Compiling takes the source code from a low-level programming language and converts it into machine code.  
They are the building blocks for all software, from the simplest applications to the most sophisticated ones.  
One approach popular for requirements analysis is Use Case analysis.  
One approach popular for requirements analysis is Use Case analysis.  
They are the building blocks for all software, from the simplest applications to the most sophisticated ones.  
It is usually easier to code in "high-level" languages than in "low-level" ones.  
To produce machine code, the source code must either be compiled or transpiled.  
For example, COBOL is still strong in corporate data centers often on large mainframe computers, Fortran in engineering applications, scripting languages in Web development, and C in embedded software.  
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.  
Text editors were also developed that allowed changes and corrections to be made much more easily than with punched cards.  
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.  
 After the bug is reproduced, the input of the program may need to be simplified to make it easier to debug.  
To produce machine code, the source code must either be compiled or transpiled.  
However, because an assembly language is little more than a different notation for a machine language, two machines with different instruction sets also have different assembly languages.  
 High-level languages made the process of developing a program simpler and more understandable, and less bound to the underlying hardware.