By the late 1960s, data storage devices and computer terminals became inexpensive enough that programs could be created by typing directly into the computers.  
Proficient programming thus usually requires expertise in several different subjects, including knowledge of the application domain, specialized algorithms, and formal logic.  
For example, when a bug in a compiler can make it crash when parsing some large source file, a simplification of the test case that results in only few lines from the original source file can be sufficient to reproduce the same crash.  
Also, those involved with software development may at times engage in reverse engineering, which is the practice of seeking to understand an existing program so as to re-implement its function in some way.  
Transpiling on the other hand, takes the source-code from a high-level programming language and converts it into bytecode.  
For example, when a bug in a compiler can make it crash when parsing some large source file, a simplification of the test case that results in only few lines from the original source file can be sufficient to reproduce the same crash.  
Relatedly, software engineering combines engineering techniques and principles with software development.  
It affects the aspects of quality above, including portability, usability and most importantly maintainability.  
It is usually easier to code in "high-level" languages than in "low-level" ones.  
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.  
Programming involves tasks such as analysis, generating algorithms, profiling algorithms' accuracy and resource consumption, and the implementation of algorithms (usually in a particular programming language, commonly referred to as coding).  
  
The first compiler related tool, the A-0 System, was developed in 1952 by Grace Hopper, who also coined the term 'compiler'.  
 New languages are generally designed around the syntax of a prior language with new functionality added, (for example C++ adds object-orientation to C, and Java adds memory management and bytecode to C++, but as a result, loses efficiency and the ability for low-level manipulation).  
The purpose of programming is to find a sequence of instructions that will automate the performance of a task (which can be as complex as an operating system) on a computer, often for solving a given problem.  
Scripting and breakpointing is also part of this process.