

Many factors, having little or nothing to do with the ability of the computer to efficiently compile and execute the code, contribute to readability. High-level languages made the process of developing a program simpler and more understandable, and less bound to the underlying hardware. Ideally, the programming language best suited for the task at hand will be selected. This can be a non-trivial task, for example as with parallel processes or some unusual software bugs. However, with the concept of the stored-program computer introduced in 1949, both programs and data were stored and manipulated in the same way in computer memory. Programs were mostly entered using punched cards or paper tape. Different programming languages support different styles of programming (called programming paradigms). Methods of measuring programming language popularity include: counting the number of job advertisements that mention the language, the number of books sold and courses teaching the language (this overestimates the importance of newer languages), and estimates of the number of existing lines of code written in the language (this underestimates the number of users of business languages such as COBOL). By the late 1960s, data storage devices and computer terminals became inexpensive enough that programs could be created by typing directly into the computers. Compilers harnessed the power of computers to make programming easier by allowing programmers to specify calculations by entering a formula using infix notation. The first step in most formal software development processes is requirements analysis, followed by testing to determine value modeling, implementation, and failure elimination (debugging). Trial-and-error/divide-and-conquer is needed: the programmer will try to remove some parts of the original test case and check if the problem still exists. FORTRAN, the first widely used high-level language to have a functional implementation, came out in 1957, and many other languages were soon developed—in particular, COBOL aimed at commercial data processing, and Lisp for computer research. Provided the functions in a library follow the appropriate run-time conventions (e.g., method of passing arguments), then these functions may be written in any other language. This can be a non-trivial task, for example as with parallel processes or some unusual software bugs. For example, when a bug in a compiler can make it crash when parsing some large source file, a simplification of the test case that results in only few lines from the original source file can be sufficient to reproduce the same crash. Programming languages are essential for software development. 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. It is usually easier to code in "high-level" languages than in "low-level" ones. Assembly languages were soon developed that let the programmer specify instruction in a text format (e.g., ADD X, TOTAL), with abbreviations for each operation code and meaningful names for specifying addresses. As early as the 9th century, a programmable music sequencer was invented by the Persian Banu Musa brothers, who described an automated mechanical flute player in the Book of Ingenious Devices. Many applications use a mix of several languages in their construction and use. Use of a static code analysis tool can help detect some possible problems. The first compiler related tool, the A-0 System, was developed in 1952 by Grace Hopper, who also coined the term 'compiler'.