The Need for Formal Methods

We all know from personal experience that computers do not work correctly all the time. For most of us this leads to nothing more than delays and frustrations as we encounter automatic bank tellers which are out-of-order, or face long delays at airports as glitches in the booking or flight control system are being catered for.

However, the problems of systems failures become more serious (and deadly) as automatic control systems find their way into almost every aspect of our lives. It is recognised that reliability of any major software system is beyond expectation. While, for example, civil and mechanical engineers can build impressive bridges which are guaranteed to remain standing, and aeronautical engineers can design airplane wings which behave in precise fashions, Software Engineers are never so successful in practice. The main reason for this failure comes from a lack of a rigorous mathematical basis to the field. While traditional engineering fields build on well-established tools and techniques from continuous mathematics, and can use this basis to guarantee the reliability of their products, Software Engineering has always relied more on the intuition and intelligence of its practitioners than on any rigorous mathematical basis.

The aim of this course is to introduce you to some of the techniques which, when applied, can help to reduce the number of errors present in a system. Errors can arise at many points in the software development process, from understanding exactly the requirements and behaviour of the system being built, to ensuring that these requirements are correctly captured in the design and implementation of the system.

We begin by considering a few high-profile episodes showing the need for reliable methods for system development.

**USS Scorpion**

In 1968, the submarine USS Scorpion was destroyed killing all of its 99 crew members. The submarine was destroyed by one of its own torpedoes which had been accidentally activated and then ejected. The torpedo sought out its nearest target as it had been designed to do so. (See P G Neumann, *Computer Related Risks*, Addison Wesley, 1994.)

The negative implications of seemingly sensible and harmless design decisions often arise only in hindsight after disaster has struck. Clearly every eventuality needs to be accounted for, especially in “safety-critical” areas, where failure of the system could lead to injury, illness or loss of life; serious environmental damage; or major financial loss.

**Therac 25 Radiotherapy Machine**

The Therac 25 was a radiation therapy system that intermittently gave the wrong radiation doses over a period of three years (1985-87) due to errors in the software controlling its operation. The problems with the Therac 25 have been very thoroughly analysed. (See N. Leveson and N., C.S. Turner, “An Investigation of the Therac-25 Accidents,” *IEEE Computer* 26(7):18–41, July 1993.) The major problems with the system have been attributed to poor interface design and software failure. Six accidents, three of them fatal, have been attributed to failures in Therac 25.

The basic issue involved the replacement of hardware interlocks used in previous models with a software-only system. The machine had two modes of operation, Electron mode and Photon (or X-ray) mode, used for treating tumours at different depths. Electron mode involved a low-power electron beam, while photon (X-ray) mode involved a high power electron beam (3 orders of magnitude more powerful), but with a metal plate between it and the patient, to generate the X-rays. The electron beam must be in low-power mode if the plate is not present, and in earlier designs (Therac 6 and Therac 20) there was a mechanical device which physically ensured this. This hardware interlock was removed from the Therac 25 which was left to rely on a (faulty) software interlock.

The software was poorly designed and specified, and much of it was imported as-is from the previous models despite changes in requirements. The problem was compounded by a complex user interface. In some cases, if the operator tried to enter certain control sequences (either in error or as shortcuts), the machine would operate incorrectly, using the high-power beam with no plate. It would then report an error, which it would normally do when no treatment had been delivered, often leading operators to repeat the process.

1
London Ambulance Service

In 1992, the London Ambulance Service installed a computerised dispatch system to control the dispatching of ambulances across London; it was to automatically match up each call to be responded to with the closest available ambulance. However, the system was unable to cope with real-time data, which was on the order of 5000 calls per day. As it became more and more swamped with information and requests, it repeatedly generated more and more exception messages requiring human intervention; the volume of these messages caused the exception messages, together with information needing to be dispatched to ambulances, to scroll off the top of the controllers’ screens. As many as twenty deaths have been attributed to failings of the system.

The London Ambulance Service quickly reverted partially to its manual dispatching system. However, after eight days, the automated system crashed completely, leading the service to revert completely to the original manual system. (See A. Finkelstein and J. Dowell, “A Comedy of Errors: The London Ambulance Service Case Study,” in The Eighth International Workshop on Software Specification and Design, IEEE CS Press, pp2–4, 1996.)

The Intel Pentium Bug

When the Intel Pentium PC was initially released in 1994, problems were found in its floating-point unit; with certain inputs, the unit gave inaccurate results when performing division, thus rendering it useless for mathematical or scientific work.

The error had been caused in the design stage of the chip when a new method for floating-point division was implemented which was three to five times faster than previous methods. The algorithm is based on using look-up tables to calculate intermediate results. The hardware was programmed using a program to download values into the look-up tables; however, there was an error in this software that caused five of the 1066 entries to be downloaded incorrectly into the logic chip.

Because the calculations are based on recursively using information from the look-up tables, the errors that can accrue magnify in scale. For example, performing the sum $x - \frac{x}{y} \times y$ should return the answer 0 for any inputs x and y. Given that computers have to deal with approximations to real numbers, we typically have to settle for a value close to zero to be returned. But with input values $x = 419583$ and $y = 3145727$ the first Pentium release gave the answer 256. (See T.R. Halfhill, “The Truth Behind the Pentium Bug,” Byte, March 1995.)

Ariane 5

In 1996, the Ariane 5 space rocket exploded 40 seconds after lift-off. Its self-destruct system was initiated when the rocket detected it was disintegrating. This damage was caused by friction with the atmosphere as the rocket was travelling at too shallow an angle.

The flight path of the rocket is controlled by two software components, one providing the flight data, and the other converting this data into signals which control nozzles that direct the rocket’s boosters. The problem was found to be with the software providing the flight data, which was imported as-is from the earlier Ariane 4 (a similar problem underlying the Therac failure).

The software executed an instruction to convert a 64-bit integer to a 16-bit representation on a number that was too big to be stored as a 16-bit integer. (Ariane 5 used a different flight path from Ariane 4 which involved a shorter period of vertical ascent before yawing over to accelerate, thus reaching shallower angles than Ariane 4 sooner in the flight; this problem thus never arose with Ariane 4.) As there was no code to deal with this exception, the program crashed, and the ensuing error messages output by the system were interpreted by the guidance system as flight data.

Ironically, the part of the software that failed was only needed by Ariane 4 before lift-off, and was only active during the first part of the flight due to the possibility of a short hold prior to lift-off. This piece software was unnecessary for Ariane 5. (See http://www.esrin.esa.it/htdocs/tidc/Press/Press96/ariane5rep.html.)