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. Debugging is a very important task in the software development process since having defects in a program can have significant consequences for its users. One approach popular for requirements analysis is Use Case analysis. After the bug is reproduced, the input of the program may need to be simplified to make it easier to debug. Code-breaking algorithms have also existed for centuries. The first step in most formal software development processes is requirements analysis, followed by testing to determine value modeling, implementation, and failure elimination (debugging). Techniques like Code refactoring can enhance readability. Techniques like Code refactoring can enhance readability. 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. In 1801, the Jacquard loom could produce entirely different weaves by changing the "program" – a series of pasteboard cards with holes punched in them. Ideally, the programming language best suited for the task at hand will be selected. High-level languages made the process of developing a program simpler and more understandable, and less bound to the underlying hardware. Different programming languages support different styles of programming (called programming paradigms). 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. One approach popular for requirements analysis is Use Case analysis. Programs were mostly entered using punched cards or paper tape. Different programming languages support different styles of programming (called programming paradigms). 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. 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). Compilers harnessed the power of computers to make programming easier by allowing programmers to specify calculations by entering a formula using infix notation. 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. Text editors were also developed that allowed changes and corrections to be made much more easily than with punched cards. However, Charles Babbage had already written his first program for the Analytical Engine in 1837. Techniques like Code refactoring can enhance readability.