There exist a lot of different approaches for each of those tasks. 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. Many factors, having little or nothing to do with the ability of the computer to efficiently compile and execute the code, contribute to readability. Some text editors such as Emacs allow GDB to be invoked through them, to provide a visual environment. 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. 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. Following a consistent programming style often helps readability. 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). 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. 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. Whatever the approach to development may be, the final program must satisfy some fundamental properties. One approach popular for requirements analysis is Use Case analysis. Also, specific user environment and usage history can make it difficult to reproduce the problem. A study found that a few simple readability transformations made code shorter and drastically reduced the time to understand it. Programmable devices have existed for centuries. 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. 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. Code-breaking algorithms have also existed for centuries. 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. 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. Whatever the approach to development may be, the final program must satisfy some fundamental properties. High-level languages made the process of developing a program simpler and more understandable, and less bound to the underlying hardware. It is very difficult to determine what are the most popular modern programming languages. The Unified Modeling Language (UML) is a notation used for both the OOAD and MDA. 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.