems data and furthermore able to penetrate connected networks of National Aeronautics and Space Administration's Goddard Space Flight Center, Wright-Patterson Air Force Base, some Defense contractors, and other private sector organizations, by posing as a trusted Rome center user.[125]

TJX customer credit card details:

In early 2007, American apparel and home goods company TJX announced that it was the victim of an unauthorized computer systems intrusion[126] and that the hackers had accessed a system that stored data on credit card, debit card, check, and merchandise return transactions.[127]

Stuxnet attack:

The computer worm known as Stuxnet reportedly ruined almost one-fifth of Iran's nuclear centrifuges[128] by disrupting industrial programmable logic controllers (PLCs) in a targeted attack generally believed to have been launched by Israel and the United States[129][130][131][132] – although neither has publicly admitted this.

Global surveillance disclosures:

Main article: Global surveillance disclosures (2013–present)

In early 2013, documents provided by Edward Snowden were published by *The Washington Post* and *The Guardian*[133][134] exposing the massive scale of NSA global surveillance. There were also indications that the NSA may have inserted a backdoor in a NIST standard for encryption.[135] This standard was later withdrawn due to widespread criticism.[136] The NSA additionally were revealed to have tapped the links between Google's data centres.[137]

Target and Home Depot breaches:

In 2013 and 2014, a Russian/Ukrainian hacking ring known as "Rescator" broke into Target Corporation computers in 2013, stealing roughly 40 million credit cards,[138] and then Home Depot computers in 2014, stealing between 53 and 56 million credit card numbers.[139] Warnings were delivered at both corporations, but ignored; physical security breaches using self checkout machines are believed to have played a large role. "The malware utilized is absolutely unsophisticated and uninteresting," says Jim Walter, director of threat intelligence operations at security technology company McAfee – meaning that the heists could have easily been stopped by existing antivirus software had administrators responded to the warnings. The size of the thefts has resulted in major attention from state and Federal United States authorities and the investigation is ongoing.

Office of Personnel Management data breach:

In April 2015, the Office of Personnel Management discovered it had been hacked more than a year earlier in a data breach, resulting in the theft of approximately 21.5 million personnel records handled by the office.[140] The Office of Personnel Management hack has been described by federal officials as among the largest breaches of government data in the history of the United States.[141] Data targeted in the breach included personally identifiable information such as Social Security Numbers,[142] names, dates and places of birth, addresses, and fingerprints of current and former government employees as well as anyone who had undergone a government background check.[143] It is believed the hack was perpetrated by Chinese hackers but the motivation remains unclear.[144]

Ashley Madison breach:

Main article: Ashley Madison Data Breach

In July 2015, a hacker group known as "The Impact Team" successfully breached the extramarital relationship website Ashley Madison. The group claimed that they had taken not only company data but user data as well. After the breach, The Impact Team dumped emails from the company's CEO, to prove their point, and threatened to dump customer data unless the website was taken down permanently. With this initial data release, the group stated "Avid Life Media has been instructed to take Ashley Madison and Established Men offline permanently in all forms, or we will release all customer records, including profiles with all the customers' secret sexual fantasies and matching credit card transactions, real names and addresses, and employee documents and emails. The other websites may stay online."[145] When Avid Life Media, the parent company that created the Ashley Madison website, did not take the site offline, The Impact Group released two more compressed files, one 9.7GB and the second 20GB. After the second data dump, Avid Life Media CEO Noel Biderman resigned, but the website remained functional.

Legal issues and global regulation:

Conflict of laws in cyberspace has become a major cause of concern for computer security community. Some of the main challenges and complaints about the antivirus industry are the lack of global web regulations, a global base of common rules to judge, and eventually punish, cyber crimes and cyber criminals. There is no global cyber law and cyber security treaty that can be invoked for enforcing global cyber security issues.

International legal issues of cyber attacks are complicated in nature. Even if an antivirus firm locates the cybercriminal behind the creation of a particular virus or piece of malware or form of cyber attack, often the local authorities cannot take action due to lack of laws under which to prosecute.[146][147] Authorship attribution for cyber crimes and cyber attacks is a major problem for all law enforcement agencies.

"Computer viruses switch from one country to another, from one jurisdiction to another – moving around the world, using the fact that we don't have the capability to globally police operations like this. So the Internet is as if someone [had] given free plane tickets to all the online criminals of the world."[146] Use of dynamic DNS, fast flux and bullet proof servers have added own complexities to this situation.

Role of government:

The role of the government is to make regulations to force companies and organizations to protect their systems, infrastructure and information from any cyberattacks, but also to protect its own national infrastructure such as the national power-grid.[148]

Government's regulatory role in cyberspace is complicated. For some, cyberspace was seen virtual space that was to remain free of government intervention, as can be seen in many of today's libertarian blockchain and bitcoin discussions.[149]

Many government officials and experts think that the government should do more and that there is a crucial need for improved regulation, mainly due to the failure of the private sector to solve efficiently the cybersecurity problem. R. Clarke said during a panel discussion at the RSA Security Conference in San Francisco, he believes that the "industry only responds when you threaten regulation. If the industry doesn't respond (to the threat), you have to follow through."[150] On the other hand, executives from the private sector agree that improvements are necessary, but think that the government intervention would affect their ability to innovate efficiently. Daniel R. McCarthy analyzed this public-private partnership in cybersecurity and reflected on the role of cybersecurity in the broader constitution of political order.[151]

International actions:

Many different teams and organisations exist, including:

- The Forum of Incident Response and Security Teams (FIRST) is the global association of CSIRTs.[152] The US-CERT, AT&T, Apple, Cisco, McAfee, Microsoft are all members of this international team.[153]
- The Council of Europe helps protect societies worldwide from the threat of cybercrime through the Convention on Cybercrime.[154]
- The purpose of the Messaging Anti-Abuse Working Group (MAAWG) is to bring the messaging industry together to work collaboratively and to successfully address the various forms of messaging abuse, such as spam, viruses, denial-of-service attacks and other messaging exploitations.[155] France Telecom, Facebook, AT&T, Apple, Cisco, Sprint are some of the members of the MAAWG.[156]
- ENISA : The European Network and Information Security Agency (ENISA) is an agency of the European Union with the objective to improve network and information security in the European Union.

Europe:

On 14 April 2016 the European Parliament and Council of the European Union adopted The General Data Protection Regulation (GDPR) (EU) 2016/679. GDPR, which became enforceable beginning 25 May 2018, provides for data protection and privacy for all individuals within the European Union (EU) and the European Economic Area (EEA). GDPR requires that business processes that handle personal data be built with data protection by design and by default. GDPR also requires that certain organizations appoint a Data Protection Officer (DPO).

National actions:

Computer emergency response teams:

Main article: Computer emergency response team

Most countries have their own computer emergency response team to protect network security.

Canada:

On 3 October 2010, Public Safety Canada unveiled Canada's Cyber Security Strategy, following a Speech from the Throne commitment to boost the security of Canadian cyberspace.[157][158] The aim of the strategy is to strengthen Canada's "cyber systems and critical infrastructure sectors, support economic growth and protect Canadians as they connect to each other and to the world."[158] Three main pillars define the strategy: securing government systems, partnering to secure vital cyber systems outside the federal government, and helping Canadians to be secure online.[158] The strategy involves multiple departments and agencies across the Government of Canada.[159] The Cyber Incident Management Framework for Canada outlines these responsibilities, and provides a plan for coordinated response between government and other partners in the event of a cyber incident.[160] The Action Plan 2010–2015 for Canada's Cyber Security Strategy outlines the ongoing implementation of the strategy.[161]

Public Safety Canada's Canadian Cyber Incident Response Centre (CCIRC) is responsible for mitigating and responding to threats to Canada's critical infrastructure and cyber systems. The CCIRC provides support to mitigate cyber threats, technical support to respond and recover from targeted cyber attacks, and provides online tools for members of Canada's critical infrastructure sectors.[162] The CCIRC posts regular cyber security bulletins on the Public Safety Canada website.[163] The CCIRC also operates an online reporting tool where individuals and organizations can report a cyber incident.[164] Canada's Cyber Security Strategy is part of a larger, integrated approach to critical infrastructure protection, and functions as a counterpart document to the National Strategy and Action Plan for Critical Infrastructure.[159]

On 27 September 2010, Public Safety Canada partnered with STOP.THINK.CONNECT, a coalition of non-profit, private sector, and government organizations dedicated to informing the general public on how to protect themselves online.[165] On 4 February 2014, the Government of Canada launched the Cyber Security Cooperation Program.[166] The program is a $1.5 million five-year initiative aimed at improving Canada's cyber systems through grants and contributions to projects in support of this objective.[167] Public Safety Canada aims to begin an evaluation of Canada's Cyber Security Strategy in early 2015.[159] Public Safety Canada administers and routinely updates the GetCyberSafe portal for Canadian citizens, and carries out Cyber Security Awareness Month during October.[168]

China:

China's Central Leading Group for Internet Security and Informatization (Chinese: 中央网络安全和信息化领导小组) was established on 27 February 2014. This Leading Small Group (LSG) of the Communist Party of China is headed by General Secretary Xi Jinping himself and is staffed with relevant Party and state decision-makers. The LSG was created to overcome the incoherent policies and overlapping responsibilities that characterized China's former cyberspace decision-making mechanisms. The LSG oversees policy-making in the economic, political, cultural, social and military fields as they relate to network security and IT strategy. This LSG also coordinates major policy initiatives in the international arena that promote norms and standards favored by the Chinese government and that emphasize the principle of national sovereignty in cyberspace.[169]

Germany:

Berlin starts National Cyber Defense Initiative: On 16 June 2011, the German Minister for Home Affairs, officially opened the new German NCAZ (National Center for Cyber Defense) Nationales Cyber-Abwehrzentrum located in Bonn. The NCAZ closely cooperates with BSI (Federal Office for Information Security) Bundesamt für Sicherheit in der Informationstechnik, BKA (Federal Police Organisation) Bundeskriminalamt (Deutschland), BND (Federal Intelligence Service) Bundesnachrichtendienst, MAD (Military Intelligence Service) Amt für den Militärischen Abschirmdienst and other national organisations in Germany taking care of national security aspects. According to the Minister the primary task of the new organization founded on 23 February 2011, is to detect and prevent attacks against the national infrastructure and mentioned incidents like Stuxnet.

India:

Some provisions for cyber security have been incorporated into rules framed under the Information Technology Act 2000.[170]

The National Cyber Security Policy 2013 is a policy framework by Ministry of Electronics and Information Technology (MeitY) which aims to protect the public and private infrastructure from cyber attacks, and safeguard "information, such as personal information (of web users), financial and banking information and sovereign data". CERT- In is the nodal agency which monitors the cyber threats in the country. The post of National Cyber Security Coordinator has also been created in the Prime Minister's Office (PMO).

The Indian Companies Act 2013 has also introduced cyber law and cyber security obligations on the part of Indian directors. Some provisions for cyber security have been incorporated into rules framed under the Information Technology Act 2000 Update in 2013.[171]

South Korea:

Following cyber attacks in the first half of 2013, when the government, news media, television station, and bank websites were compromised, the national government committed to the training of 5,000 new cybersecurity experts by 2017. The South Korean government blamed its northern counterpart for these attacks, as well as incidents that occurred in 2009, 2011,[172] and 2012, but Pyongyang denies the accusations.[173]

United States:

Legislation:

The 1986 18 U.S.C. § 1030, more commonly known as the Computer Fraud and Abuse Act is the key legislation. It prohibits unauthorized access or damage of "protected computers" as defined in 18 U.S.C. § 1030(e)(2).

Although various other measures have been proposed, such as the "Cybersecurity Act of 2010 – S. 773" in 2009, the "International Cybercrime Reporting and Cooperation Act – H.R.4962"[174] and "Protecting Cyberspace as a National Asset Act of 2010 – S.3480"[175] in 2010 – none of these has succeeded.

Executive order 13636 *Improving Critical Infrastructure Cybersecurity* was signed 12 February 2013.

Agencies:

The Department of Homeland Security has a dedicated division responsible for the response system, risk management program and requirements for cybersecurity in the United States called the National Cyber Security Division.[176][177] The division is home to US-CERT operations and the National Cyber Alert System.[177] The National Cybersecurity and Communications Integration Center brings together government organizations responsible for protecting computer networks and networked infrastructure.[178]

The third priority of the Federal Bureau of Investigation (FBI) is to: *"Protect the United States against cyber-based attacks and high-technology crimes"*,[179] and they, along with the National White Collar Crime Center (NW3C), and the Bureau of Justice Assistance (BJA) are part of the multi-agency task force, The Internet Crime Complaint Center, also known as IC3.[180]

In addition to its own specific duties, the FBI participates alongside non-profit organizations such as InfraGard.[181][182]

In the criminal division of the United States Department of Justice operates a section called the Computer Crime and Intellectual Property Section. The CCIPS is in charge of investigating computer crime and intellectual property crime and is specialized in the search and seizure of digital evidence in computers and networks.[183] In 2017, CCIPS published A Framework for a Vulnerability Disclosure Program for Online Systems to help organizations "clearly describe authorized vulnerability disclosure and discovery conduct, thereby substantially reducing the likelihood that such described activities will result in a civil or criminal violation of law under the Computer Fraud and Abuse Act (18 U.S.C. § 1030)."[184]

The United States Cyber Command, also known as USCYBERCOM, is tasked with the defense of specified Department of Defense information networks and ensures *"the security, integrity, and governance of government and military IT infrastructure and assets"*[185] It has no role in the protection of civilian networks.[186][187]

The U.S. Federal Communications Commission's role in cybersecurity is to strengthen the protection of critical communications infrastructure, to assist in maintaining the reliability of networks during disasters, to aid in swift recovery after, and to ensure that first responders have access to effective communications services.[188]

The Food and Drug Administration has issued guidance for medical devices,[189] and the National Highway Traffic Safety Administration[190] is concerned with automotive cybersecurity. After being criticized by the Government Accountability Office,[191] and following successful attacks on airports and claimed attacks on airplanes, the Federal Aviation Administration has devoted funding to securing systems on board the planes of private manufacturers, and the Aircraft Communications Addressing and Reporting System.[192] Concerns have also been raised about the future Next Generation Air Transportation System.[193]

Computer emergency readiness team:

"Computer emergency response team" is a name given to expert groups that handle computer security incidents. In the US, two distinct organization exist, although they do work closely together.

- US-CERT: part of the National Cyber Security Division of the United States Department of Homeland Security.[194]
- CERT/CC: created by the Defense Advanced Research Projects Agency (DARPA) and run by the Software Engineering Institute (SEI).

Modern warfare:

Main article: Cyberwarfare

There is growing concern that cyberspace will become the next theater of warfare. As Mark Clayton from the *Christian Science Monitor* described in an article titled "The New Cyber Arms Race":

In the future, wars will not just be fought by soldiers with guns or with planes that drop bombs. They will also be fought with the click of a mouse a half a world away that unleashes carefully weaponized computer programs that disrupt or destroy critical industries like utilities, transportation, communications, and energy. Such attacks could also disable military networks that control the movement of troops, the path of jet fighters, the command and control of warships.[195]

This has led to new terms such as *cyberwarfare* and *cyberterrorism*. The United States Cyber Command was created in 2009[196] and many other countries have similar forces.

Careers:

Cybersecurity is a fast-growing field of IT concerned with reducing organizations' risk of hack or data breach.[197] According to research from the Enterprise Strategy Group, 46% of organizations say that they have a "problematic shortage" of cybersecurity skills in 2016, up from 28% in 2015.[198] Commercial, government and non-governmental organizations all employ cybersecurity professionals. The fastest increases in demand for cybersecurity workers are in industries managing increasing volumes of consumer data such as finance, health care, and retail.[199] However, the use of the term "cybersecurity" is more prevalent in government job descriptions.[200]

Typical cyber security job titles and descriptions include:[201]

Security analyst:

Analyzes and assesses vulnerabilities in the infrastructure (software, hardware, networks), investigates using available tools and countermeasures to remedy the detected vulnerabilities, and recommends solutions and best practices. Analyzes and assesses damage to the data/infrastructure as a result of security incidents, examines available recovery tools and processes, and recommends solutions. Tests for compliance with security policies and procedures. May assist in the creation, implementation, or management of security solutions.

Security engineer:

Performs security monitoring, security and data/logs analysis, and forensic analysis, to detect security incidents, and mounts the incident response. Investigates and utilizes new technologies and processes to enhance security capabilities and implement improvements. May also review code or perform other security engineering methodologies.

Security architect:

Designs a security system or major components of a security system, and may head a security design team building a new security system.

Security administrator:

Installs and manages organization-wide security systems. May also take on some of the tasks of a security analyst in smaller organizations.

Chief Information Security Officer (CISO):

A high-level management position responsible for the entire information security division/staff. The position may include hands-on technical work.

Chief Security Officer (CSO):

A high-level management position responsible for the entire security division/staff. A newer position now deemed needed as security risks grow.

Security Consultant/Specialist/Intelligence:

Broad titles that encompass any one or all of the other roles or titles tasked with protecting computers, networks, software, data or information systems against viruses, worms, spyware, malware, intrusion detection, unauthorized access, denial-of-service attacks, and an ever increasing list of attacks by hackers acting as individuals or as part of organized crime or foreign governments.

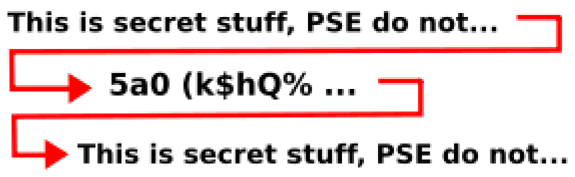
Student programs are also available to people interested in beginning a career in cybersecurity.[202][203] Meanwhile, a flexible and effective option for information security professionals of all experience levels to keep studying is online security training, including webcasts.[204][205][206] A wide range of certified courses are also available.[207]

In the United Kingdom, a nationwide set of cyber security forums, known as the U.K Cyber Security Forum, were established supported by the Government's cyber security strategy[208] in order to encourage start-ups and innovation and to address the skills gap[209] identified by the U.K Government.

Terminology:

The following terms used with regards to computer security are explained below:

- Access authorization restricts access to a computer to a group of users through the use of authentication systems. These systems can protect either the whole computer, such as through an interactive login screen, or individual services, such as a FTP server. There are many methods for identifying and authenticating users, such as passwords, identification cards, smart cards, and biometric systems.
- Anti-virus software consists of computer programs that attempt to identify, thwart, and eliminate computer viruses and other malicious software (malware).
- Applications are executable code, so general practice is to disallow users the power to install them; to install only those which are known to be reputable – and to reduce the attack surface by installing as few as possible. They are typically run with least privilege, with a robust process in place to identify, test and install any released security patches or updates for them.
- Authentication techniques can be used to ensure that communication end-points are who they say they are.]
- Automated theorem proving and other verification tools can enable critical algorithms and code used in secure systems to be mathematically proven to meet their specifications.
- Backups are one or more copies kept of important computer files. Typically, multiple copies will be kept at different locations so that if a copy is stolen or damaged, other copies will still exist.
- Capability and access control list techniques can be used to ensure privilege separation and mandatory access control. Capabilities vs. ACLs discusses their use.
- Chain of trust techniques can be used to attempt to ensure that all software loaded has been certified as authentic by the system's designers.
- Confidentiality is the nondisclosure of information except to another authorized person.[210]
- Cryptographic techniques can be used to defend data in transit between systems, reducing the probability that data exchanged between systems can be intercepted or modified.
- Cyberwarfare is an internet-based conflict that involves politically motivated attacks on information and information systems. Such attacks can, for example, disable official websites and networks, disrupt or disable essential services, steal or alter classified data, and cripple financial systems.
- Data integrity is the accuracy and consistency of stored data, indicated by an absence of any alteration in data between two updates of a data record.[211]


This is secret stuff, PSE do not...
5a0 (k$hQ% ...
This is secret stuff, PSE do not...
Cryptographic techniques involve transforming information, scrambling it so it becomes unreadable during transmission. The intended recipient can unscramble the message; ideally, eavesdroppers cannot.

- Encryption is used to protect the confidentiality of a message. Cryptographically secure ciphers are designed to make any practical attempt of breaking them infeasible. Symmetric-key ciphers are suitable for bulk encryption using shared keys, and public-key encryption using digital certificates can provide a practical solution for the problem of securely communicating when no key is shared in advance.

- Endpoint security software aids networks in preventing malware infection and data theft at network entry points made vulnerable by the prevalence of potentially infected devices such as laptops, mobile devices, and USB drives.[212]
- Firewalls serve as a gatekeeper system between networks, allowing only traffic that matches defined rules. They often include detailed logging, and may include intrusion detection and intrusion prevention features. They are near-universal between company local area networks and the Internet, but can also be used internally to impose traffic rules between networks if network segmentation is configured.
- A hacker is someone who seeks to breach defenses and exploit weaknesses in a computer system or network.
- Honey pots are computers that are intentionally left vulnerable to attack by crackers. They can be used to catch crackers and to identify their techniques.
- Intrusion-detection systems are devices or software applications that monitor networks or systems for malicious activity or policy violations.
- A microkernel is an approach to operating system design which has only the near-minimum amount of code running at the most privileged level – and runs other elements of the operating system such as device drivers, protocol stacks and file systems, in the safer, less privileged user space.
- Pinging. The standard "ping" application can be used to test if an IP address is in use. If it is, attackers may then try a port scan to detect which services are exposed.
- A port scan is used to probe an IP address for open ports with the purpose of identifying accessible network services.
- Social engineering is the use of deception to manipulate individuals to breach security.

Software engineering

From Wikipedia, the free encyclopedia

Software engineering is the application of engineering to the development of software in a systematic method.[1][2][3]

Definitions:

Notable definitions of software engineering include:

- "the systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software"—The Bureau of Labor Statistics—IEEE *Systems and software engineering - Vocabulary*[4]
- "The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software"—IEEE *Standard Glossary of Software Engineering Terminology*[5]
- "an engineering discipline that is concerned with all aspects of software production"—Ian Sommerville[6]
- "the establishment and use of sound engineering principles in order to economically obtain software that is reliable and works efficiently on real machines"—Fritz Bauer[7]

The term has also been used less formally:

- as the informal contemporary term for the broad range of activities that were formerly called computer programming and systems analysis;[8]
- as the broad term for all aspects of the *practice* of computer programming, as opposed to the *theory* of computer programming, which is called computer science;[9]

- as the term embodying the *advocacy* of a specific approach to computer programming, one that urges that it be treated as an engineering discipline rather than an art or a craft, and advocates the codification of recommended practices.[10]

History:

Main article: History of software engineering

When the first digital computers appeared in the early 1940s,[11] the instructions to make them operate were wired into the machine. Practitioners quickly realized that this design was not flexible and came up with the "stored program architecture" or von Neumann architecture. Thus the division between "hardware" and "software" began with abstraction being used to deal with the complexity of computing.

Programming languages started to appear in the early 1950s[12] and this was also another major step in abstraction. Major languages such as Fortran, ALGOL, and COBOL were released in the late 1950s to deal with scientific, algorithmic, and business problems respectively. David Parnas introduced the key concept of modularity and information hiding in 1972[13] to help programmers deal with the ever-increasing complexity of software systems.

The origins of the term "software engineering" have been attributed to various sources. The term "software engineering" appeared in a list of services offered by companies in the June 1965 issue of COMPUTERS and AUTOMATION and was used more formally in the August 1966 issue of Communications of the ACM (Volume 9, number 8) “letter to the ACM membership” by the ACM President Anthony A. Oettinger;[14], it is also associated with the title of a NATO conference in 1968 by Professor Friedrich L. Bauer, the first conference on software engineering.[15]. At the time there was perceived to be a "software crisis".[16][17][18] The 40th International Conference on Software Engineering (ICSE 2018) celebrates 50 years of "Software Engineering" with the Plenary Sessions' keynotes of Frederick Brooks[19] and Margaret Hamilton.[20]

In 1984, the Software Engineering Institute (SEI) was established as a federally funded research and development center headquartered on the campus of Carnegie Mellon University in Pittsburgh, Pennsylvania, United States. Watts Humphrey founded the SEI Software Process Program, aimed at understanding and managing the software engineering process. The Process Maturity Levels introduced would become the Capability Maturity Model Integration for Development(CMMi-DEV), which has defined how the US Government evaluates the abilities of a software development team.

Modern, generally accepted best-practices for software engineering have been collected by the ISO/IEC JTC 1/SC 7 subcommittee and published as the Software Engineering Body of Knowledge (SWEBOK).[21]

Subdisciplines:

Software engineering can be divided into sub-disciplines.[22] Some of them are:

- Software requirements[1][22] (or Requirements engineering): The elicitation, analysis, specification, and validation of requirements for software.
- Software design:[1][22] The process of defining the architecture, components, interfaces, and other characteristics of a system or component. It is also defined as the result of that process.
- Software construction:[1][22] The detailed creation of working, meaningful software through a combination of programming (aka coding), verification, unit testing, integration testing, and debugging.
- Software testing:[1][22] An empirical, technical investigation conducted to provide stakeholders with information about the quality of the product or service under test.

- Software maintenance:[1][22] The totality of activities required to provide cost-effective support to software.
- Software configuration management:[1][22] The identification of the configuration of a system at distinct points in time for the purpose of systematically controlling changes to the configuration, and maintaining the integrity and traceability of the configuration throughout the system life cycle. Modern processes use software versioning.
- Software engineering management:[1][22] The application of management activities—planning, coordinating, measuring, monitoring, controlling, and reporting—to ensure that the development and maintenance of software is systematic, disciplined, and quantified.
- Software development process:[1][22] The definition, implementation, assessment, measurement, management, change, and improvement of the software life cycle process itself.
- Software engineering models and methods[22] impose structure on software engineering with the goal of making that activity systematic, repeatable, and ultimately more success-oriented
- Software quality[22]
- Software engineering professional practice[22] is concerned with the knowledge, skills, and attitudes that software engineers must possess to practice software engineering in a professional, responsible, and ethical manner
- Software engineering economics[22] is about making decisions related to software engineering in a business context
- Computing foundations[22]
- Mathematical foundations[22]
- Engineering foundations[22]

Education:

Knowledge of computer programming is a prerequisite for becoming a software engineer. In 2004 the IEEE Computer Society produced the SWEBOK, which has been published as ISO/IEC Technical Report 1979:2004, describing the body of knowledge that they recommend to be mastered by a graduate software engineer with four years of experience.[23] Many software engineers enter the profession by obtaining a university degree or training at a vocational school. One standard international curriculum for undergraduate software engineering degrees was defined by the Joint Task Force on Computing Curricula of the IEEE Computer Society and the Association for Computing Machinery, and updated in 2014.[24] A number of universities have Software Engineering degree programs; as of 2010, there were 244 Campus Bachelor of Software Engineering programs, 70 Online programs, 230 Masters-level programs, 41 Doctorate-level programs, and 69 Certificate-level programs in the United States.[25]

In addition to university education, many companies sponsor internships for students wishing to pursue careers in information technology. These internships can introduce the student to interesting real-world tasks that typical software engineers encounter every day. Similar experience can be gained through military service in software engineering.

Profession:

Main articles: Software engineer and Software engineering professionalism

Legal requirements for the licensing or certification of professional software engineers vary around the world. In the UK, there is no licensing or legal requirement to assume or use the job title Software Engineer. In some areas of Canada, such as Alberta, British Columbia, Ontario,[26] and Quebec, software engineers can hold the Professional Engineer (P.Eng) designation and/or the Information Systems Professional (I.S.P.) designation. In Europe, Software Engineers can obtain the European Engineer (EUR ING) professional title.

The United States, since 2013, has offered an *NCEES* *Professional Engineer* exam for Software Engineering, thereby allowing Software Engineers to be licensed and recognized.[27] NCEES will end the exam after April 2019 due to lack of participation.[28] Mandatory licensing is currently still largely debated, and perceived as controversial. In some parts of the US such as Texas, the use of the term Engineer is regulated by law and reserved only for use by individuals who have a Professional Engineer license.

The IEEE Computer Society and the ACM, the two main US-based professional organizations of software engineering, publish guides to the profession of software engineering. The IEEE's *Guide to the Software Engineering Body of Knowledge - 2004 Version*, or SWEBOK, defines the field and describes the knowledge the IEEE expects a practicing software engineer to have. The most current SWEBOK v3 is an updated version and was released in 2014.[29] The IEEE also promulgates a "Software Engineering Code of Ethics".[30]

Employment:

In November 2004, the U. S. Bureau of Labor Statistics counted 760,840 software engineers holding jobs in the U.S.; in the same time period there were some 1.4 million practitioners employed in the U.S. in all other engineering disciplines combined.[31] Due to its relative newness as a field of study, formal education in software engineering is often taught as part of a computer science curriculum, and many software engineers hold computer science degrees and have no engineering background whatsoever.[32]

Many software engineers work as employees or contractors. Software engineers work with businesses, government agencies (civilian or military), and non-profit organizations. Some software engineers work for themselves as freelancers. Some organizations have specialists to perform each of the tasks in the software development process. Other organizations require software engineers to do many or all of them. In large projects, people may specialize in only one role. In small projects, people may fill several or all roles at the same time. Specializations include: in industry (analysts, architects, developers, testers, technical support, middleware analysts, managers) and in academia (educators, researchers).

Most software engineers and programmers work 40 hours a week, but about 15 percent of software engineers and 11 percent of programmers worked more than 50 hours a week in 2008. Injuries in these occupations are rare. However, like other workers who spend long periods in front of a computer terminal typing at a keyboard, engineers and programmers are susceptible to eyestrain, back discomfort, and hand and wrist problems such as carpal tunnel syndrome.[33]

Certification:

The Software Engineering Institute offers certifications on specific topics like security, process improvement and software architecture.[34] IBM, Microsoft and other companies also sponsor their own certification examinations. Many IT certification programs are oriented toward specific technologies, and managed by the vendors of these technologies.[35] These certification programs are tailored to the institutions that would employ people who use these technologies.

Broader certification of general software engineering skills is available through various professional societies. As of 2006, the IEEE had certified over 575 software professionals as a Certified Software Development Professional (CSDP).[36] In 2008 they added an entry-level certification known as the Certified Software Development Associate (CSDA).[37] The ACM had a professional certification program in the early 1980s,[*citation needed*] which was discontinued due to lack of interest. The ACM examined the possibility of professional certification of software engineers in the late 1990s, but eventually decided that such certification was inappropriate for the professional industrial practice of software engineering.[38]

In the U.K. the British Computer Society has developed a legally recognized professional certification called *Chartered IT Professional (CITP)*, available to fully qualified members (*MBCS*). Software engineers may be eligible for membership of the Institution of Engineering and Technology and so qualify for Chartered Engineer status. In Canada the Canadian Information Processing Society has developed a legally recognized professional certification called *Information Systems Professional (ISP)*.[39] In Ontario, Canada, Software Engineers who graduate from a *Canadian Engineering Accreditation Board (CEAB)* accredited program, successfully complete PEO's (*Professional Engineers Ontario*) Professional Practice Examination (PPE) and have at least 48 months of acceptable engineering experience are eligible to be licensed through the *Professional Engineers Ontario* and can become Professional Engineers P.Eng.[40] The PEO does not recognize any online or distance education however; and does not consider Computer Science programs to be equivalent to software engineering programs despite the tremendous overlap between the two. This has sparked controversy and a certification war. It has also held the number of P.Eng holders for the profession exceptionally low. The vast majority of working professionals in the field hold a degree in CS, not SE. Given the difficult certification path for holders of non-SE degrees, most never bother to pursue the license.

Impact of globalization:

The initial impact of outsourcing, and the relatively lower cost of international human resources in developing third world countries led to a massive migration of software development activities from corporations in North America and Europe to India and later: China, Russia, and other developing countries. This approach had some flaws, mainly the distance / timezone difference that prevented human interaction between clients and developers and the massive job transfer. This had a negative impact on many aspects of the software engineering profession. For example, some students in the developed world avoid education related to software engineering because of the fear of offshore outsourcing (importing software products or services from other countries) and of being displaced by foreign visa workers.[41] Although statistics do not currently show a threat to software engineering itself; a related career, computer programming does appear to have been affected.[42][43] Nevertheless, the ability to smartly leverage offshore and near-shore resources via the follow-the-sun workflow has improved the overall operational capability of many organizations.[44] When North Americans are leaving work, Asians are just arriving to work. When Asians are leaving work, Europeans are arriving to work. This provides a continuous ability to have human oversight on business-critical processes 24 hours per day, without paying overtime compensation or disrupting a key human resource, sleep patterns.

While global outsourcing has several advantages, global - and generally distributed - development can run into serious difficulties resulting from the distance between developers. This is due to the key elements of this type of distance that have been identified as geographical, temporal, cultural and communication (that includes the use of different languages and dialects of English in different locations).[45] Research has been carried out in the area of global software development over the last 15 years and an extensive body of relevant work published that highlights the benefits and problems associated with the complex activity. As with other aspects of software engineering research is ongoing in this and related areas.

Related fields:

Software engineering is a direct sub-field of engineering and has an overlap with computer science and management science [46]. It is also considered a part of overall systems engineering.

Computer Science:

This section **needs expansion**. You can help by adding to it. *(July 2017)*

In general, software engineering focuses more on techniques for the application of software development in industry,[47][48] while computer science focuses more on algorithms and theory.[49]

Controversy:

Criticism:

Software engineering sees its practitioners as individuals who follow well-defined engineering approaches to problem-solving. These approaches are specified in various software engineering books and research papers, always with the connotations of predictability, precision, mitigated risk and professionalism. This perspective has led to calls for licensing, certification and codified bodies of knowledge as mechanisms for spreading the engineering knowledge and maturing the field.

Software craftsmanship has been proposed by a body of software developers as an alternative that emphasizes the coding skills and accountability of the software developers themselves without professionalism or any prescribed curriculum leading to ad-hoc problem-solving (craftmanship) without engineering (lack of predictability, precision, missing risk mitigation, methods are informal and poorly defined). The Software Craftsmanship Manifesto extends the Agile Software Manifesto[50] and draws a metaphor between modern software development and the apprenticeship model of medieval Europe.

Software engineering extends engineering and draws on the engineering model, i.e. engineering process, engineering project management, engineering requirements, engineering design, engineering construction, and engineering validation. The concept is so new that it is rarely understood, and it is widely misinterpreted, including in software engineering textbooks, papers, and among the communities of programmers and crafters.

One of the core issues in software engineering is that its approaches are not empirical enough because a real-world validation of approaches is usually absent, or very limited and hence software engineering is often misinterpreted as feasible only in a "theoretical environment."

Edsger Dijkstra, the founder of many of the concepts used within software development today, refuted the idea of "software engineering" up until his death in 2002, arguing that those terms were poor analogies for what he called the "radical novelty" of computer science:

A number of these phenomena have been bundled under the name "Software Engineering". As economics is known as "The Miserable Science", software engineering should be known as "The Doomed Discipline", doomed because it cannot even approach its goal since its goal is self-contradictory. Software engineering, of course, presents itself as another worthy cause, but that is eyewash: if you carefully read its literature and analyse what its devotees actually do, you will discover that software engineering has accepted as its charter "How to program if you cannot."[51]

Artificial intelligence

From Wikipedia, the free encyclopedia

Jump to navigationJump to search

"AI" redirects here. For other uses, see AI (disambiguation) and Artificial intelligence (disambiguation).

Artificial intelligence (**AI**), sometimes called **machine intelligence**, is intelligence demonstrated by machines, in contrast to the **natural intelligence** displayed by humans and other animals. In computer science AI research is defined as the study of "intelligent agents": any device that perceives its environment and takes actions that maximize its chance of successfully achieving its goals.[1] Colloquially, the term "artificial intelligence" is applied when a machine mimics "cognitive" functions that humans associate with other human minds, such as "learning" and "problem solving".[2]

The scope of AI is disputed: as machines become increasingly capable, tasks considered as requiring "intelligence" are often removed from the definition, a phenomenon known as the AI effect, leading to the quip, "AI is whatever hasn't been done yet."[3] For instance, optical character recognition is frequently excluded from "artificial intelligence", having become a routine technology.[4] Modern machine capabilities generally classified as AI include successfully understanding human speech,[5] competing at the highest level in strategic game systems (such as chess and Go),[6] autonomously operating cars, and intelligent routing in content delivery networks and military simulations.

Artificial intelligence was founded as an academic discipline in 1956, and in the years since has experienced several waves of optimism,[7][8] followed by disappointment and the loss of funding (known as an "AI winter"),[9][10] followed by new approaches, success and renewed funding.[8][11] For most of its history, AI research has been divided into subfields that often fail to communicate with each other.[12] These sub-fields are based on technical considerations, such as particular goals (e.g. "robotics" or "machine learning"),[13] the use of particular tools ("logic" or artificial neural networks), or deep philosophical differences.[14][15][16] Subfields have also been based on social factors (particular institutions or the work of particular researchers).[12]

The traditional problems (or goals) of AI research include reasoning, knowledge representation, planning, learning, natural language processing, perception and the ability to move and manipulate objects.[13] General intelligence is among the field's long-term goals.[17] Approaches include statistical methods, computational intelligence, and traditional symbolic AI. Many tools are used in AI, including versions of search and mathematical optimization, artificial neural networks, and methods based on statistics, probability and economics. The AI field draws upon computer science, mathematics, psychology, linguistics, philosophy and many others.

The field was founded on the claim that human intelligence "can be so precisely described that a machine can be made to simulate it".[18] This raises philosophical arguments about the nature of the mind and the ethics of creating artificial beings endowed with human-like intelligence which are issues that have been explored by myth, fiction and philosophy since antiquity.[19] Some people also consider AI to be a danger to humanity if it progresses unabated.[20] Others believe that AI, unlike previous technological revolutions, will create a risk of mass unemployment.[21]

In the twenty-first century, AI techniques have experienced a resurgence following concurrent advances in computer power, large amounts of data, and theoretical understanding; and AI techniques have become an essential part of the technology industry, helping to solve many challenging problems in computer science, software engineering and operations research.[22][11]

History

Main articles: History of artificial intelligence and Timeline of artificial intelligence

Talos, an ancient mythical automaton with artificial intelligence

Thought-capable artificial beings appeared as storytelling devices in antiquity,[23] and have been common in fiction, as in Mary Shelley's *Frankenstein* or Karel Čapek's *R.U.R. (Rossum's Universal Robots)*.[24] These characters and their fates raised many of the same issues now discussed in the ethics of artificial intelligence.[19]

The study of mechanical or "formal" reasoning began with philosophers and mathematicians in antiquity. The study of mathematical logic led directly to Alan Turing's theory of computation, which suggested that a machine, by shuffling symbols as simple as "0" and "1", could simulate any conceivable act of mathematical deduction. This insight, that digital computers can simulate any process of formal reasoning, is known as the Church–Turing thesis.[25] Along with concurrent discoveries in neurobiology, information theory and cybernetics, this led researchers to consider the possibility of building an electronic brain. Turing proposed that "if a human could not distinguish between responses from a machine and a human, the machine could be considered “intelligent".[26] The first work that is now generally recognized as AI was McCullouch and Pitts' 1943 formal design for Turing-complete "artificial neurons".[27]

The field of AI research was born at a workshop at Dartmouth College in 1956.[28] Attendees Allen Newell (CMU), Herbert Simon (CMU), John McCarthy (MIT), Marvin Minsky (MIT) and Arthur Samuel (IBM) became the founders and leaders of AI research.[29] They and their students produced programs that the press described as "astonishing":[30] computers were learning checkers strategies (c. 1954)[31] (and by 1959 were reportedly playing better than the average human),[32] solving word problems in algebra, proving logical theorems (Logic Theorist, first run c. 1956) and speaking English.[33] By the middle of the 1960s, research in the U.S. was heavily funded by the Department of Defense[34] and laboratories had been established around the world.[35] AI's founders were optimistic about the future: Herbert Simon predicted, "machines will be capable, within twenty years, of doing any work a man can do". Marvin Minsky agreed, writing, "within a generation ... the problem of creating 'artificial intelligence' will substantially be solved".[7]

They failed to recognize the difficulty of some of the remaining tasks. Progress slowed and in 1974, in response to the criticism of Sir James Lighthill[36] and ongoing pressure from the US Congress to fund more productive projects, both the U.S. and British governments cut off exploratory research in AI. The next few years would later be called an "AI winter",[9] a period when obtaining funding for AI projects was difficult.

In the early 1980s, AI research was revived by the commercial success of expert systems,[37] a form of AI program that simulated the knowledge and analytical skills of human experts. By 1985, the market for AI had reached over a billion dollars. At the same time, Japan's fifth generation computer project inspired the U.S and British governments to restore funding for academic research.[8] However, beginning with the collapse of the Lisp Machine market in 1987, AI once again fell into disrepute, and a second, longer-lasting hiatus began.[10]

In the late 1990s and early 21st century, AI began to be used for logistics, data mining, medical diagnosis and other areas.[22] The success was due to increasing computational power (see Moore's law), greater emphasis on solving specific problems, new ties between AI and other fields (such as statistics, economics and mathematics), and a commitment by researchers to mathematical methods and scientific standards.[38] Deep Blue became the first computer chess-playing system to beat a reigning world chess champion, Garry Kasparov on 11 May 1997.[39]

In 2011, a *Jeopardy!* quiz show exhibition match, IBM's question answering system, Watson, defeated the two greatest *Jeopardy!* champions, Brad Rutter and Ken Jennings, by a significant margin.[40] Faster computers, algorithmic improvements, and access to large amounts of data enabled advances in machine learning and perception; data-hungry deep learning methods started to dominate accuracy benchmarks around 2012.[41] The Kinect, which provides a 3D body–motion interface for the Xbox 360 and the Xbox One use algorithms that emerged from lengthy AI research[42] as do intelligent personal assistants in smartphones.[43] In March 2016, AlphaGo won 4 out of 5 games of Go in a match with Go champion Lee Sedol, becoming the first computer Go-playing system to beat a professional Go player without handicaps.[6][44] In the 2017 Future of Go Summit, AlphaGo won a three-game match with Ke Jie,[45] who at the time continuously held the world No. 1 ranking for two years.[46][47] This marked the completion of a significant milestone in the development of Artificial Intelligence as Go is an extremely complex game, more so than Chess.

According to Bloomberg's Jack Clark, 2015 was a landmark year for artificial intelligence, with the number of software projects that use AI within Google increased from a "sporadic usage" in 2012 to more than 2,700 projects. Clark also presents factual data indicating that error rates in image processing tasks have fallen significantly since 2011.[48] He attributes this to an increase in affordable neural networks, due to a rise in cloud computing infrastructure and to an increase in research tools and datasets.[11] Other cited examples include Microsoft's development of a Skype system that can automatically translate from one language to another and Facebook's system that can describe images to blind people.[48] In a 2017 survey, one in five companies reported they had "incorporated AI in some offerings or processes".[49][50]

Basics

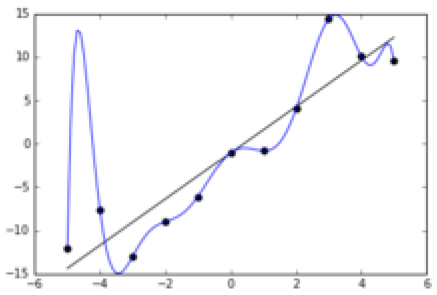
A typical AI perceives its environment and takes actions that maximize its chance of successfully achieving its goals.[1] An AI's intended goal function can be simple ("1 if the AI wins a game of Go, 0 otherwise") or complex ("Do actions mathematically similar to the actions that got you rewards in the past"). Goals can be explicitly defined, or can be induced. If the AI is programmed for "reinforcement learning", goals can be implicitly induced by rewarding some types of behavior and punishing others.[a] Alternatively, an evolutionary system can induce goals by using a "fitness function" to mutate and preferentially replicate high-scoring AI systems; this is similar to how animals evolved to innately desire certain goals such as finding food, or how dogs can be bred via artificial selection to possess desired traits.[51] Some AI systems, such as nearest-neighbor, instead reason by analogy; these systems are not generally given goals, except to the degree that goals are somehow implicit in their training data.[52] Such systems can still be benchmarked if the non-goal system is framed as a system whose "goal" is to successfully accomplish its narrow classification task.[53]

AI often revolves around the use of algorithms. An algorithm is a set of unambiguous instructions that a mechanical computer can execute.[b] A complex algorithm is often built on top of other, simpler, algorithms. A simple example of an algorithm is the following recipe for optimal play at tic-tac-toe:[54]

1. If someone has a "threat" (that is, two in a row), take the remaining square. Otherwise,
2. if a move "forks" to create two threats at once, play that move. Otherwise,
3. take the center square if it is free. Otherwise,
4. if your opponent has played in a corner, take the opposite corner. Otherwise,
5. take an empty corner if one exists. Otherwise,
6. take any empty square.

Many AI algorithms are capable of learning from data; they can enhance themselves by learning new heuristics (strategies, or "rules of thumb", that have worked well in the past), or can themselves write other algorithms. Some of the "learners" described below, including Bayesian networks, decision trees, and nearest-neighbor, could theoretically, if given infinite data, time, and memory, learn to approximate any function, including whatever combination of mathematical functions would best describe the entire world. These learners could therefore, in theory, derive all possible knowledge, by considering every possible hypothesis and matching it against the data. In practice, it is almost never possible to consider every possibility, because of the phenomenon of "combinatorial explosion", where the amount of time needed to solve a problem grows exponentially. Much of AI research involves figuring out how to identify and avoid considering broad swaths of possibilities that are unlikely to be fruitful.[55][56] For example, when viewing a map and looking for the shortest driving route from Denver to New York in the East, one can in most cases skip looking at any path through San Francisco or other areas far to the West; thus, an AI wielding an pathfinding algorithm like A* can avoid the combinatorial explosion that would ensue if every possible route had to be ponderously considered in turn.[57]

The earliest (and easiest to understand) approach to AI was symbolism (such as formal logic): "If an otherwise healthy adult has a fever, then they may have influenza". A second, more general, approach is Bayesian inference: "If the current patient has a fever, adjust the probability they have influenza in such-and-such way". The third major approach, extremely popular in routine business AI applications, are analogizers such as SVM and nearest-neighbor: "After examining the records of known past patients whose temperature, symptoms, age, and other factors mostly match the current patient, X% of those patients turned out to have influenza". A fourth approach is harder to intuitively understand, but is inspired by how the brain's machinery works: the artificial neural network approach uses artificial "neurons" that can learn by comparing itself to the desired output and altering the strengths of the connections between its internal neurons to "reinforce" connections that seemed to be useful. These four main approaches can overlap with each other and with evolutionary systems; for example, neural nets can learn to make inferences, to generalize, and to make analogies. Some systems implicitly or explicitly use multiple of these approaches, alongside many other AI and non-AI algorithms;[58] the best approach is often different depending on the problem.[59][60]



The blue line could be an example of overfitting a linear function due to random noise.

Learning algorithms work on the basis that strategies, algorithms, and inferences that worked well in the past are likely to continue working well in the future. These inferences can be obvious, such as "since the sun rose every morning for the last 10,000 days, it will probably rise tomorrow morning as well". They can be nuanced, such as "X% of families have geographically separate species with color variants, so there is an Y% chance that undiscovered black swans exist". Learners also work on the basis of "Occam's razor": The simplest theory that explains the data is the likeliest. Therefore, to be successful, a learner must be designed such that it prefers simpler theories to complex theories, except in cases where the complex theory is proven substantially better. Settling on a bad, overly complex theory gerrymandered to fit all the past training data is known as overfitting. Many systems attempt to reduce overfitting by rewarding a theory in accordance with how well it fits the data, but penalizing the theory in accordance with how complex the theory is.[61] Besides classic overfitting, learners can also disappoint by "learning the wrong lesson". A toy example is that an image classifier trained only on pictures of brown horses and black cats might conclude that all brown patches are likely to be horses.[62] A real-world example is that, unlike humans, current image classifiers don't determine the spatial relationship between components of the picture; instead, they learn

abstract patterns of pixels that humans are oblivious to, but that linearly correlate with images of certain types of real objects. Faintly superimposing such a pattern on a legitimate image results in an "adversarial" image that the system misclassifies.[c][63][64][65]

A self-driving car system may use a neural network to determine which parts of the picture seem to match previous training images of pedestrians, and then model those areas as slow-moving but somewhat unpredictable rectangular prisms that must be avoided.[66][67]

Compared with humans, existing AI lacks several features of human "commonsense reasoning"; most notably, humans have powerful mechanisms for reasoning about "naïve physics" such as space, time, and physical interactions. This enables even young children to easily make inferences like "If I roll this pen off a table, it will fall on the floor". Humans also have a powerful mechanism of "folk psychology" that helps them to interpret natural-language sentences such as "The city councilmen refused the demonstrators a permit because they advocated violence". (A generic AI has difficulty inferring whether the councilmen or the demonstrators are the ones alleged to be advocating violence.)[68][69][70] This lack of "common knowledge" means that AI often makes different mistakes than humans make, in ways that can seem incomprehensible. For example, existing self-driving cars cannot reason about the location nor the intentions of pedestrians in the exact way that humans do, and instead must use non-human modes of reasoning to avoid accidents.[71][72][73]

Problems

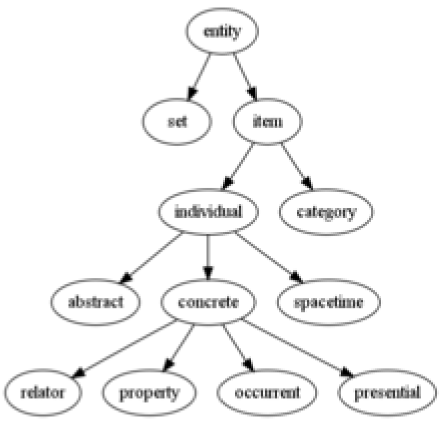
The overall research goal of artificial intelligence is to create technology that allows computers and machines to function in an intelligent manner. The general problem of simulating (or creating) intelligence has been broken down into sub-problems. These consist of particular traits or capabilities that researchers expect an intelligent system to display. The traits described below have received the most attention.[13]

Reasoning, problem solving

Early researchers developed algorithms that imitated step-by-step reasoning that humans use when they solve puzzles or make logical deductions.[74] By the late 1980s and 1990s, AI research had developed methods for dealing with uncertain or incomplete information, employing concepts from probability and economics.[75]

These algorithms proved to be insufficient for solving large reasoning problems, because they experienced a "combinatorial explosion": they became exponentially slower as the problems grew larger.[55] In fact, even humans rarely use the step-by-step deduction that early AI research was able to model. They solve most of their problems using fast, intuitive judgements.[76]

Knowledge representation



An ontology represents knowledge as a set of concepts within a domain and the relationships between those concepts.

Main articles: Knowledge representation and Commonsense knowledge

Knowledge representation[77] and knowledge engineering[78] are central to classical AI research. Some "expert systems" attempt to gather together explicit knowledge possessed by experts in some narrow domain. In addition, some projects attempt to gather the "commonsense knowledge" known to the average person into a database containing extensive knowledge about the world. Among the things a comprehensive commonsense knowledge base would contain are: objects, properties, categories and relations between objects;[79] situations, events, states and time;[80] causes and effects;[81] knowledge about knowledge (what we know about what other people know);[82] and many other, less well researched domains. A representation of "what exists" is an ontology: the set of objects, relations, concepts, and properties formally described so that software agents can interpret them. The semantics of these are captured as description logic concepts, roles, and individuals, and typically implemented as classes, properties, and individuals in the Web Ontology Language.[83] The most general ontologies are called upper ontologies, which attempt to provide a foundation for all other knowledge[84] by acting as mediators between domain ontologies that cover specific knowledge about a particular knowledge domain (field of interest or area of concern). Such formal knowledge representations can be used in content-based indexing and retrieval,[85] scene interpretation,[86] clinical decision support,[87] knowledge discovery (mining "interesting" and actionable inferences from large databases),[88] and other areas.[89]

Among the most difficult problems in knowledge representation are:

Default reasoning and the qualification problem

Many of the things people know take the form of "working assumptions". For example, if a bird comes up in conversation, people typically picture an animal that is fist sized, sings, and flies. None of these things are true about all birds. John McCarthy identified this problem in 1969[90] as the qualification problem: for any commonsense rule that AI researchers care to represent, there tend to be a huge number of exceptions. Almost nothing is simply true or false in the way that abstract logic requires. AI research has explored a number of solutions to this problem.[91]

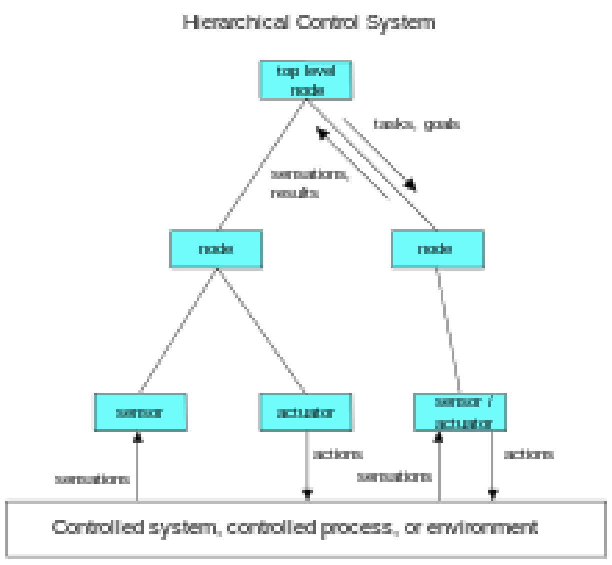
The breadth of commonsense knowledge

The number of atomic facts that the average person knows is very large. Research projects that attempt to build a complete knowledge base of commonsense knowledge (e.g., Cyc) require enormous amounts of laborious ontological engineering—they must be built, by hand, one complicated concept at a time.[92]

The subsymbolic form of some commonsense knowledge

Much of what people know is not represented as "facts" or "statements" that they could express verbally. For example, a chess master will avoid a particular chess position because it "feels too exposed"[93] or an art critic can take one look at a statue and realize that it is a fake.[94] These are non-conscious and sub-symbolic intuitions or tendencies in the human brain.[95] Knowledge like this informs, supports and provides a context for symbolic, conscious knowledge. As with the related problem of sub-symbolic reasoning, it is hoped that situated AI, computational intelligence, or statistical AI will provide ways to represent this kind of knowledge.[95]

Planning



A hierarchical control system is a form of control system in which a set of devices and governing software is arranged in a hierarchy.

Main article: Automated planning and scheduling

Intelligent agents must be able to set goals and achieve them.[96] They need a way to visualize the future—a representation of the state of the world and be able to make predictions about how their actions will change it—and be able to make choices that maximize the utility (or "value") of available choices.[97]

In classical planning problems, the agent can assume that it is the only system acting in the world, allowing the agent to be certain of the consequences of its actions.[98] However, if the agent is not the only actor, then it requires that the agent can reason under uncertainty. This calls for an agent that can not only assess its environment and make predictions, but also evaluate its predictions and adapt based on its assessment.[99]

Multi-agent planning uses the cooperation and competition of many agents to achieve a given goal. Emergent behavior such as this is used by evolutionary algorithms and swarm intelligence.[100]

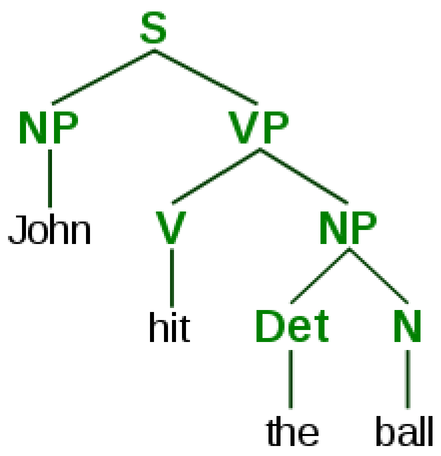
Learning

Main article: Machine learning

Machine learning, a fundamental concept of AI research since the field's inception,[101] is the study of computer algorithms that improve automatically through experience.[102][103]

Unsupervised learning is the ability to find patterns in a stream of input. Supervised learning includes both classification and numerical regression. Classification is used to determine what category something belongs in, after seeing a number of examples of things from several categories. Regression is the attempt to produce a function that describes the relationship between inputs and outputs and predicts how the outputs should change as the inputs change.[103] Both classifiers and regression learners can be viewed as "function approximators" trying to learn an unknown (possibly implicit) function; for example, a spam classifier can be viewed as learning a function that maps from the text of an email to one of two categories, "spam" or "not spam". Computational learning theory can assess learners by computational complexity, by sample complexity (how much data is required), or by other notions of optimization.[104] In reinforcement learning[105] the agent is rewarded for good responses and punished for bad ones. The agent uses this sequence of rewards and punishments to form a strategy for operating in its problem space.

Natural language processing



A parse tree represents the syntactic structure of a sentence according to some formal grammar.

Main article: Natural language processing

Natural language processing[106] (NLP) gives machines the ability to read and understand human language. A sufficiently powerful natural language processing system would enable natural-language user interfaces and the acquisition of knowledge directly from human-written sources, such as newswire texts. Some straightforward applications of natural language processing include information retrieval, text mining, question answering[107] and machine translation.[108] Many current approaches use word co-occurrence frequencies to construct syntactic representations of text. "Keyword spotting" strategies for search are popular and scalable but dumb; a search query for "dog" might only match documents with the literal word "dog" and miss a document with the word "poodle". "Lexical affinity" strategies use the occurrence of words such as "accident" to assess the sentiment of a document. Modern statistical NLP approaches can combine all these strategies as well as others, and often achieve acceptable accuracy at the page or paragraph level, but continue to lack the semantic understanding required to classify isolated sentences well. Besides the usual difficulties with encoding semantic commonsense knowledge, existing semantic NLP sometimes scales too poorly to be viable in business applications. Beyond semantic NLP, the ultimate goal of "narrative" NLP is to embody a full understanding of commonsense reasoning.[109]

Perception

Main articles: Machine perception, Computer vision, and Speech recognition

Feature detection (pictured: edge detection) helps AI compose informative abstract structures out of raw data.

Machine perception[110] is the ability to use input from sensors (such as cameras (visible spectrum or infrared), microphones, wireless signals, and active lidar, sonar, radar, and tactile sensors) to deduce aspects of the world. Applications include speech recognition,[111] facial recognition, and object recognition.[112] Computer vision is the ability to analyze visual input. Such input is usually ambiguous; a giant, fifty-meter-tall pedestrian far away may produce exactly the same pixels as a nearby normal-sized pedestrian, requiring the AI to judge the relative likelihood and reasonableness of different interpretations, for example by using its "object model" to assess that fifty-meter pedestrians do not exist.[113]

Motion and manipulation

Main article: Robotics

AI is heavily used in robotics.[114] Advanced robotic arms and other industrial robots, widely used in modern factories, can learn from experience how to move efficiently despite the presence of friction and gear slippage.[115] A modern mobile robot, when given a small, static, and visible environment, can easily determine its location and map its environment; however, dynamic environments, such as (in endoscopy) the interior of a patient's breathing body, pose a greater challenge. Motion planning is the process of breaking down a movement task into "primitives" such as individual joint movements. Such movement often involves compliant motion, a process where movement requires maintaining physical contact with an object. Moravec's paradox generalizes that low-level sensorimotor skills that humans take for granted are, counterintuitively, difficult to program into a robot; the paradox is named after Hans Moravec, who stated in 1988 that "it is comparatively easy to make computers exhibit adult level performance on intelligence tests or playing checkers, and difficult or impossible to give them the skills of a one-year-old when it comes to perception and mobility".[119] [120] this is attributed to the fact that, unlike checkers, physical dexterity has been a direct target of natural selection for millions of years.[121]

Social intelligence

Main article: Affective computing

Kismet, a robot with rudimentary social skills[122]

Moravec's paradox can be extended to many forms of social intelligence.[123][124] Distributed multi-agent coordination of autonomous vehicles remains a difficult problem.[125] Affective computing is an interdisciplinary umbrella that comprises systems which recognize, interpret, process, or simulate human affects.[126][127][128] Moderate successes related to affective computing include textual sentiment analysis and, more recently, multimodal affect analysis (see multimodal sentiment analysis), wherein AI classifies the affects displayed by a videotaped subject.[129]

In the long run, social skills and an understanding of human emotion and game theory would be valuable to a social agent. Being able to predict the actions of others by understanding their motives and emotional states would allow an agent to make better decisions. Some computer systems mimic human emotion and expressions to appear more sensitive to the emotional dynamics of human interaction, or to otherwise facilitate human–computer interaction.[130] Similarly, some virtual assistants are programmed to speak conversationally or even to banter humorously; this tends to give naïve users an unrealistic conception of how intelligent existing computer agents actually are.[131]

General intelligence

Historically, projects such as the Cyc knowledge base (1984–) and the massive Japanese Fifth Generation Computer Systems initiative (1982–1992) attempted to cover the breadth of human cognition. These early projects failed to escape the limitations of non-quantitative symbolic logic models and, in retrospect, greatly underestimated the difficulty of cross-domain AI. Nowadays, the vast majority of current AI researchers work instead on tractable "narrow AI" applications (such as medical diagnosis or automobile navigation).[132] Many researchers predict that such "narrow AI" work in different individual domains will eventually be incorporated into a machine with artificial general intelligence (AGI), combining most of the narrow skills mentioned in this article and at some point even exceeding human ability in most or all these areas.[17][133] Many advances have general, cross-domain significance. One high-profile example is that DeepMind in the 2010s developed a "generalized artificial intelligence" that could learn many diverse Atari games on its own, and later developed a variant of the system which succeeds at sequential learning.[134][135][136] Besides transfer learning,[137] hypothetical AGI breakthroughs could include the development of reflective architectures that can engage in decision-theoretic metareasoning, and figuring out how to "slurp up" a comprehensive knowledge base from the entire unstructured Web.[5] Some argue that some kind of (currently-undiscovered) conceptually straightforward, but mathematically difficult, "Master Algorithm" could lead to AGI.[138] Finally, a few "emergent" approaches look to simulating human intelligence extremely closely, and believe that anthropomorphic features like an artificial brain or simulated child development may someday reach a critical point where general intelligence emerges.[139][140]

Many of the problems in this article may also require general intelligence, if machines are to solve the problems as well as people do. For example, even specific straightforward tasks, like machine translation, require that a machine read and write in both languages (NLP), follow the author's argument (reason), know what is being talked about (knowledge), and faithfully reproduce the author's original intent (social intelligence). A problem like machine translation is considered "AI-complete", because all of these problems need to be solved simultaneously in order to reach human-level machine performance.

Approaches

There is no established unifying theory or paradigm that guides AI research. Researchers disagree about many issues.[141] A few of the most long standing questions that have remained unanswered are these: should artificial intelligence simulate natural intelligence by studying psychology or neurobiology? Or is human biology as irrelevant to AI research as bird biology is to aeronautical engineering?[14] Can intelligent behavior be described using simple, elegant principles (such as logic or optimization)? Or does it necessarily require solving a large number of completely unrelated problems?[15]

Cybernetics and brain simulation

Main articles: Cybernetics and Computational neuroscience

In the 1940s and 1950s, a number of researchers explored the connection between neurobiology, information theory, and cybernetics. Some of them built machines that used electronic networks to exhibit rudimentary intelligence, such as W. Grey Walter's turtles and the Johns Hopkins Beast. Many of these researchers gathered for meetings of the Teleological Society at Princeton University and the Ratio Club in England.[142] By 1960, this approach was largely abandoned, although elements of it would be revived in the 1980s.

Symbolic

Main article: Symbolic AI

When access to digital computers became possible in the middle 1950s, AI research began to explore the possibility that human intelligence could be reduced to symbol manipulation. The research was centered in three institutions: Carnegie Mellon University, Stanford and MIT, and as described below, each one developed its own style of research. John Haugeland named these symbolic approaches to AI "good old fashioned AI" or "GOFAI".[143] During the 1960s, symbolic approaches had achieved great success at simulating high-level thinking in small demonstration programs. Approaches based on cybernetics or artificial neural networks were abandoned or pushed into the background.[144] Researchers in the 1960s and the 1970s were convinced that symbolic approaches would eventually succeed in creating a machine with artificial general intelligence and considered this the goal of their field.

Cognitive simulation

Economist Herbert Simon and Allen Newell studied human problem-solving skills and attempted to formalize them, and their work laid the foundations of the field of artificial intelligence, as well as cognitive science, operations research and management science. Their research team used the results of psychological experiments to develop programs that simulated the techniques that people used to solve problems. This tradition, centered at Carnegie Mellon University would eventually culminate in the development of the Soar architecture in the middle 1980s.[145][146]

Logic-based

Unlike Simon and Newell, John McCarthy felt that machines did not need to simulate human thought, but should instead try to find the essence of abstract reasoning and problem-solving, regardless of whether people used the same algorithms.[14] His laboratory at Stanford (SAIL) focused on using formal logic to solve a wide variety of problems, including knowledge representation, planning and learning.[147] Logic was also the focus of the work at the University of Edinburgh and elsewhere in Europe which led to the development of the programming language Prolog and the science of logic programming.[148]

Anti-logic or scruffy

Researchers at MIT (such as Marvin Minsky and Seymour Papert)[149] found that solving difficult problems in vision and natural language processing required ad-hoc solutions – they argued that there was no simple and general principle (like logic) that would capture all the aspects of intelligent behavior. Roger Schank described their "anti-logic" approaches as "scruffy" (as opposed to the "neat" paradigms at CMU and Stanford).[15] Commonsense knowledge bases (such as Doug Lenat's Cyc) are an example of "scruffy" AI, since they must be built by hand, one complicated concept at a time.[150]

Knowledge-based

When computers with large memories became available around 1970, researchers from all three traditions began to build knowledge into AI applications.[151] This "knowledge revolution" led to the development and deployment of expert systems (introduced by Edward Feigenbaum), the first truly successful form of AI software.[37] The knowledge revolution was also driven by the realization that enormous amounts of knowledge would be required by many simple AI applications.

Sub-symbolic

By the 1980s, progress in symbolic AI seemed to stall and many believed that symbolic systems would never be able to imitate all the processes of human cognition, especially perception, robotics, learning and pattern recognition. A number of researchers began to look into "sub-symbolic" approaches to specific AI problems.[16] Sub-symbolic methods manage to approach intelligence without specific representations of knowledge.

Embodied intelligence

This includes embodied, situated, behavior-based, and nouvelle AI. Researchers from the related field of robotics, such as Rodney Brooks, rejected symbolic AI and focused on the basic engineering problems that would allow robots to move and survive.[152] Their work revived the non-symbolic viewpoint of the early cybernetics researchers of the 1950s and reintroduced the use of control theory in AI. This coincided with the development of the embodied mind thesis in the related field of cognitive science: the idea that aspects of the body (such as movement, perception and visualization) are required for higher intelligence.

Within developmental robotics, developmental learning approaches are elaborated upon to allow robots to accumulate repertoires of novel skills through autonomous self-exploration, social interaction with human teachers, and the use of guidance mechanisms (active learning, maturation, motor synergies, etc.).[153][154][155][156]

Computational intelligence and soft computing

Interest in neural networks and "connectionism" was revived by David Rumelhart and others in the middle of the 1980s.[157] Artificial neural networks are an example of soft computing --- they are solutions to problems which cannot be solved with complete logical certainty, and where an approximate solution is often sufficient. Other soft computing approaches to AI include fuzzy systems, evolutionary computation and many statistical tools. The application of soft computing to AI is studied collectively by the emerging discipline of computational intelligence.[158]

Statistical learning

Much of traditional GOFAI got bogged down on *ad hoc* patches to symbolic computation that worked on their own toy models but failed to generalize to real-world results. However, around the 1990s, AI researchers adopted sophisticated mathematical tools, such as hidden Markov models (HMM), information theory, and normative Bayesian decision theory to compare or to unify competing architectures. The shared mathematical language permitted a high level of collaboration with more established fields (like mathematics, economics or operations research).[d] Compared with GOFAI, new "statistical learning" techniques such as HMM and neural networks were gaining higher levels of accuracy in many practical domains such as data mining, without necessarily acquiring semantic understanding of the datasets. The increased successes with real-world data led to increasing emphasis on comparing different approaches against shared test data to see which approach performed best in a broader context than that provided by idiosyncratic toy models; AI research was becoming more scientific. Nowadays results of experiments are often rigorously measurable, and are sometimes (with difficulty) reproducible.[38][159] Different statistical learning techniques have different limitations; for example, basic HMM cannot model the infinite possible combinations of natural language.[160] Critics note that the shift from GOFAI to statistical learning is often also a shift away from Explainable AI. In AGI research, some scholars caution against over-reliance on statistical learning, and argue that continuing research into GOFAI will still be necessary to attain general intelligence.[161][162]

Integrating the approaches

Intelligent agent paradigm

An intelligent agent is a system that perceives its environment and takes actions which maximize its chances of success. The simplest intelligent agents are programs that solve specific problems. More complicated agents include human beings and organizations of human beings (such as firms). The paradigm allows researchers to directly compare or even combine different approaches to isolated problems, by asking which agent is best at maximizing a given "goal function". An agent that solves a specific problem can use any approach that works – some agents are symbolic and logical, some are sub-symbolic artificial neural networks and others may use new approaches. The paradigm also gives researchers a common language to communicate with other fields—such as decision theory and economics—that also use concepts of abstract agents. Building a complete agent requires researchers to address realistic problems of integration; for example, because sensory systems give uncertain information about the environment, planning systems must be able to function in the presence of uncertainty. The intelligent agent paradigm became widely accepted during the 1990s.[163]

Agent architectures and cognitive architectures

Researchers have designed systems to build intelligent systems out of interacting intelligent agents in a multi-agent system.[164] A hierarchical control system provides a bridge between sub-symbolic AI at its lowest, reactive levels and traditional symbolic AI at its highest levels, where relaxed time constraints permit planning and world modelling.[165] Some cognitive architectures are custom-built to solve a narrow problem; others, such as Soar, are designed to mimic human cognition and to provide insight into general intelligence. Modern extensions of Soar are hybrid intelligent systems that include both symbolic and sub-symbolic components.[166][167]

Tools

AI has developed a large number of tools to solve the most difficult problems in computer science. A few of the most general of these methods are discussed below.

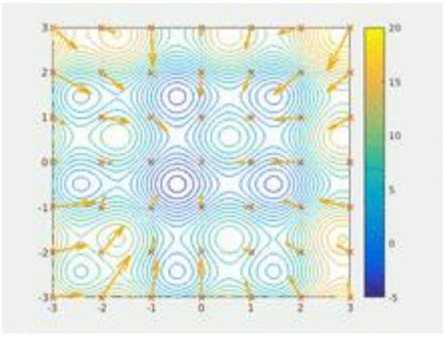
Search and optimization

Main articles: Search algorithm, Mathematical optimization, and Evolutionary computation

Many problems in AI can be solved in theory by intelligently searching through many possible solutions:[168] Reasoning can be reduced to performing a search. For example, logical proof can be viewed as searching for a path that leads from premises to conclusions, where each step is the application of an inference rule.[169] Planning algorithms search through trees of goals and subgoals, attempting to find a path to a target goal, a process called means-ends analysis.[170] Robotics algorithms for moving limbs and grasping objects use local searches in configuration space.[115] Many learning algorithms use search algorithms based on optimization.

Simple exhaustive searches[171] are rarely sufficient for most real-world problems: the search space (the number of places to search) quickly grows to astronomical numbers. The result is a search that is too slow or never completes. The solution, for many problems, is to use "heuristics" or "rules of thumb" that prioritize choices in favor of those that are more likely to reach a goal and to do so in a shorter number of steps. In some search methodologies heuristics can also serve to entirely eliminate some choices that are unlikely to lead to a goal (called "pruning the search tree"). Heuristics supply the program with a "best guess" for the path on which the solution lies.[172] Heuristics limit the search for solutions into a smaller sample size.[116]

A very different kind of search came to prominence in the 1990s, based on the mathematical theory of optimization. For many problems, it is possible to begin the search with some form of a guess and then refine the guess incrementally until no more refinements can be made. These algorithms can be visualized as blind hill climbing: we begin the search at a random point on the landscape, and then, by jumps or steps, we keep moving our guess uphill, until we reach the top. Other optimization algorithms are simulated annealing, beam search and random optimization.[173]



A particle swarm seeking the global minimum

Evolutionary computation uses a form of optimization search. For example, they may begin with a population of organisms (the guesses) and then allow them to mutate and recombine, selecting only the fittest to survive each generation (refining the guesses). Classic evolutionary algorithms include genetic algorithms, gene expression programming, and genetic programming.[174] Alternatively, distributed search processes can coordinate via swarm intelligence algorithms. Two popular swarm algorithms used in search are particle swarm optimization (inspired by bird flocking) and ant colony optimization (inspired by ant trails).[175][176]

Logic

Main articles: Logic programming and Automated reasoning

Logic[177] is used for knowledge representation and problem solving, but it can be applied to other problems as well. For example, the satplan algorithm uses logic for planning[178] and inductive logic programming is a method for learning.[179]

Several different forms of logic are used in AI research. Propositional logic[180] involves truth functions such as "or" and "not". First-order logic[181] adds quantifiers and predicates, and can express facts about objects, their properties, and their relations with each other. Fuzzy set theory assigns a "degree of truth" (between 0 and 1) to vague statements such as "Alice is old" (or rich, or tall, or hungry) that are too linguistically imprecise to be completely true or false. Fuzzy logic is successfully used in control systems to allow experts to contribute vague rules such as "if you are close to the destination station and moving fast, increase the train's brake pressure"; these vague rules can then be numerically refined within the system. Fuzzy logic fails to scale well in knowledge bases; many AI researchers question the validity of chaining fuzzy-logic inferences.[e][183][184]

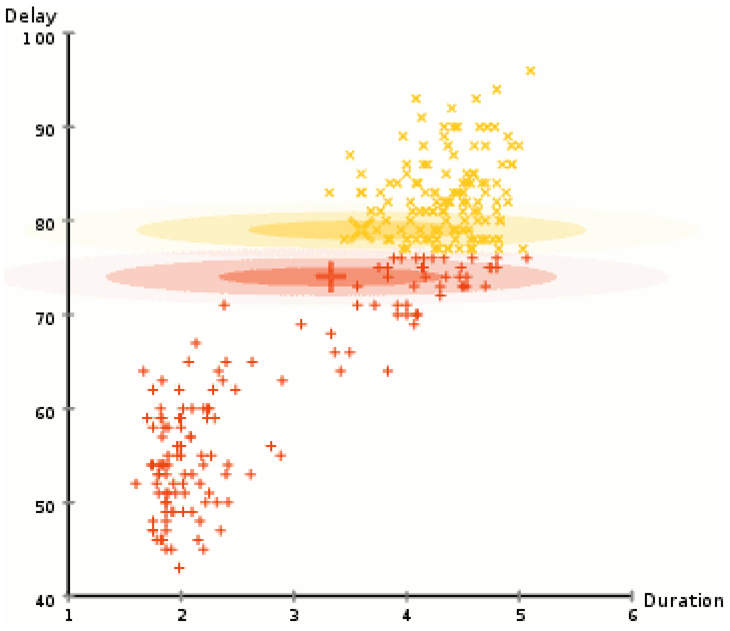
Default logics, non-monotonic logics and circumscription[91] are forms of logic designed to help with default reasoning and the qualification problem. Several extensions of logic have been designed to handle specific domains of knowledge, such as: description logics;[79] situation calculus, event calculus and fluent calculus (for representing events and time);[80] causal calculus;[81] belief calculus;[185] and modal logics.[82]

Overall, qualitiative symbolic logic is brittle and scales poorly in the presence of noise or other uncertainty. Exceptions to rules are numerous, and it is difficult for logical systems to function in the presence of contradictory rules.[186][187]

Probabilistic methods for uncertain reasoning

Main articles: Bayesian network, Hidden Markov model, Kalman filter, Particle filter, Decision theory, and Utility theory

Expectation-maximization clustering of Old Faithful eruption data starts from a random guess but then successfully converges on an accurate clustering of the two physically distinct modes of eruption.



Many problems in AI (in reasoning, planning, learning, perception, and robotics) require the agent to operate with incomplete or uncertain information. AI researchers have devised a number of powerful tools to solve these problems using methods from probability theory and economics.[188]

Bayesian networks[189] are a very general tool that can be used for a large number of problems: reasoning (using the Bayesian inference algorithm),[190] learning (using the expectation-maximization algorithm),[f][192] planning (using decision networks)[193] and perception (using dynamic Bayesian networks).[194] Probabilistic algorithms can also be used for filtering, prediction, smoothing and finding explanations for streams of data, helping perception systems to analyze processes that occur over time (e.g., hidden Markov models or Kalman filters).[194] Compared with symbolic logic, formal Bayesian inference is computationally expensive. For inference to be tractable, most observations must be conditionally independent of one another. Complicated graphs with diamonds or other "loops" (undirected cycles) can require a sophisticated method such as Markov Chain Monte Carlo, which spreads an ensemble of random walkers throughout the Bayesian network and attempts to converge to an assessment of the conditional probabilities. Bayesian networks are used on Xbox Live to rate and match players; wins and losses are "evidence" of how good a player is. AdSense uses a Bayesian network with over 300 million edges to learn which ads to serve.[195]

A key concept from the science of economics is "utility": a measure of how valuable something is to an intelligent agent. Precise mathematical tools have been developed that analyze how an agent can make choices and plan, using decision theory, decision analysis,[196] and information value theory.[97] These tools include models such as Markov decision processes,[197] dynamic decision networks,[194] game theory and mechanism design.[198]

Classifiers and statistical learning methods

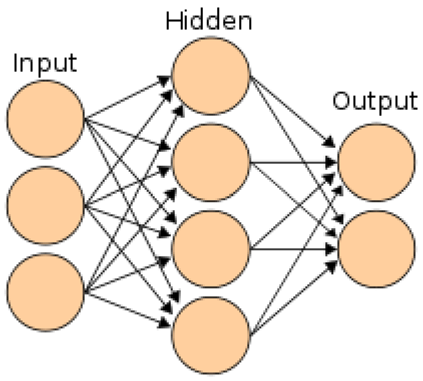
Main articles: Classifier (mathematics), Statistical classification, and Machine learning

The simplest AI applications can be divided into two types: classifiers ("if shiny then diamond") and controllers ("if shiny then pick up"). Controllers do, however, also classify conditions before inferring actions, and therefore classification forms a central part of many AI systems. Classifiers are functions that use pattern matching to determine a closest match. They can be tuned according to examples, making them very attractive for use in AI. These examples are known as observations or patterns. In supervised learning, each pattern belongs to a certain predefined class. A class can be seen as a decision that has to be made. All the observations combined with their class labels are known as a data set. When a new observation is received, that observation is classified based on previous experience.[199]

A classifier can be trained in various ways; there are many statistical and machine learning approaches. The decision tree[200] is perhaps the most widely used machine learning algorithm.[201] Other widely used classifiers are the neural network,[202] k-nearest neighbor algorithm,[g][204] kernel methods such as the support vector machine (SVM),[h][206] Gaussian mixture model,[207] and the extremely popular naive Bayes classifier.[i][209] Classifier performance depends greatly on the characteristics of the data to be classified, such as the dataset size, the dimensionality, and the level of noise. Model-based classifiers perform well if the assumed model is an extremely good fit for the actual data. Otherwise, if no matching model is available, and if accuracy (rather than speed or scalability) is the sole concern, conventional wisdom is that discriminative classifiers (especially SVM) tend to be more accurate than model-based classifiers such as "naive Bayes" on most practical data sets.[210][211]

Artificial neural networks

Main articles: Artificial neural network and Connectionism



A neural network is an interconnected group of nodes, akin to the vast network of neurons in the human brain.

Neural networks, or neural nets, were inspired by the architecture of neurons in the human brain. A simple "neuron" *N* accepts input from multiple other neurons, each of which, when activated (or "fired"), cast a weighted "vote" for or against whether neuron *N* should itself activate. Learning requires an algorithm to adjust these weights based on the training data; one simple algorithm (dubbed "fire together, wire together") is to increase the weight between two connected neurons when the activation of one triggers the successful activation of another. The net forms "concepts" that are distributed among a subnetwork of shared[j] neurons that tend to fire together; a concept meaning "leg" might be coupled with a subnetwork meaning "foot" that includes the sound for "foot". Neurons have a continuous spectrum of activation; in addition, neurons can process inputs in a nonlinear way rather than weighing straightforward votes. Modern neural nets can learn both continuous functions and, surprisingly, digital logical operations. Neural networks' early successes included predicting the stock market and (in 1995) a mostly self-driving car.[k][212] In the 2010s, advances in neural networks using deep learning thrust AI into widespread public consciousness and contributed to an enormous upshift in corporate AI spending; for example, AI-related M&A in 2017 was over 25 times as large as in 2015.[213][214]

The study of non-learning artificial neural networks[202] began in the decade before the field of AI research was founded, in the work of Walter Pitts and Warren McCullouch. Frank Rosenblatt invented the perceptron, a learning network with a single layer, similar to the old concept of linear regression. Early pioneers also include Alexey Grigorevich Ivakhnenko, Teuvo Kohonen, Stephen Grossberg, Kunihiko Fukushima, Christoph von der Malsburg, David Willshaw, Shun-Ichi Amari, Bernard Widrow, John Hopfield, Eduardo R. Caianiello, and others.

The main categories of networks are acyclic or feedforward neural networks (where the signal passes in only one direction) and recurrent neural networks (which allow feedback and short-term memories of previous input events). Among the most popular feedforward networks are perceptrons, multi-layer perceptrons and radial basis networks.[215] Neural networks can be applied to the problem of intelligent control (for robotics) or learning, using such techniques as Hebbian learning ("fire together, wire together"), GMDH or competitive learning.[216]

Today, neural networks are often trained by the backpropagation algorithm, which had been around since 1970 as the reverse mode of automatic differentiation published by Seppo Linnainmaa,[217][218] and was introduced to neural networks by Paul Werbos.[219][220][221]

Hierarchical temporal memory is an approach that models some of the structural and algorithmic properties of the neocortex.[222]

In short, most neural networks use some form of gradient descent on a hand-created neural topology. However, some research groups, such as Uber, argue that simple neuroevolution to mutate new neural network topologies and weights may be competitive with sophisticated gradient descent approaches. One advantage of neuroevolution is that it may be less prone to get caught in "dead ends".[223]

Deep feedforward neural networks

Main article: Deep learning

Deep learning is any artificial neural network that can learn a long chain of causal links. For example, a feedforward network with six hidden layers can learn a seven-link causal chain (six hidden layers + output layer) and has a "credit assignment path" (CAP) depth of seven. Many deep learning systems need to be able to learn chains ten or more causal links in length.[224] Deep learning has transformed many important subfields of artificial intelligence, including computer vision, speech recognition, natural language processing and others.[225][226][224]

According to one overview,[227] the expression "Deep Learning" was introduced to the Machine Learning community by Rina Dechter in 1986[228] and gained traction after Igor Aizenberg and colleagues introduced it to Artificial Neural Networks in 2000.[229] The first functional Deep Learning networks were published by Alexey Grigorevich Ivakhnenko and V. G. Lapa in 1965.[230][*page needed*] These networks are trained one layer at a time. Ivakhnenko's 1971 paper[231] describes the learning of a deep feedforward multilayer perceptron with eight layers, already much deeper than many later networks. In 2006, a publication by Geoffrey Hinton and Ruslan Salakhutdinov introduced another way of pre-training many-layered feedforward neural networks (FNNs) one layer at a time, treating each layer in turn as an unsupervised restricted Boltzmann machine, then using supervised backpropagation for fine-tuning.[232] Similar to shallow artificial neural networks, deep neural networks can model complex non-linear relationships. Over the last few years, advances in both machine learning algorithms and computer hardware have led to more efficient methods for training deep neural networks that contain many layers of non-linear hidden units and a very large output layer.[233]

Deep learning often uses convolutional neural networks (CNNs), whose origins can be traced back to the Neocognitron introduced by Kunihiko Fukushima in 1980.[234] In 1989, Yann LeCun and colleagues applied backpropagation to such an architecture. In the early 2000s, in an industrial application CNNs already processed an estimated 10% to 20% of all the checks written in the US.[235] Since 2011, fast implementations of CNNs on GPUs have won many visual pattern recognition competitions.[224]

CNNs with 12 convolutional layers were used in conjunction with reinforcement learning by Deepmind's "AlphaGo Lee", the program that beat a top Go champion in 2016.[236]

Deep recurrent neural networks

Main article: Recurrent neural networks

Early on, deep learning was also applied to sequence learning with recurrent neural networks (RNNs)[237] which are in theory Turing complete[238] and can run arbitrary programs to process arbitrary sequences of inputs. The depth of an RNN is unlimited and depends on the length of its input sequence; thus, an RNN is an example of deep learning.[224] RNNs can be trained by gradient descent[239][240][241] but suffer from the vanishing gradient problem.[225][242] In 1992, it was shown that unsupervised pre-training of a stack of recurrent neural networks can speed up subsequent supervised learning of deep sequential problems.[243]

Numerous researchers now use variants of a deep learning recurrent NN called the long short-term memory (LSTM) network published by Hochreiter & Schmidhuber in 1997.[244] LSTM is often trained by Connectionist Temporal Classification (CTC).[245] At Google, Microsoft and Baidu this approach has revolutionised speech recognition.[246][247][248] For example, in 2015, Google's speech recognition experienced a dramatic performance jump of 49% through CTC-trained LSTM, which is now available through Google Voice to billions of smartphone users.[249] Google also used LSTM to improve machine translation,[250] Language Modeling[251] and Multilingual Language Processing.[252] LSTM combined with CNNs also improved automatic image captioning[253] and a plethora of other applications.

Evaluating progress

Further information: Progress in artificial intelligence and Competitions and prizes in artificial intelligence

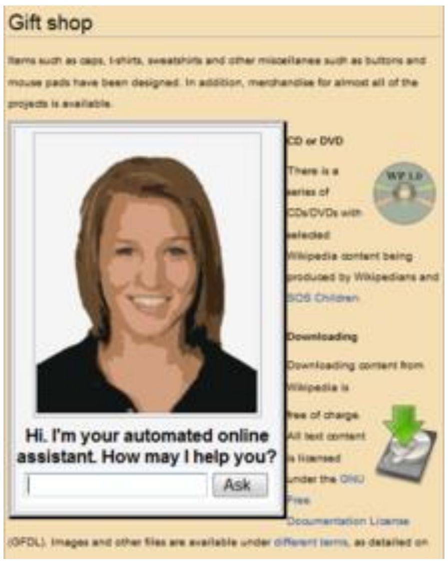
AI, like electricity or the steam engine, is a general purpose technology. There is no consensus on how to characterize which tasks AI tends to excel at.[254] While projects such as AlphaZero have succeeded in generating their own knowledge from scratch, many other machine learning projects require large training datasets.[255][256] Researcher Andrew Ng has suggested, as a "highly imperfect rule of thumb", that "almost anything a typical human can do with less than one second of mental thought, we can probably now or in the near future automate using AI."[257] Moravec's paradox suggests that AI lags humans at many tasks that the human brain has specifically evolved to perform well.[121]

Games provide a well-publicized benchmark for assessing rates of progress. AlphaGo around 2016 brought the era of classical board-game benchmarks to a close. Games of imperfect knowledge provide new challenges to AI in the area of game theory.[258][259] E-sports such as StarCraft continue to provide additional public benchmarks.[260][261] There are many competitions and prizes, such as the Imagenet Challenge, to promote research in artificial intelligence. The main areas of competition include general machine intelligence, conversational behavior, data-mining, robotic cars, and robot soccer as well as conventional games.[*citation needed*]

The "imitation game" (an interpretation of the 1950 Turing test that assesses whether a computer can imitate a human) is nowadays considered too exploitable to be a meaningful benchmark.[262] A derivative of the Turing test is the Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA). As the name implies, this helps to determine that a user is an actual person and not a computer posing as a human. In contrast to the standard Turing test, CAPTCHA is administered by a machine and targeted to a human as opposed to being administered by a human and targeted to a machine. A computer asks a user to complete a simple test then generates a grade for that test. Computers are unable to solve the problem, so correct solutions are deemed to be the result of a person taking the test. A common type of CAPTCHA is the test that requires the typing of distorted letters, numbers or symbols that appear in an image undecipherable by a computer.[263]

Proposed "universal intelligence" tests aim to compare how well machines, humans, and even non-human animals perform on problem sets that are generic as possible. At an extreme, the test suite can contain every possible problem, weighted by Kolmogorov complexity; unfortunately, these problem sets tend to be dominated by impoverished pattern-matching exercises where a tuned AI can easily exceed human performance levels.[264][265]

Applications



Gift shop

Items such as caps, t-shirts, sweatshirts and other miscellanea such as buttons and mouse pads have been designed. In addition, merchandise for almost all of the projects is available.

CD or DVD

There is a series of CDs/DVDs with selected Wikipedia content being produced by Wikipedians and SOS Children

Downloading

Downloading content from Wikipedia is free of charge. All text content is licensed under the GNU Free Documentation License (GFDL). Images and other files are available under different terms, as detailed on

Hi. I'm your automated online assistant. How may I help you?

An automated online assistant providing customer service on a web page – one of many very primitive applications of artificial intelligence

Main article: Applications of artificial intelligence

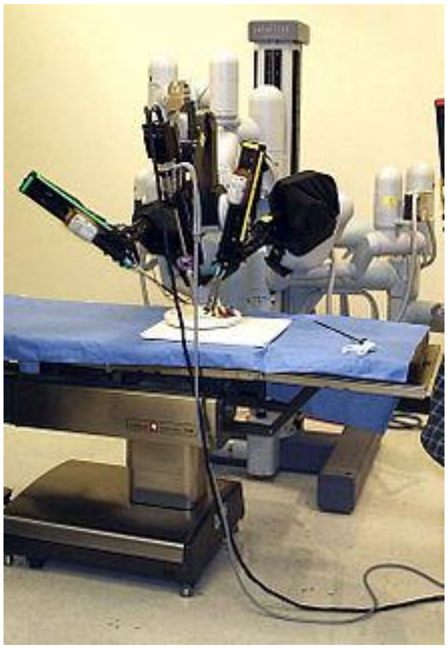
AI is relevant to any intellectual task.[266] Modern artificial intelligence techniques are pervasive and are too numerous to list here. Frequently, when a technique reaches mainstream use, it is no longer considered artificial intelligence; this phenomenon is described as the AI effect.[267]

High-profile examples of AI include autonomous vehicles (such as drones and self-driving cars), medical diagnosis, creating art (such as poetry), proving mathematical theorems, playing games (such as Chess or Go), search engines (such as Google search), online assistants (such as Siri), image recognition in photographs, spam filtering, prediction of judicial decisions[268] and targeting online advertisements.[266][269][270]

With social media sites overtaking TV as a source for news for young people and news organisations increasingly reliant on social media platforms for generating distribution,[271] major publishers now use artificial intelligence (AI) technology to post stories more effectively and generate higher volumes of traffic.[272]

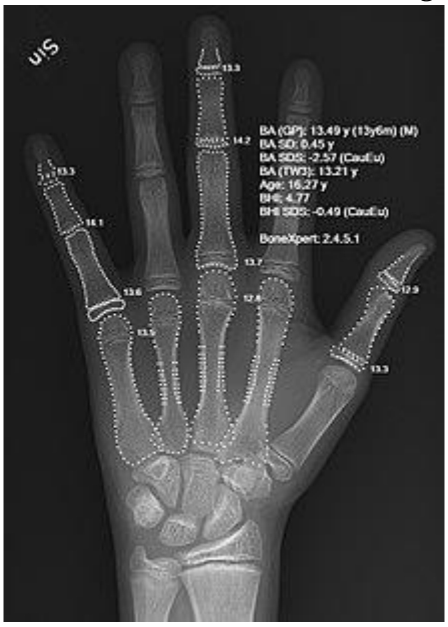
Healthcare

Main article: Artificial intelligence in healthcare



A patient-side surgical arm of Da Vinci Surgical System

AI is being applied to the high cost problem of dosage issues—where findings suggested that AI could save $16 billion. In 2016, a ground breaking study in California found that a mathematical formula developed with the help of AI correctly determined the accurate dose of immunosuppressant drugs to give to organ patients.[273]



X-ray of a hand, with automatic calculation of bone age by computer software

Artificial intelligence is breaking into the healthcare industry by assisting doctors. According to Bloomberg Technology, Microsoft has developed AI to help doctors find the right treatments for cancer.[274] There is a great amount of research and drugs developed relating to cancer. In detail, there are more than 800 medicines and vaccines to treat cancer. This negatively affects the doctors, because there are too many options to choose from, making it more difficult to choose the right drugs for the patients. Microsoft is working on a project to develop a machine called "Hanover". Its goal is to memorize all the papers necessary to cancer and help predict which combinations of drugs will be most effective for each patient. One project that is being worked on at the moment is fighting myeloid leukemia, a fatal cancer where the treatment has not improved in decades. Another study was reported to have found that artificial intelligence was as good as trained doctors in identifying skin cancers.[275] Another study is using artificial intelligence to try and monitor multiple high-risk patients, and this is done by asking each patient numerous questions based on data acquired from live doctor to patient interactions.[276]

According to CNN, a recent study by surgeons at the Children's National Medical Center in Washington successfully demonstrated surgery with an autonomous robot. The team supervised the robot while it performed soft-tissue surgery, stitching together a pig's bowel during open surgery, and doing so better than a human surgeon, the team claimed.[277] IBM has created its own artificial intelligence computer, the IBM Watson, which has beaten human intelligence (at some levels). Watson not only won at the game show *Jeopardy!* against former champions,[278] but was declared a hero after successfully diagnosing a woman who was suffering from leukemia.[279]

Automotive

Main article: driverless cars

Advancements in AI have contributed to the growth of the automotive industry through the creation and evolution of self-driving vehicles. As of 2016, there are over 30 companies utilizing AI into the creation of driverless cars. A few companies involved with AI include Tesla, Google, and Apple.[280]

Many components contribute to the functioning of self-driving cars. These vehicles incorporate systems such as braking, lane changing, collision prevention, navigation and mapping. Together, these systems, as well as high performance computers, are integrated into one complex vehicle.[281]

Recent developments in autonomous automobiles have made the innovation of self-driving trucks possible, though they are still in the testing phase. The UK government has passed legislation to begin testing of self-driving truck platoons in 2018.[282] Self-driving truck platoons are a fleet of self-driving trucks following the lead of one non-self-driving truck, so the truck platoons aren't entirely autonomous yet. Meanwhile, the Daimler, a German automobile corporation, is testing the Freightliner Inspiration which is a semi-autonomous truck that will only be used on the highway.[283]

One main factor that influences the ability for a driver-less automobile to function is mapping. In general, the vehicle would be pre-programmed with a map of the area being driven. This map would include data on the approximations of street light and curb heights in order for the vehicle to be aware of its surroundings. However, Google has been working on an algorithm with the purpose of eliminating the need for pre-programmed maps and instead, creating a device that would be able to adjust to a variety of new surroundings.[284] Some self-driving cars are not equipped with steering wheels or brake pedals, so there has also been research focused on creating an algorithm that is capable of maintaining a safe environment for the passengers in the vehicle through awareness of speed and driving conditions.[285]

Another factor that is influencing the ability for a driver-less automobile is the safety of the passenger. To make a driver-less automobile, engineers must program it to handle high-risk situations. These situations could include a head-on collision with pedestrians. The car's main goal should be to make a decision that would avoid hitting the pedestrians and saving the passengers in the car. But there is a possibility the car would need to make a decision that would put someone in danger. In other words, the car would need to decide to save the pedestrians or the passengers.[286] The programing of the car in these situations is crucial to a successful driver-less automobile.

Finance and economics

Financial institutions have long used artificial neural network systems to detect charges or claims outside of the norm, flagging these for human investigation. The use of AI in banking can be traced back to 1987 when Security Pacific National Bank in US set-up a Fraud Prevention Task force to counter the unauthorised use of debit cards. Programs like Kasisto and Moneystream are using AI in financial services.

Banks use artificial intelligence systems today to organize operations, maintain book-keeping, invest in stocks, and manage properties. AI can react to changes overnight or when business is not taking place.[287] In August 2001, robots beat humans in a simulated financial trading competition.[288] AI has also reduced fraud and financial crimes by monitoring behavioral patterns of users for any abnormal changes or anomalies.[289]

The use of AI machines in the market in applications such as online trading and decision making has changed major economic theories.[290] For example, AI based buying and selling platforms have changed the law of supply and demand in that it is now possible to easily estimate individualized demand and supply curves and thus individualized pricing. Furthermore, AI machines reduce information asymmetry in the market and thus making markets more efficient while reducing the volume of trades. Furthermore, AI in the markets limits the consequences of behavior in the markets again making markets more efficient. Other theories where AI has had impact include in rational choice, rational expectations, game theory, Lewis turning point, portfolio optimization and counterfactual thinking.

Video games

Main article: Artificial intelligence (video games)

In video games, artificial intelligence is routinely used to generate dynamic purposeful behavior in non-player characters (NPCs). In addition, well-understood AI techniques are routinely used for pathfinding. Some researchers consider NPC AI in games to be a "solved problem" for most production tasks. Games with more atypical AI include the AI director of *Left 4 Dead* (2008) and the neuroevolutionary training of platoons in *Supreme Commander 2* (2010).[291][292]

Military

Further information: Artificial intelligence arms race, Lethal autonomous weapon, and Unmanned combat aerial vehicle

Worldwide annual military spending on robotics rose from 5.1 billion USD in 2010 to 7.5 billion USD in 2015.[293][294] Military drones capable of autonomous action are widely considered a useful asset. In 2017, Vladimir Putin stated that "Whoever becomes the leader in (artificial intelligence) will become the ruler of the world".[295][296] Many artificial intelligence researchers seek to distance themselves from military applications of AI.[297]

Audit

For financial statements audit, AI makes continuous audit possible. AI tools could analyze many sets of different information immediately. The potential benefit would be the overall audit risk will be reduced, the level of assurance will be increased and the time duration of audit will be reduced.[298]

Advertising

A report by the Guardian newspaper in the UK in 2018 found that online gambling companies were using AI to predict the behavior of customers in order to target them with personalized promotions.[299] Developers of commercial AI platforms are also beginning to appeal more directly to casino operators, offering a range of existing and potential services to help them boost their profits and expand their customer base.[300]

Art

Artificial Intelligence has inspired numerous creative applications including its usage to produce visual art. The exhibition "Thinking Machines: Art and Design in the Computer Age, 1959-1989" at MoMA [301] provides a good overview of the historical applications of AI for art, architecture, and design. Recent exhibitions showcasing the usage of AI to produce art include the Google-sponsored benefit and auction at the Gray Area Foundation in San Francisco, where artists experimented with the deepdream algorithm [302] and the exhibition "Unhuman: Art in the Age of AI," which took place in Los Angeles and Frankfurt in the fall of 2017.[303][304] In the spring of 2018, the Association of Computing Machinery dedicated a special magazine issue to the subject of computers and art highlighting the role of machine learning in the arts.[305]

Philosophy and ethics

Main articles: Philosophy of artificial intelligence and Ethics of artificial intelligence

There are three philosophical questions related to AI:

1. Is artificial general intelligence possible? Can a machine solve any problem that a human being can solve using intelligence? Or are there hard limits to what a machine can accomplish?
2. Are intelligent machines dangerous? How can we ensure that machines behave ethically and that they are used ethically?
3. Can a machine have a mind, consciousness and mental states in exactly the same sense that human beings do? Can a machine be sentient, and thus deserve certain rights? Can a machine intentionally cause harm?

The limits of artificial general intelligence

Main articles: Philosophy of AI, Turing test, Physical symbol systems hypothesis, Dreyfus' critique of AI, The Emperor's New Mind, and AI effect

Can a machine be intelligent? Can it "think"?

Alan Turing's "polite convention"

We need not decide if a machine can "think"; we need only decide if a machine can act as intelligently as a human being. This approach to the philosophical problems associated with artificial intelligence forms the basis of the Turing test.[306]

The Dartmouth proposal

"Every aspect of learning or any other feature of intelligence can be so precisely described that a machine can be made to simulate it." This conjecture was printed in the proposal for the Dartmouth Conference of 1956, and represents the position of most working AI researchers.[307]

Newell and Simon's physical symbol system hypothesis

"A physical symbol system has the necessary and sufficient means of general intelligent action." Newell and Simon argue that intelligence consists of formal operations on symbols.[308] Hubert Dreyfus argued that, on the contrary, human expertise depends on unconscious instinct rather than conscious symbol manipulation and on having a "feel" for the situation rather than explicit symbolic knowledge. (See Dreyfus' critique of AI.)[309][310]

Gödelian arguments

Gödel himself,[311] John Lucas (in 1961) and Roger Penrose (in a more detailed argument from 1989 onwards) made highly technical arguments that human mathematicians can consistently see the truth of their own "Gödel statements" and therefore have computational abilities beyond that of mechanical Turing machines.[312] However, the modern consensus in the scientific and mathematical community is that these "Gödelian arguments" fail.[313][314][315]

The artificial brain argument

The brain can be simulated by machines and because brains are intelligent, simulated brains must also be intelligent; thus machines can be intelligent. Hans Moravec, Ray Kurzweil and others have argued that it is technologically feasible to copy the brain directly into hardware and software and that such a simulation will be essentially identical to the original.[139]

The AI effect

Machines are *already* intelligent, but observers have failed to recognize it. When Deep Blue beat Garry Kasparov in chess, the machine was acting intelligently. However, onlookers commonly discount the behavior of an artificial intelligence program by arguing that it is not "real" intelligence after all; thus "real" intelligence is whatever intelligent behavior people can do that machines still cannot. This is known as the AI Effect: "AI is whatever hasn't been done yet."

Potential harm

Widespread use of artificial intelligence could have unintended consequences that are dangerous or undesirable. Scientists from the Future of Life Institute, among others, described some short-term research goals to see how AI influences the economy, the laws and ethics that are involved with AI and how to minimize AI security risks. In the long-term, the scientists have proposed to continue optimizing function while minimizing possible security risks that come along with new technologies.[316]

Existential risk

Main article: Existential risk from artificial general intelligence

Physicist Stephen Hawking, Microsoft founder Bill Gates, and SpaceX founder Elon Musk have expressed concerns about the possibility that AI could evolve to the point that humans could not control it, with Hawking theorizing that this could "spell the end of the human race". [317] [318] [319]

The development of full artificial intelligence could spell the end of the human race. Once humans develop artificial intelligence, it will take off on its own and redesign itself at an ever-increasing rate. Humans, who are limited by slow biological evolution, couldn't compete and would be superseded.

— *Stephen Hawking*[320]

In his book *Superintelligence*, Nick Bostrom provides an argument that artificial intelligence will pose a threat to mankind. He argues that sufficiently intelligent AI, if it chooses actions based on achieving some goal, will exhibit convergent behavior such as acquiring resources or protecting itself from being shut down. If this AI's goals do not reflect humanity's – one example is an AI told to compute as many digits of pi as possible – it might harm humanity in order to acquire more resources or prevent itself from being shut down, ultimately to better achieve its goal.

Concern over risk from artificial intelligence has led to some high-profile donations and investments. A group of prominent tech titans including Peter Thiel, Amazon Web Services and Musk have committed $1billion to OpenAI a nonprofit company aimed at championing responsible AI development.[321] The opinion of experts within the field of artificial intelligence is mixed, with sizable fractions both concerned and unconcerned by risk from eventual superhumanly-capable AI.[322] In January 2015, Elon Musk donated ten million dollars to the Future of Life Institute to fund research on understanding AI decision making. The goal of the institute is to "grow wisdom with which we manage" the growing power of technology. Musk also funds companies developing artificial intelligence such as Google DeepMind and Vicarious to "just keep an eye on what's going on with artificial intelligence.[323] I think there is potentially a dangerous outcome there."[324][325]

For this danger to be realized, the hypothetical AI would have to overpower or out-think all of humanity, which a minority of experts argue is a possibility far enough in the future to not be worth researching.[326][327] Other counterarguments revolve around humans being either intrinsically or convergently valuable from the perspective of an artificial intelligence.[328]

Devaluation of humanity

Main article: Computer Power and Human Reason

Joseph Weizenbaum wrote that AI applications cannot, by definition, successfully simulate genuine human empathy and that the use of AI technology in fields such as customer service or psychotherapy[329] was deeply misguided. Weizenbaum was also bothered that AI researchers (and some philosophers) were willing to view the human mind as nothing more than a computer program (a position is now known as computationalism). To Weizenbaum these points suggest that AI research devalues human life.[330]

Decrease in demand for human labor

Further information: Technological unemployment § 21st century

The relationship between automation and employment is complicated. While automation eliminates old jobs, it also creates new jobs through micro-economic and macro-economic effects.[331] Unlike previous waves of automation, many middle-class jobs may be eliminated by artificial intelligence; *The Economist* states that "the worry that AI could do to white-collar jobs what steam power did to blue-collar ones during the Industrial Revolution" is "worth taking seriously".[332] Subjective estimates of the risk vary widely; for example, Michael Osborne and Carl Benedikt Frey estimate 47% of U.S. jobs are at "high risk" of potential automation, while an OECD report classifies only 9% of U.S. jobs as "high risk".[333][334][335] Jobs at extreme risk range from paralegals to fast food cooks, while job demand is likely to increase for care-related professions ranging from personal healthcare to the clergy.[336] Author Martin Ford and others go further and argue that a large number of jobs are routine, repetitive and (to an AI) predictable; Ford warns that these jobs may be automated in the next couple of decades, and that many of the new jobs may not be "accessible to people with average capability", even with retraining. Economists point out that in the past technology has tended to increase rather than reduce total employment, but acknowledge that "we're in uncharted territory" with AI.[21]

Autonomous weapons

See also: lethal autonomous weapon

Currently, 50+ countries are researching battlefield robots, including the United States, China, Russia, and the United Kingdom. Many people concerned about risk from superintelligent AI also want to limit the use of artificial soldiers and drones.[337]

Ethical machines

Machines with intelligence have the potential to use their intelligence to prevent harm and minimize the risks; they may have the ability to use ethical reasoning to better choose their actions in the world. Research in this area includes machine ethics, artificial moral agents, and friendly AI.

Artificial moral agents

Wendell Wallach introduced the concept of artificial moral agents (AMA) in his book *Moral Machines*[338] For Wallach, AMAs have become a part of the research landscape of artificial intelligence as guided by its two central questions which he identifies as "Does Humanity Want Computers Making Moral Decisions"[339] and "Can (Ro)bots Really Be Moral".[340] For Wallach the question is not centered on the issue of *whether* machines can demonstrate the equivalent of moral behavior in contrast to the *constraints* which society may place on the development of AMAs.[341]

Machine ethics

Main article: Machine ethics

The field of machine ethics is concerned with giving machines ethical principles, or a procedure for discovering a way to resolve the ethical dilemmas they might encounter, enabling them to function in an ethically responsible manner through their own ethical decision making.[342] The field was delineated in the AAAI Fall 2005 Symposium on Machine Ethics: "Past research concerning the relationship between technology and ethics has largely focused on responsible and irresponsible use of technology by human beings, with a few people being interested in how human beings ought to treat machines. In all cases, only human beings have engaged in ethical reasoning. The time has come for adding an ethical dimension to at least some machines. Recognition of the ethical ramifications of behavior involving machines, as well as recent and potential developments in machine autonomy, necessitate this. In contrast to computer hacking, software property issues, privacy issues and other topics normally ascribed to computer ethics, machine ethics is concerned with the behavior of machines towards human users and other machines. Research in machine ethics is key to alleviating concerns with autonomous systems—it could be argued that the notion of autonomous machines without such a dimension is at the root of all fear concerning machine intelligence. Further, investigation of machine ethics could enable the discovery of problems with current ethical theories, advancing our thinking about Ethics."[343] Machine ethics is sometimes referred to as machine morality, computational ethics or computational morality. A variety of perspectives of this nascent field can be found in the collected edition "Machine Ethics"[342] that stems from the AAAI Fall 2005 Symposium on Machine Ethics.[343]

Malevolent and friendly AI

Main article: Friendly AI

Political scientist Charles T. Rubin believes that AI can be neither designed nor guaranteed to be benevolent.[344] He argues that "any sufficiently advanced benevolence may be indistinguishable from malevolence." Humans should not assume machines or robots would treat us favorably because there is no *a priori* reason to believe that they would be sympathetic to our system of morality, which has evolved along with our particular biology (which AIs would not share). Hyper-intelligent software may not necessarily decide to support the continued existence of humanity and would be extremely difficult to stop. This topic has also recently begun to be discussed in academic publications as a real source of risks to civilization, humans, and planet Earth.

One proposal to deal with this is to ensure that the first generally intelligent AI is 'Friendly AI', and will then be able to control subsequently developed AIs. Some question whether this kind of check could really remain in place.

Leading AI researcher Rodney Brooks writes, "I think it is a mistake to be worrying about us developing malevolent AI anytime in the next few hundred years. I think the worry stems from a fundamental error in not distinguishing the difference between the very real recent advances in a particular aspect of AI, and the enormity and complexity of building sentient volitional intelligence."[345]

Machine consciousness, sentience and mind

Main article: Artificial consciousness

If an AI system replicates all key aspects of human intelligence, will that system also be sentient – will it have a mind which has conscious experiences? This question is closely related to the philosophical problem as to the nature of human consciousness, generally referred to as the hard problem of consciousness.

Consciousness

Main articles: Hard problem of consciousness and Theory of mind

Computationalism and functionalism

Main articles: Computationalism and Functionalism (philosophy of mind)

Computationalism is the position in the philosophy of mind that the human mind or the human brain (or both) is an information processing system and that thinking is a form of computing.[346] Computationalism argues that the relationship between mind and body is similar or identical to the relationship between software and hardware and thus may be a solution to the mind-body problem. This philosophical position was inspired by the work of AI researchers and cognitive scientists in the 1960s and was originally proposed by philosophers Jerry Fodor and Hilary Putnam.

Strong AI hypothesis

The philosophical position that John Searle has named "strong AI" states: "The appropriately programmed computer with the right inputs and outputs would thereby have a mind in exactly the same sense human beings have minds."[347] Searle counters this assertion with his Chinese room argument, which asks us to look *inside* the computer and try to find where the "mind" might be.[348]

Robot rights

If a machine can be created that has intelligence, could it also *feel*? If it can feel, does it have the same rights as a human? This issue, now known as "robot rights", is currently being considered by, for example, California's Institute for the Future, although many critics believe that the discussion is premature.[349] Some critics of transhumanism argue that any hypothetical robot rights would lie on a spectrum with animal rights and human rights.[350] The subject is profoundly discussed in the 2010 documentary film *Plug & Pray*.[351]

Superintelligence

Are there limits to how intelligent machines – or human-machine hybrids – can be? A superintelligence, hyperintelligence, or superhuman intelligence is a hypothetical agent that would possess intelligence far surpassing that of the brightest and most gifted human mind. ‘’Superintelligence’’ may also refer to the form or degree of intelligence possessed by such an agent.[133]

Technological singularity

If research into Strong AI produced sufficiently intelligent software, it might be able to reprogram and improve itself. The improved software would be even better at improving itself, leading to recursive self-improvement.[352] The new intelligence could thus increase exponentially and dramatically surpass humans. Science fiction writer Vernor Vinge named this scenario "singularity".[353] Technological singularity is when accelerating progress in technologies will cause a runaway effect wherein artificial intelligence will exceed human intellectual capacity and control, thus radically changing or even ending civilization. Because the capabilities of such an intelligence may be impossible to comprehend, the technological singularity is an occurrence beyond which events are unpredictable or even unfathomable.[353][133]

Ray Kurzweil has used Moore's law (which describes the relentless exponential improvement in digital technology) to calculate that desktop computers will have the same processing power as human brains by the year 2029, and predicts that the singularity will occur in 2045.[353]

Transhumanism

You awake one morning to find your brain has another lobe functioning. Invisible, this auxiliary lobe answers your questions with information beyond the realm of your own memory, suggests plausible courses of action, and asks questions that help bring out relevant facts. You quickly come to rely on the new lobe so much that you stop wondering how it works. You just use it. This is the dream of artificial intelligence.

Robot designer Hans Moravec, cyberneticist Kevin Warwick and inventor Ray Kurzweil have predicted that humans and machines will merge in the future into cyborgs that are more capable and powerful than either.[355] This idea, called transhumanism, which has roots in Aldous Huxley and Robert Ettinger.

Edward Fredkin argues that "artificial intelligence is the next stage in evolution", an idea first proposed by Samuel Butler's "Darwin among the Machines" (1863), and expanded upon by George Dyson in his book of the same name in 1998.[356]

In fiction

The word "robot" itself was coined by Karel Čapek in his 1921 play *R.U.R.*, the title standing for "Rossum's Universal Robots"

Thought-capable artificial beings appeared as storytelling devices since antiquity,[23] and have been a persistent theme in science fiction.

A common trope in these works began with Mary Shelley's *Frankenstein*, where a human creation becomes a threat to its masters. This includes such works as Arthur C. Clarke's and Stanley Kubrick's *2001: A Space Odyssey* (both 1968), with HAL 9000, the murderous computer in charge of the *Discovery One* spaceship, as well as *The Terminator* (1984) and *The Matrix* (1999). In contrast, the rare loyal robots such as Gort from *The Day the Earth Stood Still* (1951) and Bishop from *Aliens* (1986) are less prominent in popular culture.[357]

Isaac Asimov introduce the Three Laws of Robotics in many books and stories, most notably the "Multivac" series about a super-intelligent computer of the same name. Asimov's laws are often brought up during layman discussions of machine ethics;[358] while almost all artificial intelligence researchers are familiar with

Asimov's laws through popular culture, they generally consider the laws useless for many reasons, one of which is their ambiguity.[359]

Transhumanism (the merging of humans and machines) is explored in the manga *Ghost in the Shell* and the science-fiction series *Dune*. In the 1980s, artist Hajime Sorayama's Sexy Robots series were painted and published in Japan depicting the actual organic human form with lifelike muscular metallic skins and later "the Gynoids" book followed that was used by or influenced movie makers including George Lucas and other creatives. Sorayama never considered these organic robots to be real part of nature but always unnatural product of the human mind, a fantasy existing in the mind even when realized in actual form.

Several works use AI to force us to confront the fundamental of question of what makes us human, showing us artificial beings that have the ability to feel, and thus to suffer. This appears in Karel Čapek's "R.U.R.", the films "A.I. Artificial Intelligence" and "Ex Machina", as well as the novel *Do Androids Dream of Electric Sheep?*, by Philip K. Dick. Dick considers the idea that our understanding of human subjectivity is altered by technology created with artificial intelligence.[360]