The Chimera of Software Quality

Les Hatton, 15-Sep-2006

From time to time, I think it very important to remind the scientific community about the underlying quality of the software on which our results and progress increasingly depend. To put it baldly, most scientific results are corrupted and perhaps fatally so by undiscovered mistakes in the software used to calculate and present those results. Let me elaborate.

I've spent the last 30 years analysing the quality of software controlled systems. In every area I have looked or worked, software defects, often previously undiscovered, are rife. It is no better today than twenty years ago. In scientific modelling, these defects may lead to very significantly misleading results. 12 years ago, with a co-author I published the results of a large study of high quality signal processing software in the oil industry. Previously undiscovered defects had effectively reduced precision in these data from 6 significant figures to between 1 and 2. However, these data are used to site oil wells and need to be of at least 3 significant figure precision to perform this task, effectively randomising the decision-making progress. We were only able to discover this because the same software had accidentally evolved nine different times in different companies in commercial competition. Within five years of this, seven of these companies had been bought out or disappeared so we no longer know the scale of the problem.

A parallel experiment suggested that similar problems afflict other scientific modelling areas. Sometimes these defects reveal how smoothed our simulations actually are. Thirty years ago when translating to a sigma coordinate system, I found and corrected an alarming defect in the standard daily forecasting model at the Met Office which zeroed the non-linear terms in the governing Navier-Stokes equations every other time step. (The importance of this is that the whole of the weather is generated by those non-linear terms.) When I reran the model, the differences were almost impossible to see. We can today perform mutation tests to assess this level of sensitivity but in my experience they are rarely used. I have on an uncomfortable number of occasions been told in forceful terms by an elementary particle physicist or a specialist in anisotropic wave propagation or whatever that “their software contains no bugs because they have tested it.” This attitude really troubles me. I am a computational fluid dynamicist by training and know that verifying the science part of any model is easy compared with producing a reliable computer model of that science but I still can't convince most scientists of this even though I am a member of the same club.

So how good is good ? In computer science, we regrettably operate in a largely measurement-free zone. Very few experiments are done and even fewer results are published. This has been noted a number of times over the years by researchers such as Walter Tichy in Karlsruhe. As a result, software development isn't an engineering industry, it is a fashion industry populated by unquantifiable statements and driven by marketing needs. We are exhorted to develop using Java Beans or OO this or UML that and that this will fulfil our wildest dreams. It is arrant nonsense. Such experiments as we have managed to carry out suggest that by far the biggest quality factor in software is the ability of the person developing it. It appears to have very little to do with any technique or even language they might choose to use. In my experience as an employer, it doesn't even appear to have much to do with their educational background either.

The bottom line in software quality is still measured in the number of faults that have failed per thousand executable lines of code in the life-cycle of the software. The relationship between this and more conventional engineering measurements such as MTBF (Mean Time Between Failures)

remains unknown. However, the best systems in the world appear to be around 0.1 on this scale. In other words, the best systems in the world exhibit about one fault that fails every 10,000 executable lines of code, measured over the entire lifetime of the software from birth to death. Perfection is not an option and the effect of these faults when they fail on the output of the program is unpredictable. It costs a lot of money to stay this good and my own and other researchers work suggests that it is at least ten times worse and possibly as much as one hundred times worse in typical computer models not subject to the kind of quality control necessary to stay as low as 0.1.

There are a few more restrictions. Nobody knows how to produce a fault-free program. Nobody even knows how to prove it even supposing we were magically provided with one. I teach my students that in their whole careers, they are exceedingly unlikely ever to produce a fault-free program and if they did, they would never know it, they could never prove it and they could not systematically repeat it. It provides a usefully humble starting point. Some of my colleagues hold out hope for truly verifiable programs but such methods do not currently and may never scale to the size of systems we regularly produce and much remains to be done.

In other words, unless we are in complete denial, we know the faults are there but have no methodology to relate the nature of a fault to its ultimate effect on the run-time behaviour and results produced by a computer model.

Even in the world of pure mathematics, we are straying towards an era where computer programs are used as part or indeed all of a proof. An early example of this was the four-colour theorem. However, computer programs are fundamentally unquantifiable at the present stage of knowledge and any proof based on them *must be considered flawed* until such time as the same level of verification can be applied to a program as is applied to a theorem. Scientific papers are peer reviewed with a long-standing and highly successful system. The computer programs we use today to produce those results are generally off the peer-review radar. Even worse, scientists will swap their programs thus passing on the virus of undiscovered software fault.

Does industry fare any better or worse ? In my experience, its probably better because successful test procedures are more widely used than they are in academia which normally cannot afford the degree of verification necessary to reduce defects to an acceptable level. Even so, the world is rife with software failure. My television set-top box crashes about every 7 hours according to my records and shuts itself off in about 1 in 3 cases. It is a piece of junk. My PC packages crash frequently. Updating my gas meter reading on the British Gas telephone entry system failed the first time and accepted the same reading the second time last week, having successfully repeated it back to me both times. The automobile industry is beginning to suffer very high levels of recall based on software failures affecting all electronically-controlled parts of the car including but not limited to the brakes, engine management system and airbags and even made the New York Times last year.

Poor software quality affects us in other ways. If the country as a whole really understood just how much money UK plc throws away on failed software projects, there would be a national outcry. In 2004, the Royal Academy of Engineering made an authoritative case in a comprehensive report after interviewing a number of experts. In spite of this the initiative as far as I can see has stalled, afflicted by the peculiarly widespread *laissez faire* attitude which attends anything to do with computers. People simply do not care enough. However, the amount of money wasted is very likely in the region 10-20 billion pounds per year. This is simply thrown away. The National Institute for Standards and Technology produced a hauntingly similar conclusion in the USA in

2002. Quite recently, a number of my distinguished colleagues wrote to the Times stating the case for an independent audit of the troubled and truly gigantic National Health Service Project. Contemporaneously, I interviewed 10 very disparate members of the NHS at random and received unanimous and very deep concerns about the quality and relevance of this system. I've analysed enough failed systems in my time to know that there are two classic symptoms of a system on its way to the fairies. First, no independent audit is allowed and second, talking heads tell you everything is fine when the ultimate users tell you the opposite. Ironically, as I wrote this line, my word-processor crashed, probably in sympathy.

Not all is bleak. The Linux kernel is now arguably the most reliable complex software application the human race has ever produced with a mean time between failures reported in tens and in some cases, hundreds of years. Poetically, the development environment of Linux whereby thousands of volunteers on the web give their spare time for the public good, breaks just about every rule which software process experts hold dear but this community has demonstrated that it is perfectly possible to produce extraordinarily reliable software. There are other examples of very highly reliable applications.

The accumulating evidence of all this work is that most of the software failures and disasters which afflict us today could have been avoided using techniques we already know how to do. They affect everybody and it is no use ignoring it. In a scientific context, they undermine the very fabric of our work so must we really continue building scientific castles built on software sands when we could do so much better ? I really do hope not.

Les Hatton has spent most of his working life in industry first as a meteorologist, and then as a geophysicist and applied mathematician in the UK and the USA before returning part-time to academia as Professor of Forensic Software Engineering at Kingston University in 2004.
http://www.leshatton.org/