Consequence of the nature

Bugs are a consequence of the nature of human factors in the programming task. They arise from oversights or mutual misunderstandings made by a software team during specification, design, coding, data entry and documentation. More complex bugs can arise from unintended Interactions between different parts of a computer program. This frequently occurs because computer programs can be complex ? millions of lines long in some cases ? often having been programmed by many people over a great length of time, so that programmers are unable to mentally track every possible way In which parts can Interact.

Chain of Events: Recent efforts to reform the management of elections in Kenya received a major setback following the malfunctioning of electronic transmission of votes, putting the credibility of the presidential poll on trial. The system, based on the acquisition of one-way phone lines from telecoms operator Safari connected to servers at the Independent Electoral and Boundaries Commission's national tallying centre at the Somas of Kenya, was a key measure in improving the credibility of the presidential poll, especially the relay and central tallying of the results.

The malfunction was attributed to a software bug In a program developed In-house by the Piece's IT department, which, besides being slow, also automatically inflated figures received as rejected votes. To illustrate the scale of the anomaly, the electronically transmitted results showed a total of over 338, 000 rejected votes in the space of Just a few hours compared with Just over 58, 000 rejected votes by 6. 30 pm on Thursday evening. The first to fail were the voter identification kits, which on polling day, could not be used in at least 80 per cent of the polling stations.

Polling clerks, frustrated by passwords that did not work and batteries that had not been charged, among other glaring mistakes, were forced to resort to the manual Identification of voters. It Is said that the PIECE system could have malfunctioned due to either unauthorized access by an " outside party" or the introduction of a rogue code by the electoral body IT experts during the development of the software, which apparently was done in-house. It also emerged that PIECE had been warned before the elections that their results transmission system was not election-ready but that advice was Ignored.

Thus wowing that not enough tests were carried out on the new software. Reasons as to why the bug could not be gotten rid of: 1 OFF bugs. The bug could be fixed in a new version or patch that is not yet released. The changes to the code required to fix the bug could be large, expensive, or delay finishing the project. Even seemingly simple fixes bring the chance of introducing new unknown bugs into the system. At the end of a test/flux cycle some managers may only allow the most critical bugs to be fixed.

Users may be relying on the undocumented, buggy behavior, especially if scripts or macros rely on a behavior; it ay introduce a breaking change. Prevention of bugs occurring in software in the future: Programming style - While typos in the program code are often caught by the compiler, a bug usually appears when the programmer makes a logic error. Various innovations in programming style and defensive programming are designed to make these bugs less likely, or easier to spot.

In some programming languages, so-called typos, especially of symbols or logical/mathematical operators, actually represent logic errors, since the mistyped constructs are accepted by the compiler with a meaning other than that which the programmer intended. Programming techniques - Bugs often create inconsistencies in the internal data of a running program. Programs can be written to check the consistency of their own internal data while running. If an inconsistency is encountered, the program can immediately halt, so that the bug can be located and fixed.

Alternatively, the program can simply inform the user, attempt to correct the inconsistency, and continue running. Development methodologies - There are several schemes for managing programmer activity, so that fewer bugs are produced. Many of these fall under the discipline of software engineering (which addresses software design issues as well). For example, formal program specifications are used to state the exact behavior of programs, so that design bugs can be eliminated.

Depending on the language and implementation, this may be caught by the compiler or at run-time. In addition, many recently invented languages have deliberately excluded features which can easily lead to bugs, at the expense of making code slower than it need be: the general principle being that, because of Moor's law, computers get faster and software engineers get slower; it is almost always better to write simpler, slower code than " clever", inscrutable code, especially considering that maintenance cost is considerable.

For example, the Java programming language does not support pointer arithmetic; implementations of some languages such as Pascal and scripting languages often have runtime bounds checking of arrays, at least in a debugging build. Code analysis - Tools for code analysis help developers by inspecting the program text beyond the compiler's capabilities to spot potential problems. Although in general the problem of finding all arrogating errors given a specification is not solvable (see halting problem), these mistakes when writing software.

Instrumentation - Tools to monitor the performance of the software as it is running, either specifically to find problems such as bottlenecks or to give assurance as to correct working, may be embedded in the code explicitly (perhaps as simple as a statement saying PRINT " I AM HERE"), or provided as tools. It is often a surprise to find where most of the time is taken by a piece of code, and this removal of assumptions might cause the code to be rewritten.