However, readability is more than just programming style.  
One approach popular for requirements analysis is Use Case analysis.  
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.  
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.  
Text editors were also developed that allowed changes and corrections to be made much more easily than with punched cards.  
In 1801, the Jacquard loom could produce entirely different weaves by changing the "program" – a series of pasteboard cards with holes punched in them.  
By the late 1960s, data storage devices and computer terminals became inexpensive enough that programs could be created by typing directly into the computers.  
This can be a non-trivial task, for example as with parallel processes or some unusual software bugs.  
A study found that a few simple readability transformations made code shorter and drastically reduced the time to understand it.  
For this purpose, algorithms are classified into orders using so-called Big O notation, which expresses resource use, such as execution time or memory consumption, in terms of the size of an input.  
There exist a lot of different approaches for each of those tasks.  
 Following a consistent programming style often helps readability.  
For this purpose, algorithms are classified into orders using so-called Big O notation, which expresses resource use, such as execution time or memory consumption, in terms of the size of an input.  
Ideally, the programming language best suited for the task at hand will be selected.  
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.