

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. Programming languages are essential for software development. One approach popular for requirements analysis is Use Case analysis. 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. 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). Many programmers use forms of Agile software development where the various stages of formal software development are more integrated together into short cycles that take a few weeks rather than years. After the bug is reproduced, the input of the program may need to be simplified to make it easier to debug. 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. 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. 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. Techniques like Code refactoring can enhance readability. 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. Implementation techniques include imperative languages (object-oriented or procedural), functional languages, and logic languages. The first compiler related tool, the A-0 System, was developed in 1952 by Grace Hopper, who also coined the term 'compiler'. 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. The choice of language used is subject to many considerations, such as company policy, suitability to task, availability of third-party packages, or individual preference. Machine code was the language of early programs, written in the instruction set of the particular machine, often in binary notation. The choice of language used is subject to many considerations, such as company policy, suitability to task, availability of third-party packages, or individual preference. Text editors were also developed that allowed changes and corrections to be made much more easily than with punched cards. Compilers harnessed the power of computers to make programming easier by allowing programmers to specify calculations by entering a formula using infix notation. 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. After the bug is reproduced, the input of the program may need to be simplified to make it easier to debug. However, Charles Babbage had already written his first program for the Analytical Engine in 1837. Code-breaking algorithms have also existed for centuries. Programs were mostly entered using punched cards or paper tape.