

One approach popular for requirements analysis is Use Case analysis. High-level languages made the process of developing a program simpler and more understandable, and less bound to the underlying hardware. 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. Machine code was the language of early programs, written in the instruction set of the particular machine, often in binary notation. In 1801, the Jacquard loom could produce entirely different weaves by changing the "program" – a series of pasteboard cards with holes punched in them. It is usually easier to code in "high-level" languages than in "low-level" ones. Normally the first step in debugging is to attempt to reproduce the problem. 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. Programming languages are essential for software development. Sometimes software development is known as software engineering, especially when it employs formal methods or follows an engineering design process. 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. 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. Many factors, having little or nothing to do with the ability of the computer to efficiently compile and execute the code, contribute to readability. 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. A study found that a few simple readability transformations made code shorter and drastically reduced the time to understand it. By the late 1960s, data storage devices and computer terminals became inexpensive enough that programs could be created by typing directly into the computers. He gave the first description of cryptanalysis by frequency analysis, the earliest code-breaking algorithm. Following a consistent programming style often helps readability. It involves designing and implementing algorithms, step-by-step specifications of procedures, by writing code in one or more programming languages. This can be a non-trivial task, for example as with parallel processes or some unusual software bugs. Also, specific user environment and usage history can make it difficult to reproduce the problem. Following a consistent programming style often helps readability. In the 9th century, the Arab mathematician Al-Kindi described a cryptographic algorithm for deciphering encrypted code, in A Manuscript on Deciphering Cryptographic Messages.