Readability is important because programmers spend the majority of their time reading, trying to understand, reusing and modifying existing source code, rather than writing new source code. Integrated development environments (IDEs) aim to integrate all such help. 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. 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. Sometimes software development is known as software engineering, especially when it employs formal methods or follows an engineering design process. It is usually easier to code in "high-level" languages than in "low-level" ones. 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. Many factors, having little or nothing to do with the ability of the computer to efficiently compile and execute the code, contribute to readability. Allen Downey, in his book How To Think Like A Computer Scientist, writes: Many computer languages provide a mechanism to call functions provided by shared libraries. Machine code was the language of early programs, written in the instruction set of the particular machine, often in binary notation. Use of a static code analysis tool can help detect some possible problems. Debugging is often done with IDEs. Standalone debuggers like GDB are also used, and these often provide less of a visual environment, usually using a command line. Techniques like Code refactoring can enhance readability. He gave the first description of cryptanalysis by frequency analysis, the earliest code-breaking algorithm. This can be a non-trivial task, for example as with parallel processes or some unusual software bugs. 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. It is very difficult to determine what are the most popular modern programming languages. 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. 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. Provided the functions in a library follow the appropriate run-time conventions (e.g., method of passing arguments), then these functions may be written in any other language. Following a consistent programming style often helps 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. Some languages are very popular for particular kinds of applications, while some languages are regularly used to write many different kinds of applications. Debugging is a very important task in the software development process since having defects in a program can have significant consequences for its users. In the 1880s, Herman Hollerith invented the concept of storing data in machine-readable form.