

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. There exist a lot of different approaches for each of those tasks. As early as the 9th century, a programmable music sequencer was invented by the Persian Banu Musa brothers, who described an automated mechanical flute player in the Book of Ingenious Devices. While these are sometimes considered programming, often the term software development is used for this larger overall process – with the terms programming, implementation, and coding reserved for the writing and editing of code per se. 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). The first step in most formal software development processes is requirements analysis, followed by testing to determine value modeling, implementation, and failure elimination (debugging). 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). Programs were mostly entered using punched cards or paper tape. Debugging is often done with IDEs. Standalone debuggers like GDB are also used, and these often provide less of a visual environment, usually using a command line. 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. Sometimes software development is known as software engineering, especially when it employs formal methods or follows an engineering design process. 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. 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. 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. High-level languages made the process of developing a program simpler and more understandable, and less bound to the underlying hardware. It involves designing and implementing algorithms, step-by-step specifications of procedures, by writing code in one or more programming languages. It affects the aspects of quality above, including portability, usability and most importantly maintainability. It affects the aspects of quality above, including portability, usability and most importantly maintainability. 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. 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). 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 affects the aspects of quality above, including portability, usability and most importantly maintainability. A study found that a few simple readability transformations made code shorter and drastically reduced the time to understand it. Also, specific user environment and usage history can make it difficult to reproduce the problem.