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). 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. Many factors, having little or nothing to do with the ability of the computer to efficiently compile and execute the code, contribute to readability. 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. Expert programmers are familiar with a variety of well-established algorithms and their respective complexities and use this knowledge to choose algorithms that are best suited to the circumstances. Different programming languages support different styles of programming (called programming paradigms). There are many approaches to the Software development process. 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. 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. 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 affects the aspects of quality above, including portability, usability and most importantly maintainability. Programs were mostly entered using punched cards or paper tape. 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. It involves designing and implementing algorithms, step-by-step specifications of procedures, by writing code in one or more programming languages. Text editors were also developed that allowed changes and corrections to be made much more easily than with punched cards. 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. Code-breaking algorithms have also existed for centuries. 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. 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. 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. While these are sometimes considered programming, often the term software development is used for this larger overall process – with the terms programming, implementation, and coding reserved for the writing and editing of code per se. High-level languages made the process of developing a program simpler and more understandable, and less bound to the underlying hardware. 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. Computer programmers are those who write computer software. 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.