D3: The avoidable, the unavoidable and the unforeseeable

Les Hatton, lesh@oakcomp.co.uk

Sep 1996

Since February, I have visited Germany, Holland several times, India, Singapore, the US, and most recently, Japan, which I now visit twice a year, so its been a good year for air miles and an exceptionally poor year for my digestive system and general well-being. For someone who works with software failure, sitting on a modern fly-by-wire passenger jet for many hours is a particularly insidious form of torture, (will the passenger in seat 42D please stop screaming as he is upsetting the other passengers). My visits are concerned with the business of software defect prevention, and it is pleasant to report the importance that some nations now place on defect prevention rather than cure.

Given the strategic importance of reducing corrective maintenance in software, (for every pound you spend developing a system, you spend two pounds correcting it on average), I'll spend this month's article talking about defect prevention. From a systemÕs point of view, there are basically three kinds of defect, the avoidable, the unavoidable and the unforeseeable. Avoidable defects can be further sub-divided into directly detectable and indirectly detectable defects.

Avoiding the avoidable is an exceptionally beneficial engineering process. Numerous sources of data suggest that perhaps 40% of all system failures could have been directly detected even before compilation of the source code. These defects are typically made up of requirements problems and coding problems. The interesting thing about these is that for many years we have been making the same kinds of mistake repeatedly. The problem which spread Ariane 5 all over the landscape has been identified in numerous other really expensive system failures over the years. You can even analyse code statically using special tools, i.e. without executing the code, which detect a significant percentage of faults which are likely to develop into real failures, just like wheel tapping on trains can detect cracks, (faults) before they become fractures, (failures). Typically, such tools directly detect around 10 'time-bombs' per 1000 lines of source code even in released code. By contrast, indirectly detectable problems involve exploiting known macroscopic properties of the defect density curve, of which more later. IÕll treat the unavoidable and the unforeseeable together. I have distinguished between the two, because they are currently treated differently in European Product Liability Law, (a subtle point which I will discuss later). Unavoidable problems are known to be in the system, but there exists no systematic procedure for removing them. Unforeseeable problems, by definition, cannot be planned for. However, both categories should have their potential effect on the user of the software explored using the mechanism of hazard analysis, a conventional process in traditional engineering. So even if you canÕt stop something happening, the very least you can do is limit its potential effects by appropriate design. Unfortunately, we donÕt normally teach software engineers about such techniques.

I've given up with Windows '95. Quite apart from the bugs, it makes a Pentium run as if it was on some form of silicon narcotic. When you run Linux, you see what can be achieved. See you next month.