

Many applications use a mix of several languages in their construction and use. 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. 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. Many factors, having little or nothing to do with the ability of the computer to efficiently compile and execute the code, contribute to readability. It is usually easier to code in "high-level" languages than in "low-level" ones. 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. Readability is important because programmers spend the majority of their time reading, trying to understand, reusing and modifying existing source code, rather than writing new source code. One approach popular for requirements analysis is Use Case analysis. He gave the first description of cryptanalysis by frequency analysis, the earliest code-breaking algorithm. A similar technique used for database design is Entity-Relationship Modeling (ER Modeling). Unreadable code often leads to bugs, inefficiencies, and duplicated code. 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. 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. Programs were mostly entered using punched cards or paper tape. 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. Sometimes software development is known as software engineering, especially when it employs formal methods or follows an engineering design process. 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. 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. Some languages are more prone to some kinds of faults because their specification does not require compilers to perform as much checking as other languages. 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. A study found that a few simple readability transformations made code shorter and drastically reduced the time to understand it. Following a consistent programming style often helps readability. Unreadable code often leads to bugs, inefficiencies, and duplicated code. Various visual programming languages have also been developed with the intent to resolve readability concerns by adopting non-traditional approaches to code structure and display. 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.