However, with the concept of the stored-program computer introduced in 1949, both programs and data were stored and manipulated in the same way in computer memory. High-level languages made the process of developing a program simpler and more understandable, and less bound to the underlying hardware. 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. 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). 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. Implementation techniques include imperative languages (object-oriented or procedural), functional languages, and logic languages. Integrated development environments (IDEs) aim to integrate all such help. He gave the first description of cryptanalysis by frequency analysis, the earliest code-breaking algorithm. Some text editors such as Emacs allow GDB to be invoked through them, to provide a visual environment. This can be a non-trivial task, for example as with parallel processes or some unusual software bugs. High-level languages made the process of developing a program simpler and more understandable, and less bound to the underlying hardware. 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. Compilers harnessed the power of computers to make programming easier by allowing programmers to specify calculations by entering a formula using infix notation. Text editors were also developed that allowed changes and corrections to be made much more easily than with punched cards. There are many approaches to the Software development process. 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. 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. 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. Programming languages are essential for software development. Code-breaking algorithms have also existed for centuries. Compilers harnessed the power of computers to make programming easier by allowing programmers to specify calculations by entering a formula using infix notation. 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. Trial-and-error/divide-and-conquer is needed: the programmer will try to remove some parts of the original test case and check if the problem still exists.