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..  
Use of a static code analysis tool can help detect some possible problems.  
  
 Computer programming or coding is the composition of sequences of instructions, called programs, that computers can follow to perform tasks.  
 Programs were mostly entered using punched cards or paper tape.  
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.  
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.  
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.  
 The first step in most formal software development processes is requirements analysis, followed by testing to determine value modeling, implementation, and failure elimination (debugging).  
Trial-and-error/divide-and-conquer is needed: the programmer will try to remove some parts of the original test case and check if the problem still exists.  
Integrated development environments (IDEs) aim to integrate all such help.  
However, with the concept of the stored-program computer introduced in 1949, both programs and data were stored and manipulated in the same way in computer memory.  
Ideally, the programming language best suited for the task at hand will be selected.  
Many applications use a mix of several languages in their construction and use.  
Text editors were also developed that allowed changes and corrections to be made much more easily than with punched cards.  
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.