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. 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). Code-breaking algorithms have also existed for centuries. For example, COBOL is still strong in corporate data centers often on large mainframe computers, Fortran in engineering applications, scripting languages in Web development, and C in embedded software. 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. 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. 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. In the 1880s, Herman Hollerith invented the concept of storing data in machine-readable form. 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. For example, COBOL is still strong in corporate data centers often on large mainframe computers, Fortran in engineering applications, scripting languages in Web development, and C in embedded software. Use of a static code analysis tool can help detect some possible problems. 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. 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. 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. A similar technique used for database design is Entity-Relationship Modeling (ER Modeling). 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. Popular modeling techniques include Object-Oriented Analysis and Design (OOAD) and Model-Driven Architecture (MDA). 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. The first step in most formal software development processes is requirements analysis, followed by testing to determine value modeling, implementation, and failure elimination (debugging). Compilers harnessed the power of computers to make programming easier by allowing programmers to specify calculations by entering a formula using infix notation. In the 9th century, the Arab mathematician Al-Kindi described a cryptographic algorithm for deciphering encrypted code, in A Manuscript on Deciphering Cryptographic Messages. Use of a static code analysis tool can help detect some possible problems. Integrated development environments (IDEs) aim to integrate all such help.