One approach popular for requirements analysis is Use Case analysis..  
This can be a non-trivial task, for example as with parallel processes or some unusual software bugs.  
Provided the functions in a library follow the appropriate run-time conventions (e.g., method of passing arguments), then these functions may be written in any other language.  
Techniques like Code refactoring can enhance readability.  
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.  
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.  
Languages form an approximate spectrum from "low-level" to "high-level"; "low-level" languages are typically more machine-oriented and faster to execute, whereas "high-level" languages are more abstract and easier to use but execute less quickly.  
Some languages are more prone to some kinds of faults because their specification does not require compilers to perform as much checking as other languages.  
He gave the first description of cryptanalysis by frequency analysis, the earliest code-breaking algorithm.  
Unreadable code often leads to bugs, inefficiencies, and duplicated code.  
Ideally, the programming language best suited for the task at hand will be selected.  
It affects the aspects of quality above, including portability, usability and most importantly maintainability.  
 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.  
 Popular modeling techniques include Object-Oriented Analysis and Design (OOAD) and Model-Driven Architecture (MDA).  
 Following a consistent programming style often helps readability.