One approach popular for requirements analysis is Use Case analysis. 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. 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. Languages form an approximate spectrum from "low-level" to "high-level"; "low-level" languages are typically more machine-oriented and faster to execute, whereas "high-level" languages are more abstract and easier to use but execute less quickly. Following a consistent programming style often helps 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. Computer programmers are those who write computer software. Techniques like Code refactoring can enhance readability. Debugging is a very important task in the software development process since having defects in a program can have significant consequences for its users. Debugging is a very important task in the software development process since having defects in a program can have significant consequences for its users. A study found that a few simple readability transformations made code shorter and drastically reduced the time to understand it. After the bug is reproduced, the input of the program may need to be simplified to make it easier to debug. However, readability is more than just programming style. It involves designing and implementing algorithms, step-by-step specifications of procedures, by writing code in one or more programming languages. Whatever the approach to development may be, the final program must satisfy some fundamental properties. 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. In the 9th century, the Arab mathematician Al-Kindi described a cryptographic algorithm for deciphering encrypted code, in A Manuscript on Deciphering Cryptographic Messages. Some languages are very popular for particular kinds of applications, while some languages are regularly used to write many different kinds of applications. A similar technique used for database design is Entity-Relationship Modeling (ER Modeling). 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. 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. 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. 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. Implementation techniques include imperative languages (object-oriented or procedural), functional languages, and logic languages.