Trade-offs from this ideal involve finding enough programmers who know the language to build a team, the availability of compilers for that language, and the efficiency with which programs written in a given language execute..  
 Implementation techniques include imperative languages (object-oriented or procedural), functional languages, and logic languages.  
Use of a static code analysis tool can help detect some possible problems.  
  
 The first computer program is generally dated to 1843, when mathematician Ada Lovelace published an algorithm to calculate a sequence of Bernoulli numbers, intended to be carried out by Charles Babbage's Analytical Engine.  
 Following a consistent programming style often helps readability.  
 In the 1880s, Herman Hollerith invented the concept of storing data in machine-readable form.  
It affects the aspects of quality above, including portability, usability and most importantly maintainability.  
 A similar technique used for database design is Entity-Relationship Modeling (ER Modeling).  
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.  
Ideally, the programming language best suited for the task at hand will be selected.  
Compilers harnessed the power of computers to make programming easier by allowing programmers to specify calculations by entering a formula using infix notation.  
 New languages are generally designed around the syntax of a prior language with new functionality added, (for example C++ adds object-orientation to C, and Java adds memory management and bytecode to C++, but as a result, loses efficiency and the ability for low-level manipulation).  
 High-level languages made the process of developing a program simpler and more understandable, and less bound to the underlying hardware.  
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).