A similar technique used for database design is Entity-Relationship Modeling (ER Modeling). The first compiler related tool, the A-0 System, was developed in 1952 by Grace Hopper, who also coined the term 'compiler'. Text editors were also developed that allowed changes and corrections to be made much more easily than with punched cards. 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. These compiled languages allow the programmer to write programs in terms that are syntactically richer, and more capable of abstracting the code, making it easy to target varying machine instruction sets via compilation declarations and heuristics. Trial-and-error/divide-and-conquer is needed: the programmer will try to remove some parts of the original test case and check if the problem still exists. Compilers harnessed the power of computers to make programming easier by allowing programmers to specify calculations by entering a formula using infix notation. Programmers typically use high-level programming languages that are more easily intelligible to humans than machine code, which is directly executed by the central processing unit. Some text editors such as Emacs allow GDB to be invoked through them, to provide a visual environment. It is usually easier to code in "high-level" languages than in "low-level" ones. After the bug is reproduced, the input of the program may need to be simplified to make it easier to debug. Popular modeling techniques include Object-Oriented Analysis and Design (OOAD) and Model-Driven Architecture (MDA). 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. Sometimes software development is known as software engineering, especially when it employs formal methods or follows an engineering design process. Techniques like Code refactoring can enhance readability. 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. In the 9th century, the Arab mathematician Al-Kindi described a cryptographic algorithm for deciphering encrypted code, in A Manuscript on Deciphering Cryptographic Messages. 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). 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. Whatever the approach to development may be, the final program must satisfy some fundamental properties. A study found that a few simple readability transformations made code shorter and drastically reduced the time to understand it. 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. One approach popular for requirements analysis is Use Case analysis. Compilers harnessed the power of computers to make programming easier by allowing programmers to specify calculations by entering a formula using infix notation. However, Charles Babbage had already written his first program for the Analytical Engine in 1837.