

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. Languages form an approximate spectrum from "low-level" to "high-level"; "low-level" languages are typically more machine-oriented and faster to execute, whereas "high-level" languages are more abstract and easier to use but execute less quickly. Languages form an approximate spectrum from "low-level" to "high-level"; "low-level" languages are typically more machine-oriented and faster to execute, whereas "high-level" languages are more abstract and easier to use but execute less quickly. Popular modeling techniques include Object-Oriented Analysis and Design (OOAD) and Model-Driven Architecture (MDA). The following properties are among the most important: In computer programming, readability refers to the ease with which a human reader can comprehend the purpose, control flow, and operation of source code. The first step in most formal software development processes is requirements analysis, followed by testing to determine value modeling, implementation, and failure elimination (debugging). Popular modeling techniques include Object-Oriented Analysis and Design (OOAD) and Model-Driven Architecture (MDA). Some text editors such as Emacs allow GDB to be invoked through them, to provide a visual environment. 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. 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. Scripting and breakpointing is also part of this process. Techniques like Code refactoring can enhance readability. Unreadable code often leads to bugs, inefficiencies, and duplicated code. Machine code was the language of early programs, written in the instruction set of the particular machine, often in binary notation. After the bug is reproduced, the input of the program may need to be simplified to make it easier to debug. By the late 1960s, data storage devices and computer terminals became inexpensive enough that programs could be created by typing directly into the computers. Use of a static code analysis tool can help detect some possible problems. One approach popular for requirements analysis is Use Case analysis. Some languages are very popular for particular kinds of applications, while some languages are regularly used to write many different kinds of applications. 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. There exist a lot of different approaches for each of those tasks. Programs were mostly entered using punched cards or paper tape. 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. 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. It is usually easier to code in "high-level" languages than in "low-level" ones.