High-level languages made the process of developing a program simpler and more understandable, and less bound to the underlying hardware. 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. They are the building blocks for all software, from the simplest applications to the most sophisticated ones. 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. It is usually easier to code in "high-level" languages than in "low-level" ones. Some languages are very popular for particular kinds of applications, while some languages are regularly used to write many different kinds of applications. Machine code was the language of early programs, written in the instruction set of the particular machine, often in binary notation. 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. 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. Popular modeling techniques include Object-Oriented Analysis and Design (OOAD) and Model-Driven Architecture (MDA). In 1206, the Arab engineer Al-Jazari invented a programmable drum machine where a musical mechanical automaton could be made to play different rhythms and drum patterns, via pegs and cams. Sometimes software development is known as software engineering, especially when it employs formal methods or follows an engineering design process. It is very difficult to determine what are the most popular modern programming languages. There exist a lot of different approaches for each of those tasks. Use of a static code analysis tool can help detect some possible problems. 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. 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. Programmers typically use high-level programming languages that are more easily intelligible to humans than machine code, which is directly executed by the central processing unit. Scripting and breakpointing is also part of this process. Some text editors such as Emacs allow GDB to be invoked through them, to provide a visual environment. 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. Programmable devices have existed for centuries. 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). Use of a static code analysis tool can help detect some possible problems.