Programming languages are essential for software development. In the 1880s, Herman Hollerith invented the concept of storing data in machine-readable form. Programs were mostly entered using punched cards or paper tape. When debugging the problem in a GUI, the programmer can try to skip some user interaction from the original problem description and check if remaining actions are sufficient for bugs to appear. FORTRAN, the first widely used high-level language to have a functional implementation, came out in 1957, and many other languages were soon developed—in particular, COBOL aimed at commercial data processing, and Lisp for computer research. 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. Many applications use a mix of several languages in their construction and use. Methods of measuring programming language popularity include: counting the number of job advertisements that mention the language, the number of books sold and courses teaching the language (this overestimates the importance of newer languages), and estimates of the number of existing lines of code written in the language (this underestimates the number of users of business languages such as COBOL). Integrated development environments (IDEs) aim to integrate all such help. For this purpose, algorithms are classified into orders using so-called Big O notation, which expresses resource use, such as execution time or memory consumption, in terms of the size of an input. 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. These compiled languages allow the programmer to write programs in terms that are syntactically richer, and more capable of abstracting the code, making it easy to target varying machine instruction sets via compilation declarations and heuristics. Techniques like Code refactoring can enhance readability. It affects the aspects of quality above, including portability, usability and most importantly maintainability. Programs were mostly entered using punched cards or paper tape. 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. 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. By the late 1960s, data storage devices and computer terminals became inexpensive enough that programs could be created by typing directly into the computers. Techniques like Code refactoring can enhance readability. Later a control panel (plug board) added to his 1906 Type I Tabulator allowed it to be programmed for different jobs, and by the late 1940s, unit record equipment such as the IBM 602 and IBM 604, were programmed by control panels in a similar way, as were the first electronic computers. Normally the first step in debugging is to attempt to reproduce the problem. 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. A similar technique used for database design is Entity-Relationship Modeling (ER Modeling). 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.