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. 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. 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. 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. 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. 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). Assembly languages were soon developed that let the programmer specify instruction in a text format (e.g., ADD X, TOTAL), with abbreviations for each operation code and meaningful names for specifying addresses. Many applications use a mix of several languages in their construction and use. 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. Scripting and breakpointing is also part of this process. He gave the first description of cryptanalysis by frequency analysis, the earliest code-breaking algorithm. Implementation techniques include imperative languages (object-oriented or procedural), functional languages, and logic languages. 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). In 1801, the Jacquard loom could produce entirely different weaves by changing the "program" – a series of pasteboard cards with holes punched in them. Different programming languages support different styles of programming (called programming paradigms). However, readability is more than just programming style. Scripting and breakpointing is also part of this process. 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. In 1206, the Arab engineer Al-Jazari invented a programmable drum machine where a musical mechanical automaton could be made to play different rhythms and drum patterns, via pegs and cams. 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. 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. In the 1880s, Herman Hollerith invented the concept of storing data in machine-readable form. After the bug is reproduced, the input of the program may need to be simplified to make it easier to debug. 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.