”Lessons must be learned” - but are they ?

Les Hatton and Anne Rutkowski

“Those who cannot remember the past are condemned to repeat it.”, George Santayana, 1863-1952

In spite of all the software systems we have seen both through these columns and also through our professional experience, periodically, something happens in the world of software engineering which really takes us by surprise. The last time we were placed in this position was following the revelation of software “cheats”, algorithms deliberately introduced into a system with the specific purpose of misleading the general public and certification agencies on the nature of its emissions, (HvG (2016)). This time, we feel we must make some comment over the equally startling revelations emerging over the interaction between software, management and requirements in the sad case of the two Boeing 737 MAX crashes. We stress that it is early days in the diagnosis of what went wrong; the appropriate agencies will report in detail at some stage. However, there is understandably an enormous amount of discussion in the media as the general public gradually become aware, through columns like the Impact series, of just how much software there is in commercial devices. We won’t simply regurgitate other speculations but rather we will compare some disturbing patterns in these two crashes with past incidents in avionic and in other areas where software is rife and is implicated in their failure. What perhaps is most shocking is that the Boeing 737 has a formidable record for safety in the 50 years since its first flight and an important question for the world of avionics is to ask is whether software is really capable of destabilising such an enviable record?

The forensic process of understanding something as traumatic as an aircraft crash is both thorough and time-consuming and investigators will go to extraordinary lengths to determine the nature of the crash. At some stage, we will know precisely what went wrong. As software engineering professionals, we are however more concerned with how resulting engineering lessons from failed systems are integrated into the software profession. Historically, we have always considered safety-critical software to be necessarily deserving of higher standards at all phases of development from the extraction of its requirements, to its careful specification, to its implementation and a rigorous and as exhaustive as possible regime of testing as can be achieved in the time and budget. At the other end of the spectrum of software development, we have the rapid growth of methodologies intended to deliver something of value with as light a development footprint as possible to reduce costs , represented largely by *agile* technologies. We did not expect there to be symptoms of these two polar opposites of development merging. It is hard however to deny that this is happening in the rush to provide increasing amounts of software for example, in autonomous vehicles. We have previously written that conventional software systems typically grow at 20% per annum with safety-related systems somewhat slower at around 10%, (HSvG (2017)), but are we now seeing efforts which perhaps unwittingly attempt to close this gap ?What is really necessary in safety-critical engineering is to preserve vision and to have a detailed knowledge of well-known failure modes so they can be avoided. This is what we mean by good practice. Even a simple and apparently innocuous system can if used in a chain involving criticality, lead to life-threatening circumstances. For example, if a hospital information system fails to contact a patient for a follow-up cancer check, this can lead to a potential delay in life-preserving treatment. As we have said, it is early days for us to comment on the Boeing 737MAX incidents but there is already some evidence that we have seen similar kinds of failure before, and similarity is an important clue in forensic engineering

First of all, for a safety-critical software system to rely on interpreting the input safely from a single external sensor would be considered highly questionable at best, yet this appears to have been done with the Boeing 737MAX MCAS system, (NIF (2019)), with a single "AoA" vane (Angle of Attack) which measures the plane's angle between the airflow and the wing. Single sensor reliance in software has long been a documented problem in the case of the pitot tube or air-speed indicator. Icing of pitot tubes followed by erratic behaviour has been encountered before in aircraft incidents, to the extent that following a number of occurrences between 2008 and 2009 and culminating in the June 2009 Airbus 330 crash in the Atlantic, Airbus issued three Mandatory Service Bulletins requiring that all A330 and A340 aircraft be fitted with *three* pitot tubes, (AF447 (2019)), therefore satisfying one of the most fundamental principles of resilient engineering – redundancy. Indeed this can be carried over into the internal design of the software itself, (Hat (1997)). Even modern fire alarms have redundancy, combining inputs from both visual and chemical sensors to reduce the frequency of false alarms.

There are actually two AoA vanes on a Boeing 737MAX *but it is reported that the software was designed to take a reading from only one of them*, (NIF (2019)). Unfortunately, software engineers tend not to do very well dealing with absurd input values and it has been a well-known source of error in software systems for decades, (Whit (2002)). Worse still, adding an additional AoA vane with software to warn the pilots if there was a disagreement was apparently *an optional extra, requiring further payment*, (NIF (2019)). We find this an unusual practice to say the least for a safety-critical system whereby the customer has to pay extra to buy redundancy, a feature which has been considered good and some would argue indispensable practice in safety-critical software systems for 25 years, (see Mell (1994) for a comprehensive discussion of the contributory role of software in aircraft crashes). We should note that apparently, both of the crashed 737MAXes were delivered *without* this option, a decision that has now been reversed with the March update of the 737MAX software (WSJ 2019b).

Interpreting external inputs safely with software is a recurrent theme in safety-critical system failures, even with multiple sensors. This was for example a factor in the first Tesla fatality where the software failed to distinguish a large white 18-wheel truck and trailer crossing the highway against a bright spring sky in May 2016, (Tesl (2016)).

There is another thread in the 737MAX crashes we would like to address. How does software in a safety-critical system convey both unequivocally and quickly to the operator why it is doing what it is doing? Justification of behaviour to its operator is a *sine qua non* in a safety-critical system. We are accustomed to arguments about software being "more reliable" than human operators, but there are enough incidents involving unexpected software behaviour to justify much more attention to the information flow between software and operator. In 1994, an Airbus A340 displayed a gnomic "Please Wait" message on its Flight Management System during an emergency landing at London's Heathrow airport, (AAIB (1994)); a test pilot described the operating manuals of the MD-11 as having been written "by creatures from another planet" (Drur (1997)), (it is reported that the pilots of the preceding ill-fated Lion Air 737MAX were consulting the aircraft manuals shortly before they crashed, Guar (2019)); in the 1994 Nagoya Airbus crash, the pilots were fighting the TOGA (Take Off Go Around) software which they had inadvertently engaged – the plane was trying to go round whilst the pilots were trying to land, (AvSa (1994)). If we find it so difficult to include the human operator in software-based decisions currently, what then is the future of AI based systems where such justification is much more difficult to realise?

Apparently, a design consideration has been to prevent information overload in the cockpit. Indeed, "One senior Boeing official said the company had decided against disclosing details about the system that it felt would inundate the average pilot with too much information—and significantly more technical data—than he or she needed or could realistically digest." [WSJ 2019b], Information overload is a known problem in our IT dominated world. As we have learned, information overload is not about the amount of information but about the relevance [RutkS 2019]. The next statement from the same WSJ article is more shocking: American Airlines pilots did not experience the problems that Lion Air and Ethiopian Air experienced because "American paid for an additional cockpit warning light that would have alerted them to the problem, while Lion Air and most other airlines didn't." Why indeed was this an optional extra ?

Going from information underload to information overload within seconds is also a known risk [RutkS 2019]. It is for example, experienced in robotic surgery when a surgeon has to grab control from the robot in case of complications. Conversion from robotic surgery to laparotomy (open surgery) occurs in about 4 to 7% percent of the operations [reference to be provided by Anne]. Some wrongly argue that training for surgical conversion can be minimized since it is expected to happen less and less. This approach seems similar to the one Boeing took: "The company had promised Southwest Airlines Co. , the plane's biggest customer, to keep pilot training to a minimum so the new jet could seamlessly slot into the carrier's fleet" [WSJ2019b]. However, as the WSJ noted; "amid the chaos of an aircraft lurching into a steep dive with emergency warnings blaring, it is unrealistic to expect pilots to recognize what is happening and respond almost instantaneously". Although surgeons should receive additional training for laparotomy, we must remember that a surgical team is typically available to respond to such a crisis; in other words there is fallback position, another essential feature of a safety-critical system. This is not always the case in an aircraft in trouble since an emergency team cannot be brought into the cockpit and even connecting experts on the ground via an audio or data conection with the cockpit may not be possible.

A further vital link in the chain of safety critical systems is their certification. In certification, an independent body with great experience in assessing such systems is tasked with assessing their safety or otherwise. These bodies include the FAA for aircraft registered in the USA and EASA for those registered in Europe. The whole idea of independence is that it is free of the usual commercial pressures and timelines so it can assess the system and documentation as presented as sufficiently safe for its intended use.   However, the FAA delegated some certification tasks for the 737MAX to Boeing itself, (WSJ (2019)).

Finally, what do you do when a safety-critical software system is deemed unsafe or is involved in a fatal incident? The software must of course be patched. We as software engineers are used to patches. Indeed in the world of truly "agile" development, they are a never-ending part of not only the development cycle but the active use cycle also. Mobile phone apps are endlessly patching themselves, and the same is true for all commercially used systems, whether they be browsers, word processors or whatever. Frequent patching like this is highly beneficial to the user (who sees a more reliable and perhaps more secure product), and also to the developer, because it allows them to overlap development with active use giving quicker time to market. However is it really appropriate for safety-critical systems and under what conditions ? In short, we probably don't know. Unlike normal software patch delivery times which might be measured even in hours, changing safety-critical software would normally expect to re-engage the certification cycle, particularly after a fatal accident. It is most decidedly not a quick fix. In the case of the 737MAX, Boeing are however promising in an open letter from Boeing's chairman Dennis Mullenberg, "Soon we'll release a software update and related pilot training for the 737 MAX that will address concerns discovered in the aftermath of the Lion Air Flight 610 accident", (Boe (2019)). No doubt the fact that its fleet is grounded is concentrating their minds appropriately, but this seems unusually quick for avionics software. Isn't this at risk of breaking another golden rule of software engineering ? Even in the VW emissions scandal, disabling the software "cheat" appears to have destabilised at least some automobiles, (Mass (2017), just as we predicted in HvG (2016)). Unintended side-effect is an inevitable feature when attempting to update complex software systems. This is a lesson we seem incapable of learning.

Having said that, on the fly fixing of potentially critical systems has already become a feature of autonomous software in motor vehicles such as Tesla, where updates are delivered on a regular basis over the net. We do not know how updates to vehicle software are certified, who by and under what conditions and invite somebody from the motor industry to tell us in one of these columns. When does a patch reduce the risk of re-occurrence of failure ? How was it certified or was this bypassed ? Does rapid incremental patching make things better or worse ? Will we as users be told these important details or are we to be blinded by technology or worst of all, told that all will be well, AI will save us all ?

It is really important for the software engineering community to appreciate how we are to live with the benefits and potential risks of incremental and increasingly frequent updating in safety critical software systems, just as we have had to with commercial non safety-related systems. Yes, we are indeed wallowing in software but blindly trusting a system to keep us safe is not the way forward. We invite anybody working at the coal face in such systems to tell us how they balance the polar extremes of quick response for commercial opportunity against the careful and hard won lessons needed to build systems on which peoples lives depend, and which we so easily seem to forget.

Acknowledgements

We gratefully acknowledge the help of Dr. Deborah Ancell for essential background information.

References

[AAIB 1994] Air Accident Investigation Board (1994), Bulletin 3/95, Ref EW/C94/9/2, available at https://assets.publishing.service.gov.uk/media/422fb2440f0b6134600089b/Airbus_A340-311__G-VAEL_03-95.pdf, accessed 26-Mar-2019

[AF447 2019] https://en.wikipedia.org/wiki/Air_France_Flight_447, accessed 20-Mar-2019

[AvSa 1994] https://aviation-safety.net/database/record.php?id=19940426-0, accessed 26-Mar-2019

[Boe 2019] https://boeing.mediaroom.com/2019-03-18-Letter-from-Boeing-CEO-Dennis-Muilenburg-to-Airlines-Passengers-and-the-Aviation-Community, accessed 20-Mar-2019

[Drur 1997] Drury R.S. (1997) “Flying the MD-11: one pilot’s perspective”, Airways magazine, Sept/Oct 1997, pp 39-49.

[Guar 2019] https://www.theguardian.com/world/2019/mar/20/lion-air-pilots-were-looking-at-handbook-when-plane-crashed, accessed 26-Mar-2019

[Hat 1997] Hatton L. (1997) “N-version design versus one good version”, IEEE Software, 14(6), pp. 71-76

[HvG 2016] Hatton L. and van Genuchten M. (2016) “When Software crosses a line”, IEEE Software 33(1), pp 29-31.

[HsvG 2017] Hatton L., Spinellis D., and van Genuchten M. (2017) “The long term growth rate of evolving software: Empirical results and implications”, Journal of Software: Evolution and Process, 29(5)

[NIF 2019] News In Flight (2019), http://newsinflight.com/2019/03/18/the-updating-changes-to-mcas-software-on-boeing-737max/, 18-Mar-2019, accessed 20-Mar-2019

[Mass 2017] https://www.thecarexpert.co.uk/volkswagen-17000-complaints-dieselgate/, accessed 26-Mar-2019

[Mell 1994] P. Mellor. (1994) “CAD: Computer aided disaster”, High Integrity Systems, 1(2):101-156,1994.

[RutkS 2019] Rutkowski, Anne-Francoise, & Saunders, Carol (2019). Emotional and Cognitive Overload: The Dark Side of Information Technology. Oxon: Routledge. [Tesl 2016] https://www.theguardian.com/technology/2016/jun/30/tesla-autopilot-death-self-driving-car-elon-musk, accessed 26-Mar-2019

[WSJ 2019] https://www.wsj.com/articles/fast-tracked-aircraft-certification-pushed-by-boeing-comes-under-the-spotlight-11553428800, accessed 26-Mar-2019

[WSJ 2019b] How Boeing’s 737 MAX Failed. The plane’s safety systems, and how they were developed, are at the center of the aerospace giant’s unfolding crisis, March, 27, 2019.

[Whit 2002] James A Whittaker (2002), “How to Break Software”, Addison-Wesley Longman, 2002. ISBN 0201796198