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..  
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.  
Ideally, the programming language best suited for the task at hand will be selected.  
 Programmable devices have existed for centuries.  
 Implementation techniques include imperative languages (object-oriented or procedural), functional languages, and logic languages.  
While these are sometimes considered programming, often the term software development is used for this larger overall process – with the terms programming, implementation, and coding reserved for the writing and editing of code per se.  
Techniques like Code refactoring can enhance readability.  
There exist a lot of different approaches for each of those tasks.  
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.  
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.  
One approach popular for requirements analysis is Use Case analysis.  
 Different programming languages support different styles of programming (called programming paradigms).  
 The academic field and the engineering practice of computer programming are both largely concerned with discovering and implementing the most efficient algorithms for a given class of problems.  
 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.  
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.