Toyota Unintended Acceleration and the Big Bowl of “Spaghetti” Code

Posted on Thursday, Nov 7th, 2013

Category: *electronic-throttle electronic-throttle-control sudden-unintended-acceleration toyota*

Last month, Toyota hastily settled an Unintended Acceleration lawsuit – hours after an Oklahoma jury determined that the automaker acted with “reckless disregard,” and delivered a $3 million verdict to the plaintiffs – but before the jury could determine punitive damages.

What did the jury hear that constituted such a gross neglect of Toyota’s due care obligations? The testimony of two plaintiff’s experts in software design and the design process gives some eye-popping clues. After reviewing Toyota’s software engineering process and the source code for the 2005 Toyota Camry, both concluded that the system was defective and dangerous, riddled with bugs and gaps in its failsafes that led to the root cause of the crash.

Bookout and Schwarz v. Toyota emanated from a September 2007 UA event that caused a fatal crash. Jean Bookout and her friend and passenger Barbara Schwarz were exiting Interstate Highway 69 in Oklahoma, when she lost throttle control of her 2005 Camry. When the service brakes would not stop her speeding sedan, she threw the parking brake, leaving a 150-foot skid mark from right rear tire, and a 25-foot skid mark from the left. The Camry, however, continued speeding down the ramp and across the road at the bottom, crashing into an embankment. Schwarz died of her injuries; Bookout spent five months recovering from head and back injuries.

Attorney Graham Esdale, of Beasley Allen, who represented the plaintiffs is the first to say that the Bookout verdict – in some measure – rested on those two black skid marks scoring the off- ramp.

“Toyota just couldn’t explain those away,” Esdale said. “The skid marks showed that she was braking.”

The jury was very attentive, despite the technical discussions that dominated the testimony. After the jury learned that the case had been settled, jurors asked Judge Patricia Parrish if they could stay and discuss the trial. A dozen jurors, Judge Parrish, and the plaintiff’s lawyers discussed it. Esdale says that it was obvious from that conversation that the jury was poised to punish Toyota for its conduct and cover-up.

Skid marks notwithstanding, two of the plaintiffs’ software experts, Phillip Koopman, and Michael Barr, provided fascinating insights into the myriad problems with Toyota’s software development process and its source code – possible bit flips, task deaths that would disable the failsafes, memory corruption, single-point failures, inadequate protections against stack overflow and buffer overflow, single-fault containment regions, thousands of global variables. The list of deficiencies in process and product was lengthy.

Michael Barr, a well-respected embedded software specialist, spent more than 20 months reviewing Toyota’s source code at one of five cubicles in a hotel-sized room, supervised by security guards, who ensured that entrants brought no paper in or out, and wore no belts or watches. Barr testified about the specifics of Toyota’s source code, based on his 800-page report. Phillip Koopman, a Carnegie Mellon University professor in computer engineering, a safety critical embedded systems specialist, authored a textbook, Better Embedded System Software, and performs private industry embedded software design reviews – including in the automotive industry – testified about Toyota’s engineering safety process. Both used a programmer’s derisive term for what they saw: spaghetti code – badly written and badly structured source code.

Barr testified:

There are a large number of functions that are overly complex. By the standard industry metrics some of them are untestable, meaning that it is so complicated a recipe that there is no way to develop a reliable test suite or test methodology to test all the possible things that can happen in it. Some of them are even so complex that they are what is called unmaintainable, which means that if you go in to fix a bug or to make a change, you're likely to create a new bug in the process. Just because your car has the latest version of the firmware -- that is what we call embedded software -- doesn't mean it is safer necessarily than the older one….And that conclusion is that the failsafes are inadequate. The failsafes that they have contain defects or gaps. But on the whole, the safety architecture is a house of cards. It is possible for a large percentage of the failsafes to be disabled at the same time that the throttle control is lost.

Even a Toyota programmer described the engine control application as “spaghetti-like” in an October 2007 document Barr read into his testimony.

Koopman was highly critical of Toyota’s computer engineering process. The accepted, albeit voluntary, industry coding standards were first set by Motor Industry Software Reliability Association (MISRA) in 1995. Accompanying these rules is an industry metric, which equates broken rules with the introduction of a number

of software bugs: For every 30 rule violations, you can expect on average three minor bugs and one major bug. Toyota made a critical mistake in declining to follow those standards, he said.

When NASA software engineers evaluated parts of Toyota's source code during their NHTSA contracted review in 2010, they checked 35 of the MISRA-C rules against the parts of the Toyota source to which they had access and found 7,134 violations. Barr checked the source code against MISRA's 2004 edition and found 81,514 violations.

Toyota substituted its own process, which had little overlap with the industry standard. Even so, Toyota's programmers often broke their own rules. And they failed to keep adequate track of their departures from those rules – and the justification for doing so, which is also standard practice. Koopman testified that if safety is not baked into the recipe in the process of creating the product, it cannot be added later.

"You have to exercise great care when you're doing safety critical software. You can't just wing it. And Toyota exercised some care, but they did not reach the level of accepted practice in how you need to design safety critical systems," he said.

One of the biggest safety standards Toyota broke was allowing single point failures within its system. (Single point failure refers to a piece of hardware or software that has complete control over whether a system is safe or not—such as a single-engine airplane.) Koopman testified:

"If there is a single point of failure, by every safety standard I have ever seen, it is by definition unsafe, and no amount of countermeasures, no amount of failsafes will fix that. They will reduce how often it happens, but it won't completely fix it. Because we have millions of vehicles out there, it will find a way to fail that you didn't think of, and it will fail."

Other egregious deviations from standard practice were the number of global variables in the system. (A variable is a location in memory that has a number in it. A global variable is any piece of software anywhere in the system can get to that number and read it or write it.) The academic standard is zero. Toyota had more than 10,000 global variables.

"And in practice, five, ten, okay, fine. 10,000, no, we're done. It is not safe, and I don't need to see all 10,000 global variables to know that that is a problem," Koopman testified.

Other important design process errors Barr and Koopman identified were an absence of a peer code review, and Toyota's failure to check the source code of its second CPU, supplied by Denso —even as executives assured Congress and NHTSA that the cause of UA couldn't be in the engine software.

Barr testified to some of the vehicle behavior malfunctions caused by the death of tasks within the CPU, and concluded that Bookout's UA was more likely than not caused by the death of a redacted-name task, called Task X at trial. Barr dubbed it "the kitchen-sink" task, because it controlled a lot of the vehicle's functions, including throttle control; the cruise control – turning it on, maintain the speed and turning it off – and many of the failsafes on the main CPU.

He was critical of Toyota watchdog supervisor – software to detect the death of a task -- design. He testified that Toyota's watchdog supervisor "is incapable of ever detecting the death of a major task. That's its whole job. It doesn't do it. It's not designed to do it."

Instead, Toyota designed it to monitor CPU overload, and, Barr testified: "it doesn't even do that right. CPU overload is when there's too much work in a burst, a period of time to do all the tasks. If that happens for too long, the car can become dangerous because tasks not getting to use the CPU is like temporarily tasks dying."

Barr also testified that Toyota's software threw away error codes from the operating system, ignoring codes identifying a problem with a task. At trial, Barr said:

And task death, although I focused a lot of task X here, because it does so much and it does throttle control and it does failsafe, it's pretty important, but there is [redacted] tasks and they can die in different combinations. It could be task 3 and task X, or task 3and task 7 and task X, or just task 9. And those can cause an unpredictable range of vehicle misbehaviors. It turns out that unintended acceleration is just the most dangerous thing your car can do when it malfunctions.

Even if you were to dismiss their conclusions as nothing but paid-for expert testimony, Koopman and Barr's assessment about software errors as a possible UA root cause go a long way in explaining so much: how Toyota's system could fail and leave no trace; why we are still seeing UAs in late model Toyota vehicles and why Toyota can't seem to fix it with floor mat and pedal recalls; how it could get away with hiding some of the root causes of UA events for so long.

Their descriptions of the incredible complexity of Toyota's software also explain why NHTSA has reacted the way it has and why NASA never found a flaw it could connect to a Toyota's engine going to a wide open throttle, ignoring the driver's commands to stop and not set a diagnostic trouble code. For one, Barr testified,

the NASA engineers were time limited, and did not have access to all of the source code. They relied on Toyota’s representations – and in some cases, Toyota misled NASA. For example, NASA was under the false belief that Toyota had designed in hardware bit flip protections called Error Detection and Correction Codes, (EDAC). The 2005 Camry for example did not have EDAC, Barr testified, but in an email Toyota told NASA that it did. At trial he said:

NASA didn't know that that wasn't there. It wasn't there in the 2005 Camry. And so if the bit-flip occurred, there would be no hardware mechanism to find it. And if it occurred in a critical value that was not mirrored, there would be no software protections against it. So the conclusion here is that there are critical variables in which bits could flip.

Their testimony explains why it would be near impossible for NHTSA to ever pin an electronic failure on a problem buried in software. NHTSA didn’t even have any software engineers on ODI’s staff during the myriad Toyota UA investigations. They have no real expertise on the complexities that actually underpin all of the safety-critical vehicle functions of today’s cars. It’s as if ODI engineers are investigating with an abacus, a chisel and a stone tablet. One begins to understand the agency’s stubborn doubling, tripling, quadrupaling down on floor mats and old ladies as explanations for UA events.

But even if NHTSA did have this expertise, the software piece is so complex ODI would never have the time or budget to assess an automaker’s source code. This is why we keep harping on the need for NHTSA to write a functional safety regulation – under its own steam or Congressional mandate.

We are posting preliminary drafts of Koopman’s (part 1 and part 2) and Barr's trial testimony, along with Barr's slides – long, but well worth a read for anyone interested in understanding more about embedded software systems in automobiles and how not to design one; where NHTSA went wrong: and the unbelievably shaky software at the foundation of Toyota’s electronic architecture.

Normally, one associates a company’s desire to shield trade secrets with the protection of something valuable. That something, one presumes, is the technology itself -- the secret recipe a company uses in making its product. Rather than protecting the automotive equivalent of formula for Coke, the testimony of Koopman and Barr suggest that Toyota really wanted to hide was its formula for disaster. Consider the contents of a September 2007 email among Toyota employees:

“‘In truth technology such as failsafe is not part of the Toyota's engineering division's DNA,’ ” Barr read in court. “And it continues, ‘But isn't it good that it is recognized as one of the major strengths of Toyota and its system controls industry.’ And then I highlighted also the portion that says, ‘Continuing on as is would not be a good thing.’”