By the late 1960s, data storage devices and computer terminals became inexpensive enough that programs could be created by typing directly into the computers. 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. Text editors were also developed that allowed changes and corrections to be made much more easily than with punched cards. 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. The first compiler related tool, the A-0 System, was developed in 1952 by Grace Hopper, who also coined the term 'compiler'. Sometimes software development is known as software engineering, especially when it employs formal methods or follows an engineering design process. 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. 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. The choice of language used is subject to many considerations, such as company policy, suitability to task, availability of third-party packages, or individual preference. One approach popular for requirements analysis is Use Case analysis. The choice of language used is subject to many considerations, such as company policy, suitability to task, availability of third-party packages, or individual preference. This can be a non-trivial task, for example as with parallel processes or some unusual software bugs. Text editors were also developed that allowed changes and corrections to be made much more easily than with punched cards. Compilers harnessed the power of computers to make programming easier by allowing programmers to specify calculations by entering a formula using infix notation. Implementation techniques include imperative languages (object-oriented or procedural), functional languages, and logic languages. When debugging the problem in a GUI, the programmer can try to skip some user interaction from the original problem description and check if remaining actions are sufficient for bugs to appear. 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. Techniques like Code refactoring can enhance readability. By the late 1960s, data storage devices and computer terminals became inexpensive enough that programs could be created by typing directly into the computers. Various visual programming languages have also been developed with the intent to resolve readability concerns by adopting non-traditional approaches to code structure and display. One approach popular for requirements analysis is Use Case analysis. It involves designing and implementing algorithms, step-by-step specifications of procedures, by writing code in one or more programming languages. Use of a static code analysis tool can help detect some possible problems. Following a consistent programming style often helps readability.