Embedded Software under the Courtroom Microscope – A Case Study of the Toyota Unintended Acceleration Trial

David M. Cummings

dcummings@kellytechnologygroup.com

This article appears in the December 2016 issue of IEEE Technology & Society Magazine, pages 76-84 [DOI: 10.1109/MTS.2016.2618681].

1 Introduction

Product liability trials involving embedded software, such as those in which an automobile driver alleges that faulty embedded software was responsible for an accident, require the same standard of proof as other civil trials. The plaintiff must convince a jury that the defendant (for example, the automobile manufacturer) more likely than not is responsible for the accident [1]-[4].[1] This includes showing that the defendant failed to fulfill their duty, and that the defendant's failure was the cause of the accident (causation) [5]-[12]. The defendant may offer evidence to counter the plaintiff's arguments. For example, if the plaintiff's technical expert testifies that in his/her opinion it is more likely than not that a specific problem in the defendant's embedded software caused the accident, the defendant might identify flaws in the analysis. As will be shown, when a non-technical jury in such a trial is confronted with complex software issues, the verdict they deliver may not be supported by the facts. The jury can get it wrong.

It is well known that nearly all non-trivial software has bugs [13]-[15]. Furthermore, because there are virtually an infinite number of different ways of solving a non-trivial problem using software, one can often find many opportunities for criticizing software quality, sometimes using criteria that are highly subjective. As a result, the plaintiffs in a software trial can be expected to attack the defendant's software by looking for bugs and criticizing the software's quality. If they can find bugs, and show that more likely than not those bugs caused the accident, then they can establish causation. They can also use purported measures of software quality, which may be subjective, to argue that the defendant failed to fulfill their duty to provide software of sufficient quality.

This can present unique challenges in software cases. The plaintiffs' expert can bombard a non-technical jury with criticisms of the defendant's software that sound convincing, but which may or may not be

[1] "More likely than not" is sometimes described as "at least 51 percent of the evidence favors the plaintiff's outcome" [3], or "the evidence shows the defendant is more than 50% likely to be responsible" [4].

justified because of the subjective nature of many such criticisms. He/she can then conclude that because the defendant's software is substandard, it is more likely than not that it was responsible for the accident. The problem with this line of argument is that even if bugs or design flaws are found in the defendant's software, this does not mean the software caused the accident. The plaintiffs may not have fulfilled their responsibility to show a connection between the potential bugs and the accident. The plaintiffs' expert may convince him/herself that they have found the cause of the accident, when all they have really found are potential bugs unrelated to the accident. Alternatively, the plaintiffs' expert may realize that they have not found a credible connection between the potential bugs and the accident, but they are willing to tell the jury that more likely than not those bugs are responsible for the accident. (That would be a violation of ethical standards including IEEE ethical standards [16] and ACM ethical standards [17].) Either way, the jurors are told by an expert that problems were found in the software, and in the expert's opinion it is more likely than not that those problems caused the accident.

In order to illuminate these issues, we will look at a specific case, the 2013 Oklahoma trial in which a jury determined that Toyota's engine control software was to blame for reported unintended acceleration that ended in a fatal accident. We are fortunate in that one of the plaintiffs' software experts made his trial testimony and slides available to the public (see links below). This gives those of us in the engineering community a rare opportunity to get a first-hand look at the technical arguments that were presented. The expert's post-trial commentary included the following:

> …The second round began with an over 750 page formal written expert report by me in April 2013 and culminated this week in an Oklahoma jury's decision that the multiple defects in Toyota's engine software directly caused a September 2007 single vehicle crash that injured the driver and killed her passenger. …In a nutshell, the team led by Barr Group found what the NASA team sought but couldn't find: "a systematic software malfunction in the Main CPU that opens the throttle without operator action and continues to properly control fuel injection and ignition" that is not reliably detected by any fail-safe. …Now it's your turn to judge for yourself. Though I don't think you can find my expert report outside the Court system, here are links to the trial transcript of my expert testimony to the Oklahoma jury and a (redacted) copy of the slides I shared with the jury in Bookout, et.al. v. Toyota. [18]

As the expert suggested, I examined his trial materials and judged for myself. I had a particular interest in this subject, having previously written an opinion piece for the Los Angeles Times about Toyota's investigation into the cause of reported unintended acceleration, and the possibility that software was to blame [19]. Based on the statements made by the expert, including those above, as well as favorable articles in various publications describing his trial testimony, I expected to find a convincing case built on compelling evidence that there was indeed "a systematic software malfunction in the Main CPU" that directly caused the automobile accident. Unfortunately, I found instead a disappointingly flawed theory of causation that is not credible. (The National Highway Traffic Safety Administration [NHTSA] apparently agrees with this assessment [20].)

Thus, this trial serves as an informative case study demonstrating how a jury can be led to an apparently incorrect conclusion about software based on technical arguments that sound convincing, but which do not withstand technical scrutiny. And as we become increasingly reliant on embedded software in our daily lives, for example, with the advent of self-driving cars, trucks, and automotive "autopilot" features, our legal system is likely to face this issue with increasing frequency.[2]

Since my review of the trial materials, my company has been retained by several automobile manufacturers. Confidentiality agreements prevent me from revealing details about those engagements.

2 Background

I will refer below to the expert's trial testimony and slides, which can be found at:

- http://www.safetyresearch.net/Library/Bookout_v_Toyota_Barr_REDACTED.pdf ("Testimony")
- http://www.safetyresearch.net/Library/BarrSlides_FINAL_SCRUBBED.pdf ("Slides")

The expert concluded that the accident was caused by the death of a critical task within the engine control software, which he referred to as "Task X." He testified:

> So to a reasonable degree of engineering certainty, it's my opinion that it was more likely than not, a task X death, possibly in combination with other tasks that occurred that day, causing a loss of throttle control and in [sic] inability to stop the vehicle's full momentum because of the vacuum loss. [Testimony, PDF page 192.]
> See also Testimony, PDF page 187, and Slide 54.

This conclusion does not appear to be supported by the evidence presented at trial.

Before examining the expert's conclusion in detail, some background information about Task X, derived from the above trial materials[3], is required. Also required is background information, derived from the trial materials, regarding a fail-safe called the "Brake Echo Check," which is a key element of the accident theory involving the death of Task X.

Task X is a periodic task that executes multiple times per second on the engine control processor (the main CPU). One of its many responsibilities is to determine the correct throttle angle setting (how far open the throttle should be) based on how hard the driver is pressing on the accelerator pedal (as well as other factors). Therefore, every *n* milliseconds[4] Task X wakes up and, among other things, determines the current accelerator pedal position and sets a throttle angle variable accordingly. The throttle angle variable is then used by another part of the software to set the throttle to the angle specified in that variable.

[2] It remains to be seen if the new federal safety guidelines for autonomous vehicles, not yet released at the time of this writing, will impact this issue. As discussed in [19], stricter automotive safety guidelines are overdue.
[3] I have not seen Toyota's source code. My descriptions of how Toyota's software operates are based on the trial materials.
[4] The Slides suggest that Task X runs every 8 milliseconds [Slide 35].

All the tasks running on the main CPU are managed by a multitasking operating system. The operating system maintains a data structure containing one bit per task. If the bit is set, the task is alive and is subject to task scheduling. If the bit is clear, the task is not running and it will not be scheduled for execution. According to the expert, this data structure (as well as the throttle angle variable) was not protected by software techniques such as mirroring, or by hardware techniques such as error detection and correction. Therefore, if this data structure became corrupted due to a software bug or a single event upset, the corruption would not be detected or corrected.

There is a second processor called the monitor CPU that executes the Brake Echo Check fail-safe software. The Brake Echo Check is designed to behave as follows: If Task X died, and if the driver then stepped on the brake or released the brake, then about 200 milliseconds later the Brake Echo Check on the monitor CPU would detect an inconsistency resulting from the death of Task X on the main CPU, and would force the throttle to idle. About 3 seconds later it would stall the engine. When the throttle is at idle, braking will successfully stop the vehicle.

3 Theory Regarding the Death of Task X

According to the accident theory presented to the jury, the following 3 things had to happen together just prior to the accident:

A. The bit corresponding to Task X in the operating system data structure was somehow flipped from one to zero, resulting in the death of Task X.
B. At the time of this bit flip, the throttle angle variable maintained by Task X contained a large value, corresponding to an open throttle. Because Task X never ran again, the throttle angle variable was stuck at this value, and the throttle remained open.
C. The Brake Echo Check did not work for some reason. When the driver stepped on the brake, the Brake Echo Check did not correctly detect the inconsistency due to the death of Task X and force the throttle to idle. Because the throttle remained open, the driver was unable to stop the vehicle by braking.

Let’s examine each of these:

- A is merely hypothetical. No evidence was presented that this bit was flipped prior to the accident[5], nor was any specific bug in the software identified that caused this bit to flip. Instead, the expert identified what he said were a number of problems in the software that could possibly cause an unspecified memory corruption under some circumstances, which he speculated might possibly include the corruption of this bit. More detail will be provided below.

[5] None of the expert’s testing demonstrated the occurrence of this bit flip, or any bit flip. Rather, he manually flipped bits to force Task X and other tasks to die [Testimony, PDF pages 79-81, 173-174, 182, 184, 238-239]. Regarding his lack of evidence from the accident, he appears to explain this by saying: (a) the software did not perform logging [Testimony, PDF page 190, and Slide 54]; and (b) although there were DTCs (diagnostic trouble codes) stored in battery backed memory on the vehicle, Task X was responsible for recording most of them [Testimony, PDF pages 51, 62-63, 138-140, and Slide 39].

- B appears to be inconsistent with the facts of the accident. As will be shown, the driver was slowing the vehicle on an exit ramp when the expert theorizes Task X died, and therefore the throttle would likely have been at idle.
- C is merely hypothetical. Even if the bit flipped, and even if the throttle was open when that happened, there is no evidence that the Brake Echo Check, executing on a different processor, would not have worked correctly to force the throttle to idle. In fact, as will be shown, the expert did not even speculate at trial why the Brake Echo Check might possibly fail under any circumstances. He simply asserted that it was unreliable without providing any reasons. (He said that his expert report contained reasons, but he couldn't recall any of them [Testimony, PDF page 147].) Moreover, as will be shown, all of the Brake Echo Check testing he described showed it working exactly as designed after the death of Task X.

Thus, this theory is not credible as the likely explanation for the accident for at least the following reasons:

a. It requires the nearly simultaneous occurrence of 2 hypothetical failures, A and C.
b. For hypothetical failure A, no evidence was provided that it occurred at the time of the accident, or under any circumstances. The expert merely speculated that it might possibly occur under some circumstances due to problems he claimed to have identified in the software. No connection was established between any of those claimed problems and the specific bit in question.
c. For hypothetical failure C, no evidence was provided that it occurred at the time of the accident, or under any circumstances. In fact, at trial the expert merely proposed that it occurred without even speculating why.
d. Also for C, all of the testing the expert described showed the Brake Echo Check working exactly as designed.
e. A and C are independent. Hypothetical failure A is associated with software on a different processor than the software associated with hypothetical failure C. What's more, no argument was made that there was a dependence between these two hypothetical failures. The probability of two independent low-probability failures occurring together is much lower (multiplicatively) than the probability of either one occurring alone. Because no evidence was presented that either A or C occurred under any circumstances, it is reasonable to treat each of these as very low probability failures.[6] The probability of both occurring together, then, would be expected to be extremely low (all the more so because not even a speculative reason was offered as to why C might ever occur).
f. In addition, the theory requires a third item, B. But B appears to be contradicted by the facts of the accident. That makes the theory, which already lacks credibility due to the extremely low probability of A and C ever occurring together, even less credible.

[6] For hypothetical failure A, a single event upset was also suggested as a potential cause [Testimony, PDF page 72]. Single event upsets are very low probability failures. The probability of a single event upset affecting a specific bit is even lower.

Can we completely rule out the theoretical possibility that under some circumstances, A, B, and C could conceivably occur together, as improbable as that may be? Maybe not. But the mere possibility that this could theoretically happen is not enough to conclude "to a reasonable degree of engineering certainty" that this is the likely explanation for this accident. (It is all the more improbable because the facts of the accident suggest that B did not occur. Furthermore, no reason was given why C would ever occur under any circumstances.)

The expert emphasized that the Brake Echo Check fail-safe only acts if the driver steps on the brake or releases the brake. He suggested that because of this, if the driver's foot was already on the brake when Task X supposedly died, the Brake Echo Check fail-safe would not act to force the throttle to idle. [Testimony, PDF pages 89, 135-136, 233-234.] As will be shown below, this is irrelevant. If the driver's foot was on the brake when Task X died, then there is no need for the Brake Echo Check fail-safe, because the throttle would already be at idle when Task X died, and braking would stop the vehicle.

4 Alternative Theory Regarding the Death of Task X

The jury was given an alternative theory that did not rely on assumption B (the assumption that the throttle angle variable contained a large value when Task X supposedly died). The expert replaced B with the assumption that, in addition to the memory corruption that flipped the bit (hypothetical failure A), a second memory corruption caused the throttle angle variable to be overwritten with a large value.

The jury was told that memory corruptions are like ricocheting bullets, such that a single memory corruption can affect multiple areas of memory. The alternative theory thus requires either two independent memory corruptions (one that flipped the bit and another that overwrote the throttle angle variable with a large value), or an initial memory corruption that somehow ended up causing these two additional corruptions.

Let's examine both possibilities:

- The occurrence of two independent memory corruptions is hypothetical. The first hypothetical corruption, the bit flip, has already been addressed. Regarding the corruption of the throttle angle variable, the expert did not identify any evidence that such a corruption occurred prior to the accident, or under any circumstances. Nor did he identify any specific bug in the software that caused a corruption of that variable. Instead, he speculated that the problems he claimed to have identified in the software, which he speculated might cause the bit flip, might also cause the corruption of the throttle angle variable. No connection was established between any of those claimed problems and the throttle angle variable.
- The occurrence of an initial memory corruption that somehow led to the two additional corruptions is also hypothetical. The expert did not identify evidence of any initial corruption that led to any other corruption, let alone to the two specific corruptions as required. Nor did he identify any specific bug in the software that caused such an initial corruption. Furthermore:
    I. Not all possible hypothetical memory corruptions would necessarily lead to additional memory corruptions.

II. No explanation was provided as to how the hypothetical bit flip could ever lead to a throttle angle variable corruption, or vice versa, nor was any evidence provided that these two locations were near each other in memory.

Thus, the alternative theory, which relies on one of these two possibilities, is not credible as the likely explanation for the accident for at least the following reasons:

a2. Assuming two independent corruptions: Because no evidence was presented and no specific bugs were identified for either of these two hypothetical corruptions, it is reasonable to treat each as a very low probability occurrence. The probability of both occurring together, then, would be expected to be extremely low.

b2. Assuming two corruptions that resulted from an initial corruption: Because no evidence was presented and no specific bugs were identified for an initial hypothetical corruption leading to any other corruption, it is reasonable to treat that as a very low probability occurrence. The probability of that occurring and then causing hypothetical corruption of not just one but two specific locations would be even lower. It would be lower still for at least reason II above, and still lower for at least the following additional reasons:

- Random corruptions are not enough. The throttle angle variable must be overwritten with a large value, and the bit in question must be zeroed.
- The number of corruptions resulting from the initial corruption must be limited (e.g., pinpointing those two locations but sparing all or most others) such that much of the software continued to function.
- The theory requires particular timing and ordering of the two hypothetical corruptions (more detail in testimony below).

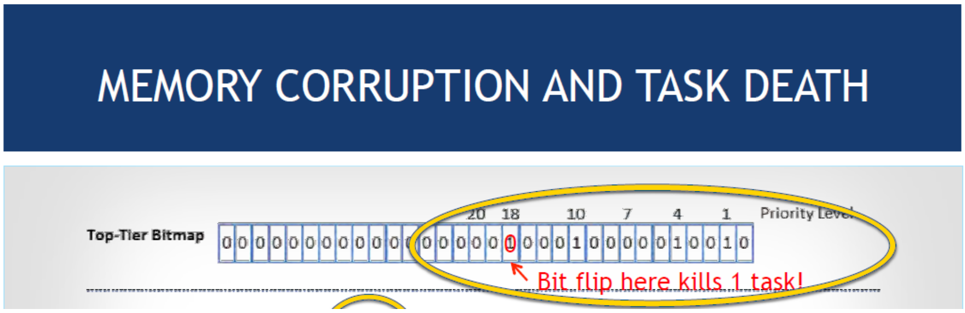
Can we completely rule out the theoretical possibility that under one of the two assumptions above, these two hypothetical memory corruptions might occur together, as improbable as that may be? Maybe not. But the mere possibility that this could theoretically happen is not enough to conclude "to a reasonable degree of engineering certainty" that it is likely to have occurred at the time of the accident.

Furthermore, many scenarios covered by this alternative theory also require that the Brake Echo Check not work correctly. For reasons already discussed, that reduces the credibility of the alternative theory even further.

5 Example Testimony Relevant to the Two Theories

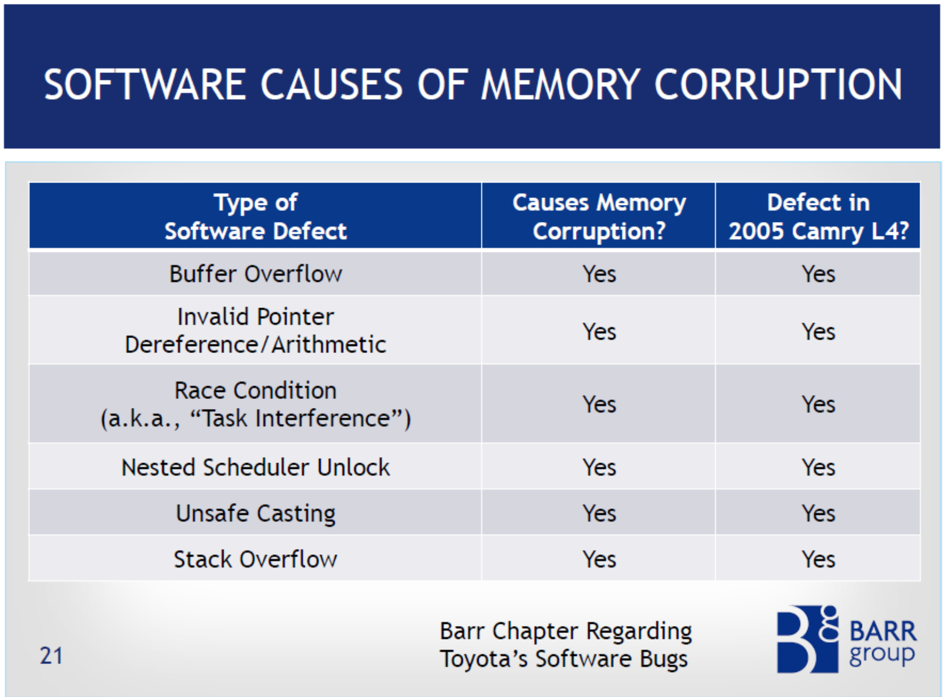
5.1 Bit Flip Hypothesis

The expert presented Slide 19, shown below.


MEMORY CORRUPTION AND TASK DEATH

(See also Slide 37, and Testimony, PDF pages 70-72.)

He identified what he said were a number of problems in the software that could possibly cause an unspecified memory corruption under some circumstances. For example, see Slide 21 below.


SOFTWARE CAUSES OF MEMORY CORRUPTION

Type of Software Defect	Causes Memory Corruption?	Defect in 2005 Camry L4?
Buffer Overflow	Yes	Yes
Invalid Pointer Dereference/Arithmetic	Yes	Yes
Race Condition (a.k.a., "Task Interference")	Yes	Yes
Nested Scheduler Unlock	Yes	Yes
Unsafe Casting	Yes	Yes
Stack Overflow	Yes	Yes

21 Barr Chapter Regarding Toyota's Software Bugs

(This slide appears to incorrectly represent that certain defects were found that caused memory corruption. For example, the expert testified that pointers were not checked before being dereferenced [Testimony, PDF pages 91-92]. This means that if an invalid pointer value were to occur due to a bug or hardware error, the software would not detect the invalid value and prevent it from being used. It does not mean that the expert found an actual occurrence of an invalid pointer value. This slide appears to represent instead that he found an actual occurrence of an invalid pointer value that caused memory corruption. As another example, the expert testified that there was significantly less safety margin in the stack sizing than Toyota thought, resulting in an increased potential for stack overflow [Testimony, PDF pages 105-108, 111-112, and Slides 25-27]. This slide appears to represent instead that the expert found an actual occurrence of stack overflow.)

The expert said that memory corruption could potentially cause task death:

> Well memory corruption is so significant because it's a memory corruption that can cause a task death and task death can cause in a general sense unpredictable results, but in a specific sense, as with task X, cause loss of throttle control and also a disablement of a number of the fail safes. [Testimony PDF page 96]

Nowhere did he identify specific evidence of the bit flip (or any memory corruption) having occurred, nor did he identify any specific bug that caused this bit to flip.

5.2 Open Throttle Hypothesis

The facts of the accident appear to contradict this hypothesis. As confirmed by the expert's answers below, the driver was slowing the vehicle on an exit ramp, which means the throttle was likely at idle (discussed in more detail later):

> Q. And you understand that Ms. Bookout successfully slowed her vehicle and exited on the exit ramp for Texanna Road, correct?
> A. I do.
> Q. And you know that Ms. Bookout told us in her deposition that she applied the brake to slow the car so as to exit on Highway 69, do you understand that?
> A. I do.
> …
> Q. And you understand that the exit ramp there from the Highway 69 to Texanna Road is fairly long, somewhere in the neighborhood of a thousand feet or more?
> A. That is about the distance I understand.
> Q. It's your belief that the alleged unintended acceleration incident began somewhere on the exit ramp, correct?
> A. That's my understanding.
> [Testimony, PDF pages 223-224]

5.3 Unreliable Brake Echo Check Hypothesis

Descriptions of the intended operation of the Brake Echo Check can be found in Testimony, PDF pages 135, 174, 225-226, and 245.

No evidence was presented to support the expert's assertion that the Brake Echo Check was unreliable. He testified that there were unspecified reasons why the Brake Echo Check might not work as designed, but never identified any such reasons or presented any supporting evidence. Furthermore, it worked as designed in every test he discussed:

> A. Just simply from my analysis of the source code, there are several reasons. I put them in my report my [sic] this brake echo check is also nonreliable.
> Q. And why is that?

> A. There is some reasons why it's not -- it's not designed to be 100 percent reliable. There are several reasons, I'd have to look at my report to refresh my memory. [Testimony, PDF page 147] See also Slide 42.
>
> A. In the limited testing that's been done within the essentially infinite space of vehicle and software states, it is true in that limited testing the brake echo check has stepped in if her foot was on the gas pedal and then she transitioned to the brake pedal. However, that doesn't rule out that the brake echo would not act for a number of reasons. [Testimony, PDF page 225] See also Testimony, PDF pages 233-234.

What's more, whether or not the Brake Echo Check works reliably is irrelevant if the throttle was not open when Task X supposedly died (and as discussed, the throttle would likely have been at idle). If the throttle was at idle, then the driver would have been able to stop the car by braking.

Let's examine all 3 possible cases of driver foot position at the time of the hypothesized bit flip:

i. The driver's foot was pressing the gas pedal when Task X supposedly died, or
ii. the driver's foot was pressing the brake pedal when Task X supposedly died, or
iii. the driver's foot was on neither pedal when Task X supposedly died.[7]

As shown below, case ii would not require the Brake Echo Check, and case iii would only require the Brake Echo Check if the supposed death of Task X occurred during a small timing window. In all cases, the driver would be able to stop the vehicle by braking:

i. In this case, when the driver subsequently stepped on the brake, the Brake Echo Check would have forced the throttle to idle 200 milliseconds later. This would have ensured that the vehicle could be stopped by braking.
ii. In this case, the driver would have first taken her foot off the gas pedal. Soon after that, Task X would have executed (it executes multiple times per second), recognized that the gas pedal was not depressed, and set the throttle angle variable to idle. By the time the driver stepped on the brake, the throttle angle variable would be at idle. Then when Task X supposedly died, the throttle angle variable would be left at idle. Braking would have stopped the vehicle, without requiring the Brake Echo Check.[8]
iii. This is similar to the preceding case. Soon after taking her foot off the gas, Task X would have executed, recognized that the gas pedal was not depressed, and set the throttle angle variable to idle. When Task X supposedly died, the throttle angle variable would be left at idle. Subsequent braking would have stopped the vehicle, without requiring the Brake Echo Check.

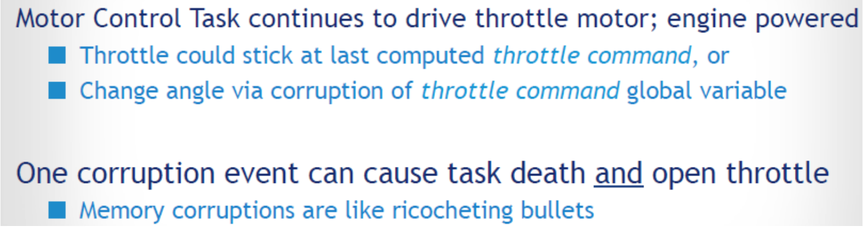
[7] The driver pressing both pedals simultaneously is explicitly excluded by the expert. He assumed there was no "pedal misapplication, human mistake." [Testimony, PDF pages 187-188, and Slide 54.]

[8] Theoretically it might be possible for the driver to take her foot off the gas and then step on the brake so quickly that Task X does not have a chance to execute between those 2 events. Whether this is possible would depend on how frequently Task X executes and how long it takes to move one's foot from the gas to the brake. If Task X executes every 8 milliseconds, it would seem impossible, based on studies of driver braking (e.g., [21]). In any event, no argument was presented that this occurred.

> There could be a small timing window (milliseconds) between when the driver takes her foot off the gas and when Task X executes. If Task X were to die within that timing window, before it had a chance to execute and set the throttle angle variable to idle, then that is the same as Task X dying while the driver was pressing on the gas, which is addressed by case i.

5.4 Two Hypothesized Memory Corruptions

The expert presented Slide 37, shown below.



Motor Control Task continues to drive throttle motor; engine powered

- Throttle could stick at last computed *throttle command*, or
- Change angle via corruption of *throttle command* global variable

One corruption event can cause task death and open throttle

- Memory corruptions are like ricocheting bullets

He testified as follows:

> A. … And so, either if task X is dead, you can get a stuck throttle, which is the last calculated command, or the last computed one over here, or it can change it to a corrupt value through an additional memory corruption.
>
> …
>
> Q. Does the task death of X in that scenario involving the throttle angle variable have to occur first or after the memory corruption of the throttle angle variable?
>
> A. If they are close in time, the two memory corruptions are close in time, it could be an either/or. If task X was dead for a while though and then the second memory corruption happened some time later, then it could also happen that way. So if the two corruptions happen close in time, which is likely when you have memory corruption, often it's not just a single -- when it's a software bug or even hardware bit flip, it can be ricochet and bounce around like a bullet inside, and so it can cause multiple memory locations to be damaged. And so that can begin small and grow over time. And so, if they both happen right about the same time, it could be that the throttle command is corrupted first and then the task dies. But there's more time opportunity the other way. [Testimony, PDF pages 130-132]
>
> See also Testimony, PDF page 56.

He did not identify specific evidence of any memory corruption having occurred, let alone both specific corruptions (the bit flip and the overwriting of the throttle angle variable) having occurred together. Nor did he identify any specific bug that caused either of these hypothesized corruptions.

6 Related Testimony and Slides

There are a number of trial slides that apparently are intended to provide evidence for the expert's accident theories, but in fact they do not. For example:

- Slide 20: This slide shows a test in which killing Task X during cruise control “resume" resulted in unintended acceleration until the brake was pressed, because Task X is responsible for detecting when the cruise control set speed has been reached [Testimony, PDF pages 81-82, 88, 260-263]. This test is not relevant to the expert’s accident theories, because by his own testimony, cruise control was not enabled at the time of the accident [Testimony, PDF page 223]. Furthermore, this test shows that even if Task X died during cruise control “resume,” stepping on the brake would close the throttle via the Brake Echo Check and stop the vehicle [Testimony, PDF pages 82-83, 88-89].
- Slide 53: This slide shows a test in which the driver had been pressing on both the brake and the accelerator for 20 seconds at the time Task X was killed [Testimony, PDF pages 184-186]. It demonstrates that if Task X were to die while both pedals were pressed, the Brake Echo Check would not close the throttle until the driver took her foot off the brake. But this is not what happened in this accident, according to the expert’s theories. As discussed previously, he assumed there was no “pedal misapplication, human mistake.” Note also that the tests shown in slides 20 and 53 are the only tests involving Task X death presented by the expert.
- Slide 49: This slide says that the expert found what “NASA sought,” including that a “single memory corruption results in UA.” Similarly, at Testimony PDF page 193, when discussing single points of failure, the expert said “we've demonstrated that a single byte can cause a UA that can go on until you run out of fuel.” These statements do not accurately reflect the expert’s accident theories, which require more than just a single memory corruption. As discussed previously, his first theory requires a memory corruption (a specific bit flip) **plus** a simultaneous failure of the Brake Echo Check. And his alternative theory requires the same memory corruption **plus** a second memory corruption (the throttle angle variable) **plus,** in many scenarios, yet a third simultaneous failure (the Brake Echo Check). Neither theory is satisfied by just a single memory corruption without at least one other simultaneous failure.

The expert presented many slides criticizing Toyota’s software quality. See, for example, Slides 11, 14, 23-25, 27-33, 35, 38, 39, 41, 43-48, 50-52. But he did not provide evidence for the occurrence of the specific failures required by his accident theories.

A second expert also offered testimony criticizing Toyota’s software quality [22]-[24], although he did not examine Toyota’s source code [25]-[26]. That expert did not provide an opinion of his own regarding causation, nor did he provide an opinion about the first expert’s causation theories [27].

7 Causation not Demonstrated

The jury was told that “to a reasonable degree of engineering certainty, it was more likely than not” that Task X death caused the accident. Two theories were presented to support this conclusion, but neither theory was credible. Nonetheless, the jury accepted the conclusion.

The message provided to the jury by the plaintiffs was twofold:

- The defendant's software was poorly designed and full of bugs.
- These bugs likely caused the accident.

As with all non-trivial software, there likely were bugs in the defendant's software. Perhaps the plaintiffs found some of them. But this does not mean that the accident was caused by any of the purported bugs. Causation was not demonstrated.

Due to the nature of software, it seems likely that this same scenario will play out in future embedded software trials. A key to ensuring that an informed decision will be reached by the jury in such a trial is for both plaintiffs and defendants to adequately address the second bullet point above, causation. Otherwise, the jury can be convinced that causation has been demonstrated when in fact all that has occurred is that the quality of the defendant's software has been criticized, perhaps legitimately and perhaps not. This appears to be what happened in this trial.

8 References

[1] "Burden of Proof," *Cornell University Law School Legal Information Institute*, https://www.law.cornell.edu/wex/burden_of_proof.

[2] "Preponderance," *Cornell University Law School Legal Information Institute*, https://www.law.cornell.edu/wex/preponderance.

[3] "Evidentiary Standards and Burdens of Proof, " *Justia*, https://www.justia.com/trials-litigation/evidentiary-standards-burdens-proof/.

[4] "Legal Standards of Proof," *Nolo*, http://www.nolo.com/legal-encyclopedia/legal-standards-proof.html.

[5] "Actual and Proximate Cause," *Justia*, https://www.justia.com/injury/negligence-theory/actual-and-proximate-cause/.

[6] "Actual Cause," *Cornell University Law School Legal Information Institute*, https://www.law.cornell.edu/wex/actual_cause.

[7] "But-for test," *Cornell University Law School Legal Information Institute*, https://www.law.cornell.edu/wex/but-for_test.

[8] "Proximate cause," *Cornell University Law School Legal Information Institute*, https://www.law.cornell.edu/wex/proximate_cause.

[9] "Elements of a Negligence Case," *FindLaw/Thomson Reuters*, http://images.findlaw.com/optimost/accident-injury-law/elements-of-a-negligence-case-2.html.

[10] "liability," *Law.com*, http://dictionary.law.com/Default.aspx?selected=1151.

[11] “proximate cause,” *Law.com*, http://dictionary.law.com/Default.aspx?selected=1669.

[12] “proximate cause,” *The Free Dictionary*, http://legal-dictionary.thefreedictionary.com/proximate+cause.

[13] N. Leveson, “White Paper on the Use of Safety Cases in Certification and Regulation,” 2011, p. 6, http://sunnyday.mit.edu/SafetyCases.pdf.

[14] D. Pogue, “5 Most Embarrassing Software Bugs in History,” *Scientific American*, Nov. 1, 2014, http://www.scientificamerican.com/article/pogue-5-most-embarrassing-software-bugs-in-history/.

[15] C. Metz, “Is Microsoft to Blame?” *PC Magazine*, August 3, 2004, http://www.pcmag.com/article2/0,2817,1618393,00.asp.

[16] IEEE Code of Ethics, #3 and #9, http://www.ieee.org/about/corporate/governance/p7-8.html.

[17] ACM Software Engineering Code of Ethics and Professional Practice, #1.06 and #4.03, http://www.acm.org/about/se-code.

[18] M. Barr, “An Update on Toyota and Unintended Acceleration,” *Embedded Gurus*, October 26, 2013, http://embeddedgurus.com/barr-code/2013/10/an-update-on-toyota-and-unintended-acceleration/.

[19] D. Cummings, “Haven't found that software glitch, Toyota? Keep trying,” *Los Angeles Times online*, March 11, 2010, http://www.latimes.com/news/opinion/opinionla/la-oew-cummings12-2010mar12,0,2595172.story.

[20] National Highway Traffic Safety Administration, Denial of Motor Vehicle Defect Petition, May 1, 2015, http://www.safetyresearch.net/Library/INFD-DP14003-61928.pdf, pp. 11-15.

[21] G. M. Fitch, et al., “Driver Braking Performance to Surprise and Expected Events,” in *Proc. of the Human Factors and Ergonomics Society Annual Meeting*, Sep. 2010, pp. 2076-2080, https://www.researchgate.net/publication/266491470_Driver_Braking_Performance_to_Surprise_and_Expected_Events.

[22] “Transcript Of Morning Trial Proceedings,” October 13, 2013, http://www.safetyresearch.net/Library/Koopman%2010-11-13%20a.m.PDF.

[23] “Transcript Of Afternoon Trial Proceedings,” October 13, 2013, http://www.safetyresearch.net/Library/Koopman%2010-11-13%20p.m.pdf.

[24] P. Koopman, “A Case Study of Toyota Unintended Acceleration and Software Safety,” September 14, 2014, https://users.ece.cmu.edu/~koopman/pubs/koopman14_toyota_ua_slides.pdf.

[25] Transcript Of Morning Trial Proceedings, October 13, 2013, [22], pages 23, 33.

[26] Transcript Of Afternoon Trial Proceedings, October 13, 2013, [23], pages 41, 44-45.

[27] Transcript Of Afternoon Trial Proceedings, October 13, 2013, [23], pages 80-82, 111.