

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. In the 9th century, the Arab mathematician Al-Kindi described a cryptographic algorithm for deciphering encrypted code, in *A Manuscript on Deciphering Cryptographic Messages*. A study found that a few simple readability transformations made code shorter and drastically reduced the time to understand it. Some text editors such as Emacs allow GDB to be invoked through them, to provide a visual environment. Programmable devices have existed for centuries. 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. Techniques like Code refactoring can enhance readability. In the 1880s, Herman Hollerith invented the concept of storing data in machine-readable form. After the bug is reproduced, the input of the program may need to be simplified to make it easier to debug. 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. There are many approaches to the Software development process. Integrated development environments (IDEs) aim to integrate all such help. Some text editors such as Emacs allow GDB to be invoked through them, to provide a visual environment. 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. Integrated development environments (IDEs) aim to integrate all such help. Their jobs usually involve: Although programming has been presented in the media as a somewhat mathematical subject, some research shows that good programmers have strong skills in natural human languages, and that learning to code is similar to learning a foreign language. High-level languages made the process of developing a program simpler and more understandable, and less bound to the underlying hardware. Use of a static code analysis tool can help detect some possible problems. 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 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. It affects the aspects of quality above, including portability, usability and most importantly maintainability. 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. After the bug is reproduced, the input of the program may need to be simplified to make it easier to debug. Popular modeling techniques include Object-Oriented Analysis and Design (OOAD) and Model-Driven Architecture (MDA). Text editors were also developed that allowed changes and corrections to be made much more easily than with punched cards.