specifications and writes a program in accordance with them. In large systems there will be some interaction between these two steps but they are essentially separate procedures.

Software errors are not always the result of carelessness (for example, a missing coma). Frequently, errors cannot be discovered until after a program has been implemented:

> The "unforeseen events" that cause trouble are less often unforeseen external events than perverse concatenations of perfectly normal events.

Complex interaction between the various parts of a program can occasionally produce wholly unexpected, erratic results. There are too many possibilities for interaction in modern computer systems for it to be feasible to test all of them, so numerous techniques have been developed to find errors.

Two classes of errors occur in writing programs from specifications. The first is the "typo" – A Mariner space probe was lost because a period was put where a comma should have been. The second, the "thinko," involves minor errors in reasoning. Most of the latter result either from the fact that the specifications themselves contain ambiguities or lapses in reasoning, or from the problem that English, with all its ambiguities, has become the language most often used for specifications.

In theory, it should be possible to check a program against its specifications, but the techniques of "program verification" are not as accurate as the name would suggest. Ironically, both Greg Nelson and David Parnas, two top software engineers in this area, have become vocal opponents of the Strategic Defense Initiative because they believe it is impossible to build trustworthy software. Although program verification is an important and powerful tool, it is not a foolproof solution. In large systems the full program is never run, since full-scale testing can be embarrassing or truly impracticable. Consequently many possibilities are never investigated. Smart designers and programmers are the best antidote for bugs, but they cannot provide a complete solution. Techniques of simulation also help to identify errors. Here person A writes the specifications for the original program and then person B writes specifications for a program designed to imitate the environment with which the original program is to deal. This method is of great help in identifying errors but it, too, is affected by shared oversights and misconceptions. Having identified errors, fixing them, especially in large systems in which they are deeply imbedded, can be a difficult process and may itself create further problems. For example, the Advanced Research Projects Agency (ARPA) computer network needed a major redesign after it grew beyond a certain size.

To cope with the persistence of errors, humans have developed "fault-tolerance" techniques, based on the idea of building spare parts into a