By the late 1960s, data storage devices and computer terminals became inexpensive enough that programs could be created by typing directly into the computers. In the 9th century, the Arab mathematician Al-Kindi described a cryptographic algorithm for deciphering encrypted code, in A Manuscript on Deciphering Cryptographic Messages. One approach popular for requirements analysis is Use Case analysis. In the 9th century, the Arab mathematician Al-Kindi described a cryptographic algorithm for deciphering encrypted code, in A Manuscript on Deciphering Cryptographic Messages. FORTRAN, the first widely used high-level language to have a functional implementation, came out in 1957, and many other languages were soon developed—in particular, COBOL aimed at commercial data processing, and Lisp for computer research. Unreadable code often leads to bugs, inefficiencies, and duplicated code. 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. 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. He gave the first description of cryptanalysis by frequency analysis, the earliest code-breaking algorithm. 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. Debugging is often done with IDEs. Standalone debuggers like GDB are also used, and these often provide less of a visual environment, usually using a command line. The first compiler related tool, the A-0 System, was developed in 1952 by Grace Hopper, who also coined the term 'compiler'. He gave the first description of cryptanalysis by frequency analysis, the earliest code-breaking algorithm. Machine code was the language of early programs, written in the instruction set of the particular machine, often in binary notation. 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. Following a consistent programming style often helps readability. Unreadable code often leads to bugs, inefficiencies, and duplicated 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. Machine code was the language of early programs, written in the instruction set of the particular machine, often in binary notation. After the bug is reproduced, the input of the program may need to be simplified to make it easier to debug. Different programming languages support different styles of programming (called programming paradigms). Different programming languages support different styles of programming (called programming paradigms). Integrated development environments (IDEs) aim to integrate all such help. It is usually easier to code in "high-level" languages than in "low-level" ones. Sometimes software development is known as software engineering, especially when it employs formal methods or follows an engineering design process.