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. 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. 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. High-level languages made the process of developing a program simpler and more understandable, and less bound to the underlying hardware. Different programming languages support different styles of programming (called programming paradigms). Text editors were also developed that allowed changes and corrections to be made much more easily than with punched cards. Whatever the approach to development may be, the final program must satisfy some fundamental properties. They are the building blocks for all software, from the simplest applications to the most sophisticated ones. Programs were mostly entered using punched cards or paper tape. For example, COBOL is still strong in corporate data centers often on large mainframe computers, Fortran in engineering applications, scripting languages in Web development, and C in embedded software. They are the building blocks for all software, from the simplest applications to the most sophisticated ones. Programmable devices have existed for centuries. The Unified Modeling Language (UML) is a notation used for both the OOAD and MDA. In 1206, the Arab engineer Al-Jazari invented a programmable drum machine where a musical mechanical automaton could be made to play different rhythms and drum patterns, via pegs and cams. After the bug is reproduced, the input of the program may need to be simplified to make it easier to debug. 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. 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. However, Charles Babbage had already written his first program for the Analytical Engine in 1837. Later a control panel (plug board) added to his 1906 Type I Tabulator allowed it to be programmed for different jobs, and by the late 1940s, unit record equipment such as the IBM 602 and IBM 604, were programmed by control panels in a similar way, as were the first electronic computers. 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. 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. Unreadable code often leads to bugs, inefficiencies, and duplicated code. 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. However, readability is more than just programming style. 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.