Programs were mostly entered using punched cards or paper tape. Scripting and breakpointing is also part of this process. However, because an assembly language is little more than a different notation for a machine language, two machines with different instruction sets also have different assembly 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. 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. Some languages are very popular for particular kinds of applications, while some languages are regularly used to write many different kinds of applications. High-level languages made the process of developing a program simpler and more understandable, and less bound to the underlying hardware. It affects the aspects of quality above, including portability, usability and most importantly maintainability. Text editors were also developed that allowed changes and corrections to be made much more easily than with punched cards. This can be a non-trivial task, for example as with parallel processes or some unusual software bugs. Compilers harnessed the power of computers to make programming easier by allowing programmers to specify calculations by entering a formula using infix notation. There exist a lot of different approaches for each of those tasks. Ideally, the programming language best suited for the task at hand will be selected. Implementation techniques include imperative languages (object-oriented or procedural), functional languages, and logic languages. 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. It is usually easier to code in "high-level" languages than in "low-level" ones. Code-breaking algorithms have also existed for centuries. 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. He gave the first description of cryptanalysis by frequency analysis, the earliest code-breaking algorithm. Many factors, having little or nothing to do with the ability of the computer to efficiently compile and execute the code, contribute to readability. One approach popular for requirements analysis is Use Case analysis. This can be a non-trivial task, for example as with parallel processes or some unusual software bugs.