Zero-defect software and other figments of the imagination

I had a choice of juicy failures to talk about this week. I finally decided to discuss briefly the air-traffic hiatus which occurred at NATS on the 4th June when a 'routine upgrade' (one of my favourite ways of screwing up a system) led to a 45 minute loss of service and some significant amount of inconvenience. I found this interesting because the first thing I knew about it was when a journalist rang me up asking for my comments. I am always a little leery about this because the most obvious deficiency in software system failures in my experience is any kind of detailed information which would help to shine some light on it.

Let me develop this theme a little. I'll start with the notion of zero-defect systems, a tantalising but unattainable product of somebody's overactive imagination some years ago. The reality is this: it is overwhelmingly unlikely that any programmer will produce a zero-defect system of any complexity in their lifetime. Moreover, if they did, first of all, they wouldn't know it as we have no technology to guarantee the absence of defects and secondly, they wouldn't be able to repeat it as it would be no more than a statistical anomaly. This leaves us with the following two engineering obligations which should be taught to all budding software developers, (and some not so budding).

First, when a software developer designs a system, he or she should design it so that WHEN it fails (and it most surely will), it fails in such a way as to cause minimum inconvenience and danger to the user.

Second, the system should be designed in such a way that the failure can be quickly and efficiently traced back to its corresponding fault or faults so that they can be corrected and any future re-occurrence of the failure avoided. As a result, the system will get incrementally better as time goes by.

These two obligations are as fundamental to software development as Isaac Asimov's venerable and justifiably famous laws of robotics would be to some future robotics engineer, (always assuming of course that the robot control system hadn't unfortunately crashed preventing it from carrying out its long defined duties).

This is not the first time NATS has had problems (for example there were three different outages in 5 weeks in Spring 2002) and nor will it be the last. I do not know what their policy is with respect to such failures but it is unfortunately the case in most organisations that finding out anything relevant to the above two obligations is usually very difficult. There are exceptions such as the enquiry into the spectacular failure of Ariane 5 in 1996 which rapidly produced a publicly accessible and detailed description as to why it failed allowing others to learn all-important lessons. Until we do this systematically with all such failures, we will continue to produce systems which according to the recent report produced jointly by the Royal Academy of Engineering and BCS cost the UK alone billions of pounds per year. I don't know how much longer we will continue producing poor systems when we can do so much better but if web software is anything to go by, I'm not holding my breath.

L.Hatton@kent.ac.uk