Scripting and breakpointing is also part of this process. Different programming languages support different styles of programming (called programming paradigms). 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. Many factors, having little or nothing to do with the ability of the computer to efficiently compile and execute the code, contribute to readability. Code-breaking algorithms have also existed for centuries. 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. It affects the aspects of quality above, including portability, usability and most importantly maintainability. Compilers harnessed the power of computers to make programming easier by allowing programmers to specify calculations by entering a formula using infix notation. They are the building blocks for all software, from the simplest applications to the most sophisticated ones. High-level languages made the process of developing a program simpler and more understandable, and less bound to the underlying hardware. 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. 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. 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. 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. The academic field and the engineering practice of computer programming are both largely concerned with discovering and implementing the most efficient algorithms for a given class of problems. 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. Many factors, having little or nothing to do with the ability of the computer to efficiently compile and execute the code, contribute to readability. Techniques like Code refactoring can enhance readability. 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. Implementation techniques include imperative languages (object-oriented or procedural), functional languages, and logic languages. 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. The first step in most formal software development processes is requirements analysis, followed by testing to determine value modeling, implementation, and failure elimination (debugging). Many factors, having little or nothing to do with the ability of the computer to efficiently compile and execute the code, contribute to readability. Programs were mostly entered using punched cards or paper tape. Many applications use a mix of several languages in their construction and use.