

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. The first computer program is generally dated to 1843, when mathematician Ada Lovelace published an algorithm to calculate a sequence of Bernoulli numbers, intended to be carried out by Charles Babbage's Analytical Engine. One approach popular for requirements analysis is Use Case analysis. It involves designing and implementing algorithms, step-by-step specifications of procedures, by writing code in one or more programming languages. 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). 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. It is usually easier to code in "high-level" languages than in "low-level" ones. For this purpose, algorithms are classified into orders using so-called Big O notation, which expresses resource use, such as execution time or memory consumption, in terms of the size of an input. After the bug is reproduced, the input of the program may need to be simplified to make it easier to debug. 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 academic field and the engineering practice of computer programming are both largely concerned with discovering and implementing the most efficient algorithms for a given class of problems. 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. There are many approaches to the Software development process. Some of these factors include: The presentation aspects of this (such as indents, line breaks, color highlighting, and so on) are often handled by the source code editor, but the content aspects reflect the programmer's talent and skills. A similar technique used for database design is Entity-Relationship Modeling (ER Modeling). However, Charles Babbage had already written his first program for the Analytical Engine in 1837. Text editors were also developed that allowed changes and corrections to be made much more easily than with punched cards. In the 9th century, the Arab mathematician Al-Kindi described a cryptographic algorithm for deciphering encrypted code, in A Manuscript on Deciphering Cryptographic Messages. One approach popular for requirements analysis is Use Case analysis. Many factors, having little or nothing to do with the ability of the computer to efficiently compile and execute the code, contribute to readability. Implementation techniques include imperative languages (object-oriented or procedural), functional languages, and logic languages. A similar technique used for database design is Entity-Relationship Modeling (ER Modeling). 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. The academic field and the engineering practice of computer programming are both largely concerned with discovering and implementing the most efficient algorithms for a given class of problems. High-level languages made the process of developing a program simpler and more understandable, and less bound to the underlying hardware.