A personal reflection on software engineering 1990-2010

Les Hatton

February 27, 2011

1 The first twenty years

Twenty years ago, I had just changed careers from geophysics to computer science and had the great pleasure of meeting Tom, Bev, Felix and others associated with the Club early in this transition. I was embroiled in the middle of two large and very time-consuming experiments which were eventually published as the T-experiments, [2]. Both were associated with the distribution of defects in software systems, an area of intense interest in the construction of safety-critical systems. I was naive enough to think it would be of interest to all software engineers believing that the optimal elimination of defect was highly desirable in all systems, whatever those systems did. Unfortunately the profession as a whole mostly does not share our enthusiasm and, by and large, we have continued in the "Ready, Shoot, Aim" school of software development but more of this later.

The results of these and other more recent experiments convinced me that software defect exercised (and continues to exercise) a pervasive and essentially unquantifiable effect on a very wide variety of software systems, including those influencing safety. Indeed certain kinds of defect appeared to be just as prevalent in safety-critical systems as they were in safety-agnostic systems, notably defects in the use of programming language. These were even present in systems developed using formal methods, [4]. At the time, formalism was a strong contender for use, certainly in critical systems, so in common with a number of other researchers, I set about trying to understand this and indeed how to prevent such defects.

Eventually this resulted in mechanisms for dealing with deficiencies and ambiguities in standardised languages and contributed in some small way to the use of safer subsets which all languages need to a greater or lesser extent. How much has this helped ? Well, although we made some early progress in [4] demonstrating that such defects were directly associated with recorded failures in that system, experimental support as with so many of our other conjectures has not been pursued widely and the relationship between defect and failure remains opaque.

1.1 Vaulting ambition and language decay

Programming languages are an obvious target for rationalisation for anybody who has used a few of them over the years. Although programming skill is in steep decline in computing, certainly in the UK, fluency in them remains of course a *sine qua non* when building a system. Unfortunately, languages tend to evolve rather iconoclastically. We have known for almost 50 years thanks to the Böhm-Jacopini theorem [1] that the only constructs a programming language actually *needs* are sequence, selection and iteration and the ability to create identifiers. Add a bit of maths in the form of arithmetic operations and off you go. Unfortunately, language designers don't stop there and add all sorts of exciting stuff, experimental and otherwise, to attract users like flies to jam.

This would be OK in itself until you add in the sinister aspect of international standardisation. Whilst this has sensible principles at its heart, (one standard definition of a language which everybody uses), in practice the logistics of standardisation tend to make it easy to add new untested features whilst making it almost impossible to remove existing features which have unintended consequences. We call this particular insanity, *backwards compatibility*, which requires screw-ups to continue to screw-up in the same way in perpetuity.

The bottom line is that safety-critical language experts have had to spend too much of their time trying to remove features from programming languages which have proven to be unsafe but seemed like a good idea at the time to the language designers. This means that on each international language update, the standards get bigger and bigger and bigger - 'Hmmm, C has got pointers, great idea, let's have them in ours; dynamic memory, wonderful, let's get it in; Hmmm, what's this OO stuff, never mind, let's have some; Zowie, lambda calculus, that's as incomprehensible as entropy, we must have that' - and so it goes on. The current dinosaur in the room is the latest draft of C++ which weighs in at more than 1,300 pages (and has all of the above, including lashings of entropy).

Do we seriously expect programmers to implement systems which need to be exceptionally well-specified in monster duck-billed platypus languages like these ? Unfortunately we do.

1.2 Bureaucracy sprouts like weeds

Twenty years ago, development was also characterised by great optimism that process would save us and if that didn't, metrics would, whatever they are. Software process is a method of introducing bureaucracy into software development. Such bureaucracy would be valuable if it was related to measurement but we are not in general a critical discipline and certainly not a scientific one, so we just have bureaucracy instead. The measurement principles originally at the heart of process models like the CMM seem to have been forgotten.

Unfortunately, bureaucracy unconstrained by scientific principles grows in an unbounded way; it sprouts like weeds because there is simply nothing to stop it. I have written elsewhere on this, [3], although nowhere is it more evident or

tragic than in the 2006 Nimrod disaster where a long period of bungling and incompetence by all parties concerned led to this extraordinary summary by Charles Haddon-Cave QC, the Chairman of the accident inquiry:-

> *"The Nimrod Safety Case was a lamentable job from start to finish. It was riddled with errors. It missed key dangers. Its production is a story of incompetence, complacency and cynicism."*

In other words, what we seem to have is the rise of slavish adherence to rather arbitrary rules by managers who frequently do not have an engineering background or if they do, are ground down by the system and degenerate into box-tickers. This is certainly not helped by the oft repeated mantra that we need engineers with "more business skills". Let me give you another quotation if I may, this time from the British Computer Society web-site:-

> *"When we talked to CIOs of large companies with a turnover 250m - 1bn, over a third told us they want more business and management skills in their department, the lack of which, they say, is costing more than 10 per cent in departmental productivity and performance."*

I also invite you to read the rest of this page[1]. Think of it as an epitaph on what was once Britain's great engineering past entombed in paper by a management class who generally do not have the first idea of engineering principles.

At the risk of tarnishing my already tattered image even further, much more of this clap-trap and we won't have an IT industry at all. How did we allow something as serious as engineering to be hi-jacked in this way ?

Well, this is how.

1.3 Decline in mathematics skills

First of all, thanks to 25 years of deranged education policies and the systematic undermining of engineering and scientific careers in the UK, we have finished up in the current situation where "hard" subjects like mathematics and science have been systematically usurped by "soft" subjects. At my university students' fair last year, the Computing and Maths Faculty was next to the Arts Faculty. During the day, the Arts Faculty exhibit was like an ant's nest whilst we had I think 12 all day. Please don't think I have it in for the Arts. I adore music and read avidly about archaelogy and history but, and its a crucial difference, I also understand partial differential equations and can reason about pointers to pointers without vital life functions shutting down completely.

What seems to have happened is that subjects like mathematics and English have been treated as a kind of option next to subjects like citizenship. No doubt citizenship is important but it remains a subject, an option. Mathematics and English are *not* subjects, they are core life skills. As a result, in the UK, our OECD Reading, Mathematics and Science triplet world rankings have slumped from (7,8,4) to (20,22,11) in just 10 years.

[1] http://www.bcs.org/content/conWebDoc/37718, accessed 21-Feb-2011

So what has this to do with computing ? Well, ever since mathematics became sort of optional like flower-arranging, it isn't required for computing degrees any more. Unfortunately, programming languages incorporate mathematical expressions and require the kind of problem solving skills which only appear to evolve with the "hard" subjects. Consequently we are producing a generation of computing students a large percentage of whom have effectively no programming skills. We call this subject Information Systems (IS) and it helps promote the great fallacy amongst IT management, politicians and civil servants, that implementation is "a detail".

So, let me summarise:-

- We make programming languages and supporting methodologies progressively more incomprehensible
- We deprive computing students of vital mathematics and therefore programming language and analytic skills.
- We give the vast majority of the student population 2:1s or firsts
- We claim through our professional societies that business skills are a priority.

So can anybody tell me who is actually going to build the damn things, or are we just going to outsource them all ? Actually, I think I've answered my own question since the financial sector takes all the good students and at 550 pounds a day for C++ and "risk" skills in an investment bank as I saw today, who can blame them ? It certainly beats the pants off being a professor of computer science.

1.4 Measurement and Popperian deniability

Even if we were to resolve the UK's historic undervaluing of engineers and scientists, there remain computing's own technical deficiencies. In the last 20 years, what technological improvements have we seen exactly ?

Well, as I have hinted, the languages have all become far more complex. In one or two cases, efforts have been made to ameliorate this for example by making compiler test suites publicly available, as in the Ada community. Others like C and C++ are busy devouring their own tails like the worm Ourobouros. On top of this we are riddled with bureaucracy and we have downgraded important technical skills.

In my view however, the real villain is ourselves for not taking the opportunity to lay down a measurement system which allows us to make systematic progress. Systematic progress means the careful exploitation of best practice. We still don't even know what best practice means. If we had the makings of an empirical system we would not have become slaves to ephemeral management practices and there would have been a systematic improvement in our tools and technologies. Instead, we (mostly) build systems the way we always did - "Ready, Shoot, Aim" as I said at the beginning. We have managed to disguise

this by inventing new concepts like "agile" ("Ready, Shoot, Aim, Shoot, Aim, Shoot, Aim" ...), IS ("Ready, Ready, Ready") and "mash-up" ("Shoot, Aim, Ready"). I suppose that's progress of a sort.

2 A positive outlook

You may wonder if I have a positive outlook ? Well, it is 20 years since the Club was established. It is still together and it attracts people who care about these things. I am sorry to see that interest has not grown as much as it should or deserved but that is not the fault of the Club. That should be laid squarely at the door of our policy makers who remain mired in complacency.

I feel very optimistic about open source. In some ways, it is the antithesis of the traditional life-cycle models but it has already produced some extremely good systems on which we all depend. There are plenty of lessons here for those who wish to learn.

Finally, this year, I gave 100% for an MSc project for the first time in my life for a superb piece of work so there is much to look forward to.

References

[1] C. Boehm and G. Jacopini. Flow Diagrams, Turing machines, and Languages with only Two Formation Rules". *Communications of the ACM*, 9(5):p.366–371, 1966.

[2] L. Hatton. The t experiments: Errors in scientific software. *IEEE Computational Science and Engineering*, 4(2):27–38, April 1997.

[3] L. Hatton. Bureaucracy, Safety and Software: a potentially lethal cocktail. In C. Dale and T. Anderson, editors, *Making Systems Safer: Proceedings of the 18th Safety-Critical Systems Symposium*, pages p.21–36, London, UK, 2009. Springer.

[4] S.L. Pfleeger and L. Hatton. Do formal methods really work ? *IEEE Computer*, 30(2):p.33–43, 1997.