

Proficient programming usually requires expertise in several different subjects, including knowledge of the application domain, details of programming languages and generic code libraries, specialized algorithms, and formal logic. The Unified Modeling Language (UML) is a notation used for both the OOAD and MDA. 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. 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. 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. It is usually easier to code in "high-level" languages than in "low-level" ones. 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. By the late 1960s, data storage devices and computer terminals became inexpensive enough that programs could be created by typing directly into the computers. The first computer program is generally dated to 1843, when mathematician Ada Lovelace published an algorithm to calculate a sequence of Bernoulli numbers, intended to be carried out by Charles Babbage's Analytical Engine. Also, specific user environment and usage history can make it difficult to reproduce the problem. Computer programming or coding is the composition of sequences of instructions, called programs, that computers can follow to perform tasks. Different programming languages support different styles of programming (called programming paradigms). 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. 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. Some languages are very popular for particular kinds of applications, while some languages are regularly used to write many different kinds of applications. Following a consistent programming style often helps readability. Programming languages are essential for software development. The first step in most formal software development processes is requirements analysis, followed by testing to determine value modeling, implementation, and failure elimination (debugging). Unreadable code often leads to bugs, inefficiencies, and duplicated code. Use of a static code analysis tool can help detect some possible problems. In the 9th century, the Arab mathematician Al-Kindi described a cryptographic algorithm for deciphering encrypted code, in A Manuscript on Deciphering Cryptographic Messages. Programs were mostly entered using punched cards or paper tape. 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. Many applications use a mix of several languages in their construction and use. Many factors, having little or nothing to do with the ability of the computer to efficiently compile and execute the code, contribute to readability.