These compiled languages allow the programmer to write programs in terms that are syntactically richer, and more capable of abstracting the code, making it easy to target varying machine instruction sets via compilation declarations and heuristics. Programming languages are essential for software development. They are the building blocks for all software, from the simplest applications to the most sophisticated ones. Programmable devices have existed for centuries. By the late 1960s, data storage devices and computer terminals became inexpensive enough that programs could be created by typing directly into the computers. In the 9th century, the Arab mathematician Al-Kindi described a cryptographic algorithm for deciphering encrypted code, in A Manuscript on Deciphering Cryptographic Messages. One approach popular for requirements analysis is Use Case analysis. Later a control panel (plug board) added to his 1906 Type I Tabulator allowed it to be programmed for different jobs, and by the late 1940s, unit record equipment such as the IBM 602 and IBM 604, were programmed by control panels in a similar way, as were the first electronic computers. Programmable devices have existed for centuries. Debugging is a very important task in the software development process since having defects in a program can have significant consequences for its users. The first compiler related tool, the A-0 System, was developed in 1952 by Grace Hopper, who also coined the term 'compiler'. Some of these factors include: The presentation aspects of this (such as indents, line breaks, color highlighting, and so on) are often handled by the source code editor, but the content aspects reflect the programmer's talent and skills. 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. It involves designing and implementing algorithms, step-by-step specifications of procedures, by writing code in one or more programming languages. 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. 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). A similar technique used for database design is Entity-Relationship Modeling (ER Modeling). Many applications use a mix of several languages in their construction and use. Some of these factors include: The presentation aspects of this (such as indents, line breaks, color highlighting, and so on) are often handled by the source code editor, but the content aspects reflect the programmer's talent and skills. High-level languages made the process of developing a program simpler and more understandable, and less bound to the underlying hardware. Integrated development environments (IDEs) aim to integrate all such help. 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. This can be a non-trivial task, for example as with parallel processes or some unusual software bugs. However, readability is more than just programming style. The first step in most formal software development processes is requirements analysis, followed by testing to determine value modeling, implementation, and failure elimination (debugging).