

In the 9th century, the Arab mathematician Al-Kindi described a cryptographic algorithm for deciphering encrypted code, in A Manuscript on Deciphering Cryptographic Messages. High-level languages made the process of developing a program simpler and more understandable, and less bound to the underlying hardware. 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. The first step in most formal software development processes is requirements analysis, followed by testing to determine value modeling, implementation, and failure elimination (debugging). Normally the first step in debugging is to attempt to reproduce the problem. Implementation techniques include imperative languages (object-oriented or procedural), functional languages, and logic languages. 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. One approach popular for requirements analysis is Use Case analysis. Some languages are very popular for particular kinds of applications, while some languages are regularly used to write many different kinds of applications. 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. Scripting and breakpointing is also part of this process. Many factors, having little or nothing to do with the ability of the computer to efficiently compile and execute the code, contribute to 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). 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. Ideally, the programming language best suited for the task at hand will be selected. When debugging the problem in a GUI, the programmer can try to skip some user interaction from the original problem description and check if remaining actions are sufficient for bugs to appear. Techniques like Code refactoring can enhance readability. Programming languages are essential for software development. 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. Code-breaking algorithms have also existed for centuries. There exist a lot of different approaches for each of those tasks. 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. One approach popular for requirements analysis is Use Case analysis. Machine code was the language of early programs, written in the instruction set of the particular machine, often in binary notation. The first step in most formal software development processes is requirements analysis, followed by testing to determine value modeling, implementation, and failure elimination (debugging).