

Later a control panel (plug board) added to his 1906 Type I Tabulator allowed it to be programmed for different jobs, and by the late 1940s, unit record equipment such as the IBM 602 and IBM 604, were programmed by control panels in a similar way, as were the first electronic computers. Machine code was the language of early programs, written in the instruction set of the particular machine, often in binary notation. New languages are generally designed around the syntax of a prior language with new functionality added, (for example C++ adds object-orientation to C, and Java adds memory management and bytecode to C++, but as a result, loses efficiency and the ability for low-level manipulation). High-level languages made the process of developing a program simpler and more understandable, and less bound to the underlying hardware. Many applications use a mix of several languages in their construction and use. Unreadable code often leads to bugs, inefficiencies, and duplicated code. Auxiliary tasks accompanying and related to programming include analyzing requirements, testing, debugging (investigating and fixing problems), implementation of build systems, and management of derived artifacts, such as programs' machine code. One approach popular for requirements analysis is Use Case analysis. Some languages are very popular for particular kinds of applications, while some languages are regularly used to write many different kinds of applications. Some text editors such as Emacs allow GDB to be invoked through them, to provide a visual environment. Trade-offs from this ideal involve finding enough programmers who know the language to build a team, the availability of compilers for that language, and the efficiency with which programs written in a given language execute. Compilers harnessed the power of computers to make programming easier by allowing programmers to specify calculations by entering a formula using infix notation. However, because an assembly language is little more than a different notation for a machine language, two machines with different instruction sets also have different assembly languages. Allen Downey, in his book *How To Think Like A Computer Scientist*, writes: Many computer languages provide a mechanism to call functions provided by shared libraries. High-level languages made the process of developing a program simpler and more understandable, and less bound to the underlying hardware. Proficient programming usually requires expertise in several different subjects, including knowledge of the application domain, details of programming languages and generic code libraries, specialized algorithms, and formal logic. A study found that a few simple readability transformations made code shorter and drastically reduced the time to understand it. Computer programming or coding is the composition of sequences of instructions, called programs, that computers can follow to perform tasks. Languages form an approximate spectrum from "low-level" to "high-level"; "low-level" languages are typically more machine-oriented and faster to execute, whereas "high-level" languages are more abstract and easier to use but execute less quickly. Allen Downey, in his book *How To Think Like A Computer Scientist*, writes: Many computer languages provide a mechanism to call functions provided by shared libraries. Compilers harnessed the power of computers to make programming easier by allowing programmers to specify calculations by entering a formula using infix notation. They are the building blocks for all software, from the simplest applications to the most sophisticated ones. Integrated development environments (IDEs) aim to integrate all such help. Many applications use a mix of several languages in their construction and use. The first step in most formal software development processes is requirements analysis, followed by testing to determine value modeling, implementation, and failure elimination (debugging).