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..  
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.  
 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.  
 Different programming languages support different styles of programming (called programming paradigms).  
 These compiled languages allow the programmer to write programs in terms that are syntactically richer, and more capable of abstracting the code, making it easy to target varying machine instruction sets via compilation declarations and heuristics.  
Some of these factors include:  
 The presentation aspects of this (such as indents, line breaks, color highlighting, and so on) are often handled by the source code editor, but the content aspects reflect the programmer's talent and skills.  
Compilers harnessed the power of computers to make programming easier by allowing programmers to specify calculations by entering a formula using infix notation.  
Unreadable code often leads to bugs, inefficiencies, and duplicated code.  
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.  
 A similar technique used for database design is Entity-Relationship Modeling (ER Modeling).  
A study found that a few simple readability transformations made code shorter and drastically reduced the time to understand it.  
 The first step in most formal software development processes is requirements analysis, followed by testing to determine value modeling, implementation, and failure elimination (debugging).  
 Implementation techniques include imperative languages (object-oriented or procedural), functional languages, and logic languages.  
Also, specific user environment and usage history can make it difficult to reproduce the problem.  
 High-level languages made the process of developing a program simpler and more understandable, and less bound to the underlying hardware.