Some languages are very popular for particular kinds of applications, while some languages are regularly used to write many different kinds of applications. By the late 1960s, data storage devices and computer terminals became inexpensive enough that programs could be created by typing directly into the computers. They are the building blocks for all software, from the simplest applications to the most sophisticated ones. 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. Integrated development environments (IDEs) aim to integrate all such help. 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. Different programming languages support different styles of programming (called programming paradigms). 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. Methods of measuring programming language popularity include: counting the number of job advertisements that mention the language, the number of books sold and courses teaching the language (this overestimates the importance of newer languages), and estimates of the number of existing lines of code written in the language (this underestimates the number of users of business languages such as COBOL). Normally the first step in debugging is to attempt to reproduce the problem. 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. One approach popular for requirements analysis is Use Case analysis. However, readability is more than just programming style. Following a consistent programming style often helps readability. Programmable devices have existed for centuries. 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. 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. 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. Provided the functions in a library follow the appropriate run-time conventions (e.g., method of passing arguments), then these functions may be written in any other language. Also, specific user environment and usage history can make it difficult to reproduce the problem. After the bug is reproduced, the input of the program may need to be simplified to make it easier to debug. 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. Normally the first step in debugging is to attempt to reproduce the problem. It is very difficult to determine what are the most popular modern programming languages. Whatever the approach to development may be, the final program must satisfy some fundamental properties.