

Implementation techniques include imperative languages (object-oriented or procedural), functional languages, and logic languages. High-level languages made the process of developing a program simpler and more understandable, and less bound to the underlying hardware. 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. This can be a non-trivial task, for example as with parallel processes or some unusual software bugs. Various visual programming languages have also been developed with the intent to resolve readability concerns by adopting non-traditional approaches to code structure and display. Different programming languages support different styles of programming (called programming paradigms). It affects the aspects of quality above, including portability, usability and most importantly maintainability. However, Charles Babbage had already written his first program for the Analytical Engine in 1837. Unreadable code often leads to bugs, inefficiencies, and duplicated code. When debugging the problem in a GUI, the programmer can try to skip some user interaction from the original problem description and check if remaining actions are sufficient for bugs to appear. 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. Readability is important because programmers spend the majority of their time reading, trying to understand, reusing and modifying existing source code, rather than writing new source code. 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. Machine code was the language of early programs, written in the instruction set of the particular machine, often in binary notation. 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. 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. It involves designing and implementing algorithms, step-by-step specifications of procedures, by writing code in one or more programming languages. 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. 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. Some languages are more prone to some kinds of faults because their specification does not require compilers to perform as much checking as other languages. Implementation techniques include imperative languages (object-oriented or procedural), functional languages, and logic languages. Ideally, the programming language best suited for the task at hand will be selected. 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. This can be a non-trivial task, for example as with parallel processes or some unusual software bugs. 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.