The first step in most formal software development processes is requirements analysis, followed by testing to determine value modeling, implementation, and failure elimination (debugging)..  
For this purpose, algorithms are classified into orders using so-called Big O notation, which expresses resource use, such as execution time or memory consumption, in terms of the size of an input.  
 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.  
Expert programmers are familiar with a variety of well-established algorithms and their respective complexities and use this knowledge to choose algorithms that are best suited to the circumstances.  
However, readability is more than just programming style.  
Use of a static code analysis tool can help detect some possible problems.  
 Machine code was the language of early programs, written in the instruction set of the particular machine, often in binary notation.  
Unreadable code often leads to bugs, inefficiencies, and duplicated code.  
 Readability is important because programmers spend the majority of their time reading, trying to understand, reusing and modifying existing source code, rather than writing new source code.  
  
 Implementation techniques include imperative languages (object-oriented or procedural), functional languages, and logic languages.  
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.  
 Some languages are very popular for particular kinds of applications, while some languages are regularly used to write many different kinds of applications.  
A study found that a few simple readability transformations made code shorter and drastically reduced the time to understand it.  
Integrated development environments (IDEs) aim to integrate all such help.