Software Quality, Dependability and Safety in Embedded Systems (Invited Talk)

Philip Koopman

Carnegie Mellon University,
Pittsburgh, PA, USA
koopman@cmu.edu

We often trust embedded systems with mission-critical functions, and even our own lives. But the designers of such systems (and especially their managers) are often domain experts who have not been formally trained in software development. While many embedded systems work well, in my design reviews I frequently see problems ranging from the subtle to the catastrophic. I have identified commonly occurring technical, process, and quality assurance issues based on my experience performing more than 135 industry design reviews. Common problems include a lack of embedded-specific software engineering skills, software process gaps, and a failure to appreciate that more than just product-level testing is required to create high quality software. Most of these problems cannot simply be fixed by adopting a tool, but rather require a change of culture and perspective in engineering organizations. All too often, the developers and their management simply don't realize they have gotten in over their heads as their product's software has escalated from performing a simple supporting function to providing make-or-break product functionality.

The Toyota Electronic Throttle Control System (ETCS) is a system deployed in almost a decade of vehicle production that exhibits many of the common problems I have seen in design reviews. It has numerous lapses in following good software practices in general, and safety-critical software practices in particular.

Briefly, the ETCS takes inputs from the driver (for example the accelerator pedal position and brake pedal activation), and has complete control over the throttle position as well as fuel and spark. There are practical scenarios in which a fully open throttle can overpower the brakes in Toyota vehicles. This makes the ETCS a safety-critical throttle-by-wire system.

Mishaps involving the Toyota ETCS have resulted in billions of dollars of costs in the US, including an economic loss class action settlement, a criminal cover-up case, and undisclosed settlements in hundreds of individual death and injury cases. Recalls have been issued – but for mechanical issues rather than for software defects. A redacted NASA report has been made public, as well as some transcripts from the one public trial that featured software safety testimony (including testimony by this author). The jury in that one trial found Toyota liable for a fatal crash based on testimony that alleged software defects were responsible. Toyota has denied that software defects have resulted in this or any other mishap.

While the question as to whether software defects caused the hundreds of other loss events involved in lawsuits remains open, the following observations about the ETCS are for practical purposes uncontested: Applying brakes will not necessarily stop the car if the throttle is commanded wide open (whether by floor mat entrapment or a possible software defect). Toyota did not follow an applicable set of safety guidelines (e.g., the MISRA Development Guidelines), and has not made an argument that their development processes are comparably rigorous to any established safety guideline. A significant amount of testing was performed both at the module and vehicle level but, as one would expect, this testing effort pales in comparison to the exposure of a fleet of millions of deployed vehicles. While there are dual redundant analog signals from the accelerator pedal to the ETCS, they both go through the same A/D converter on the same chip. While the ETCS has two CPU chips, they do not form a proper dual path system. While Toyota did have some coding rules, developers did not always follow their own coding rules, and did not have a formalized (written) waiver process. Static analysis of the ETCS software reveals global variables declared with different types, casts that alter values, condition evaluations with side effects, and uninitialized variables. The ETCS main CPU software has approximately 10,000 global variables, most of which could have been declared "local static" or "file static" to reduce their scope – but weren't. Shared global variables are not all declared "volatile," and shared global variables are not always accessed under the protection of interrupt masks. Moreover, NASA identified a specific concurrency hazard situation with the ETCS. Many ETCS software functions are quite long, and modularity is poor in general. There is no mitigation for stack overflow, and NASA did not find it possible to establish a maximum stack depth due to the presence of recursion. The main CPU can be more than 80% loaded, but beyond that NASA found that timing analysis was too difficult to complete due to, for example, the presence of busy-wait loops and indirect recursion. A watchdog timer is used to monitor average CPU load, but is not able to detect some task deaths. Much of the "paperwork" that is typically associated with a rigorous software development process does not seem to exist, including: defect logs, peer review records, comprehensive test plans, recorded test results, and process quality audit records.