

Programming languages are essential for software development. Different programming languages support different styles of programming (called programming paradigms). 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. 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. Text editors were also developed that allowed changes and corrections to be made much more easily than with punched cards. A study found that a few simple readability transformations made code shorter and drastically reduced the time to understand it. Some languages are very popular for particular kinds of applications, while some languages are regularly used to write many different kinds of applications. Also, specific user environment and usage history can make it difficult to reproduce the problem. Techniques like Code refactoring can enhance readability. 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. The first step in most formal software development processes is requirements analysis, followed by testing to determine value modeling, implementation, and failure elimination (debugging). 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. 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. Machine code was the language of early programs, written in the instruction set of the particular machine, often in binary notation. Debugging is a very important task in the software development process since having defects in a program can have significant consequences for its users. By the late 1960s, data storage devices and computer terminals became inexpensive enough that programs could be created by typing directly into the computers. Implementation techniques include imperative languages (object-oriented or procedural), functional languages, and logic languages. Some of these factors include: The presentation aspects of this (such as indents, line breaks, color highlighting, and so on) are often handled by the source code editor, but the content aspects reflect the programmer's talent and skills. 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). 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. 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. Some languages are very popular for particular kinds of applications, while some languages are regularly used to write many different kinds of applications. The first compiler related tool, the A-0 System, was developed in 1952 by Grace Hopper, who also coined the term 'compiler'. Some text editors such as Emacs allow GDB to be invoked through them, to provide a visual environment.