

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. 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. High-level languages made the process of developing a program simpler and more understandable, and less bound to the underlying hardware. 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. Some text editors such as Emacs allow GDB to be invoked through them, to provide a visual environment. Techniques like Code refactoring can enhance readability. 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). 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. In the 9th century, the Arab mathematician Al-Kindi described a cryptographic algorithm for deciphering encrypted code, in A Manuscript on Deciphering Cryptographic Messages. The choice of language used is subject to many considerations, such as company policy, suitability to task, availability of third-party packages, or individual preference. Many factors, having little or nothing to do with the ability of the computer to efficiently compile and execute the code, contribute to readability. Ideally, the programming language best suited for the task at hand will be selected. Following a consistent programming style often helps readability. Text editors were also developed that allowed changes and corrections to be made much more easily than with punched cards. 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. Also, specific user environment and usage history can make it difficult to reproduce the problem. It involves designing and implementing algorithms, step-by-step specifications of procedures, by writing code in one or more programming languages. For example, COBOL is still strong in corporate data centers often on large mainframe computers, Fortran in engineering applications, scripting languages in Web development, and C in embedded software. Also, specific user environment and usage history can make it difficult to reproduce the problem. After the bug is reproduced, the input of the program may need to be simplified to make it easier to debug. It is usually easier to code in "high-level" languages than in "low-level" ones. Debugging is a very important task in the software development process since having defects in a program can have significant consequences for its users. Machine code was the language of early programs, written in the instruction set of the particular machine, often in binary notation. The following properties are among the most important: In computer programming, readability refers to the ease with which a human reader can comprehend the purpose, control flow, and operation of source code.