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. Some text editors such as Emacs allow GDB to be invoked through them, to provide a visual environment. High-level languages made the process of developing a program simpler and more understandable, and less bound to the underlying hardware. After the bug is reproduced, the input of the program may need to be simplified to make it easier to debug. Debugging is often done with IDEs. Standalone debuggers like GDB are also used, and these often provide less of a visual environment, usually using a command line. It is very difficult to determine what are the most popular modern programming languages. 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. Also, specific user environment and usage history can make it difficult to reproduce the problem. Proficient programming usually requires expertise in several different subjects, including knowledge of the application domain, details of programming languages and generic code libraries, specialized algorithms, and formal logic. 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. The first compiler related tool, the A-0 System, was developed in 1952 by Grace Hopper, who also coined the term 'compiler'. It affects the aspects of quality above, including portability, usability and most importantly maintainability. Debugging is a very important task in the software development process since having defects in a program can have significant consequences for its users. 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. 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. 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. 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. This can be a non-trivial task, for example as with parallel processes or some unusual software bugs. Popular modeling techniques include Object-Oriented Analysis and Design (OOAD) and Model-Driven Architecture (MDA). 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. 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. Compilers harnessed the power of computers to make programming easier by allowing programmers to specify calculations by entering a formula using infix notation. For example, COBOL is still strong in corporate data centers often on large mainframe computers, Fortran in engineering applications, scripting languages in Web development, and C in embedded software.