BOOKOUT V. TOYOTA

2005 Camry L4
Software Analysis

Michael Barr
MICHAEL BARR
Embedded Software Expert

Electrical Engineer (BSEE/MSEE)

Experienced Embedded Software Developer
- Named inventor on 3 patents

Consultant & Trainer (1999-present)
- Embedded Software Process and Architecture for reliability
- Various industries (e.g., pacemakers, industrial controls)

Former Adjunct Professor
- University of Maryland 2000-2003 (Design and Use of Operating Systems)
- Johns Hopkins University 2012 (Embedded Software Architecture)

Served as Editor-in-Chief, Columnist, Conference Chair

Author of 3 books and 65+ articles/papers
BOOKS BY MICHAEL BARR

1. **Programming Embedded Systems with C and GNU Development Tools**
   - 1st ed.: 1999; 2nd ed.: 2006

2. **EMBEDDED SYSTEMS DICTIONARY**
   - 1st ed.: 2003

3. **Embedded C Coding Standard**
   - 1st ed.: 2008; 2nd ed.: 2012
“Embedded Systems”
- Electronics + software for a dedicated purpose
- Many billion more new embedded systems each year
  microwave ovens, digital watches, pacemakers, thermostats
  You are surrounded by them (like it or not; safe or not)

Embedded systems in cars
- Modern cars contain networks of embedded computers!
  Anti-lock brakes, airbags, speedometer, GPS, radio, ...
- Some carmakers brag over 100 microprocessors inside!
  Each headlight, each mirror, each seat, ...

Barr Chapter Regarding
Toyota’s Operating Systems
Access to Toyota’s “electronic throttle” source code

- In a secure room in Maryland
- Subject to confidentiality agreements
- For vehicle models with ETCS spanning ~2002-2010 model years
  Camry, Lexus ES, Tacoma, and others

Approximately 18 months of calendar time with code

- By a very experienced team of embedded systems experts
  Including 3 other engineers from Barr Group
- Building upon NASA’s earlier source code review; digging deeper

NASA must reach a clear-cut conclusion by the end of August.
So they are under a fair amount of pressure.
```c
int larger_of(int a, int b)
{
    if (a > b)
    {
        return a;  /* a contains the larger value */
    }
    else
    {
        return b;  /* b contains the larger value */
    }
}
```
ELECTRONIC THROTTLE CONTROL
TOYOTA’S ENGINE CONTROL MODULE (ECM)

Main CPU ("V850") contains software

Monitor Chip ("ESP-B2") contains software
Not all embedded systems can kill or injure people ...

- Those that can do harm are “safety-critical systems”

What could possibly go wrong?

- A glitch in the electronics (*random hardware faults will happen*)
- A bug in the software (*any reasonably complex software has bugs*)
- An unforeseen gap in the intended safety features
- Or all three: glitch activates bug and that slips thru safety gap

Safety cannot be an afterthought; must be designed in

- Redundancy and fault containment are key

Barr Chapter Regarding
Toyota’s Watchdog Supervisor
ELECTRONIC THROTTLE CONTROL (ETCS)

“Toyota ETCS-i is an example of a safety-critical hard real-time system.”
- NASA, Appendix A, p. 118
SUMMARY OF 2005 CAMRY L4 CONCLUSIONS

Toyota’s ETCS source code is of unreasonable quality

- Toyota’s source code is **defective and contains bugs**
  - Including bugs that can cause unintended acceleration
- Code quality metrics predict presence of additional bugs

Toyota’s fail safes are defective and inadequate

- “House of cards” safety architecture
  - Random hardware and software faults are a fact of life

Misbehaviors of Toyota’s ETCS are a cause of UA
UNINTENDED ACCELERATION (UA)

I use the same definition as NHTSA and NASA:
- “any degree of acceleration that the vehicle driver did not purposely cause”

1 In this report, “unintended acceleration” refers to the occurrence of any degree of acceleration that the vehicle driver did not purposely cause to occur. Contrast this with the term “sudden acceleration incident,” which refers to “unintended, unexpected, high-power accelerations from a stationary position or a very low initial speed accompanied by an apparent loss of braking effectiveness.” An Examination of Sudden Acceleration, DOT-TSC-NHTSA-89-1 at v. As used here, unintended acceleration is a very broad term that encompasses sudden acceleration as well as incidents at higher speeds and incidents where brakes were partially or fully effective, including occurrences such as pedal entrapment by floor mats at full throttle and high speeds and incidents of lesser throttle openings at various speeds. NHTSA, p. vi

I also use the phrase “loss of throttle control”
- Throttle controls airflow, which controls engine power
The NESC team identified two hypothetical ETCS-i failure mode scenarios (as opposed to non-electronic pedal problems caused by sticking accelerator pedal, floor mat entrapment, or operator misapplication) that could lead to a UA without generating a diagnostic trouble code (DTC): specific dual failures in the pedal position sensing system and a systematic software malfunction in the main central processor unit (CPU) that is not detected by the monitor system.

The second postulated scenario is a systematic software malfunction in the Main CPU that opens the throttle without operator action and continues to properly control fuel injection and ignition.

Because proof that the ETCS-i caused the reported UAs was not found does not mean it could not occur. However, the testing and analysis described in this report did not find that TMC

Due to system complexity which will be described and the many possible electronic hardware and software systems interactions, it is not realistic to attempt to “prove” that the ETCS-i cannot cause UAs. Today’s vehicles are sufficiently complex that no reasonable amount of analysis or testing can prove electronics and software have no errors. Therefore, absence of proof that the ETCS-i has caused a UA does not vindicate the system. From calendar year 2005 to 2010 TMC
THERE ARE DEFECTS IN TOYOTA’S ETCS

2005 Camry L4 source code and in-vehicle tests confirm:

- Some critical variables are not protected from corruption
  - Mirroring was not always done
  - NASA didn’t know this (*believed mirroring was always done*)
  - No hardware protection against bit flips
  - NASA didn’t know this (*was told main CPU’s RAM had EDAC*)

- Sources of memory corruption are present
  - Stack overflow can occur
  - NASA didn’t know this (*was told stack less than half used*)
  - There are software bugs
  - NASA found bugs (*and Barr Group has found others*)

Thus Toyota’s ETCS software can malfunction ...
ETCS SOFTWARE MALFUNCTION
SOFTWARE MALFUNCTIONS HAPPEN

All kinds of embedded systems experience partial software malfunction from time-to-time

- e.g., most other apps working, but phone calls go direct to voice mail
- “Have you tried rebooting it?”

The 2005 Camry L4 software has a set of 24 “apps” (called “tasks”)
- All are meant to be running always
TOYOTA’S OPERATING SYSTEM (OSEK)

Task 1  Task 2  ...  Task N

Operating System Software

Critical Data Structures

Main CPU Hardware

Barr Chapter Regarding
Toyota’s Operating Systems
OSEK’S CRITICAL DATA STRUCTURES

Top-Tier Bitmap

Middle-Tier Array

Bottom Tier Queues

Barr Chapter Regarding Toyota’s Operating Systems
MEMORY CORRUPTION AND TASK DEATH

Barr Chapter Regarding Toyota’s Operating Systems
EXAMPLE OF UNINTENDED ACCELERATION

Representative of task death in real-world

Dead task also monitors accelerator pedal, so loss of throttle control
  ✓ Confirmed in tests

When this task’s death begins with brake press (any amount), driver must fully remove foot from brake to end UA
  ✓ Confirmed in tests

Source: Loudon Vehicle Testing
<table>
<thead>
<tr>
<th>Type of Software Defect</th>
<th>Causes Memory Corruption?</th>
<th>Defect in 2005 Camry L4?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer Overflow</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Invalid Pointer Dereference/Arithmetic</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Race Condition (a.k.a., “Task Interference”)</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Nested Scheduler Unlock</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Unsafe Casting</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Stack Overflow</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
**SPAGHETTI CODE DEFINED**

- Difficult to follow data/control paths
- Bugs likely to appear when modified
- Unnecessarily complex

**space**  
*n.* A logic 0 on an RS-232 link. Any voltage between +3 and +25 V. See also mark.

**spaghetti code**  
*n.* Incomprehensible source code, typically including apparently meaningless jumps or gos or a high degree of unnecessary coupling between modules.

**spawn**  
*v.* To create a new thread of execution.

**SPDT**  
(as letters) **abbr.** A type of switch that has one actuator (pole) that connects to one of two contacts. Short for Single Pole, Double Throw. Used to select one of two conditions. Compare to SPST.

**spec**  
*(speck)* **abbr.** See specification.

---

3. Software assembly for power train ECU

After the 4th Steering Committee, rebuilding of engine control and actions for software assembly were started.

(1) Achievements

① Identification of current issues with software assembly ..... Ongoing
- There are C sources for which there is no specification document. (e.g., communication related)
- Specification document and C source do not correspond one-to-one. (e.g., cruise, communication related)

② Activities to improve the spaghetti-like status of engine control application were started.
(Control structure reform has already started in Engine Div. In coordination with this, software structure reform will be carried out. As a first step, it has been decided to transfer two employees from Engine Div. and carry out trial with purge control.)

Because structure design is not being implemented, a "spaghetti" state arises, both TMC and suppliers struggle to confirm overall situation

Without care, systems can quickly get too big and complex, and like dinosaurs, will eventually go extinct.
TYPES OF SPAGHETTI CODE

Data-flow spaghetti
- Complex coupling between software modules and between tasks
- Count of global variables is a software metric for “tangledness”
  - 2005 Camry L4 has >11,000 global variables (NASA)

Control-flow spaghetti
- Many long, overly-complex function bodies
- Cyclomatic Complexity is a software metric for “testability”
  - 2005 Camry L4 has 67 functions scoring >50 (“untestable”)
  - The throttle angle function scored over 100 (unmaintainable)

Barr Chapter Regarding Toyota’s Code Complexity
STACK ANALYSIS FOR 2005 CAMRY L4

OSEK Data + Recursion

4,096 bytes

94% (vs. the 41% Toyota told NASA!)

Recursion violates a MISRA-C rule
(1998: #70; 2004: #16.2)

1,024 bytes

Barr Chapter Regarding Toyota’s Stack Analysis
NASA was concerned about possible stack overflow...

- Deeply nested recursion could exhaust the stack space, leading to memory corruption and run-time failures that may be difficult to detect in testing.

The question, then, is how to verify that the indirect recursion present in the ETCS-i does in fact terminate (i.e., has no infinite recursion) and does not cause a stack overflow.

For the case of stack overflow, the CPU in the ETCS-i does not have protected memory, and therefore a stack overflow condition cannot be detected precisely. It is likely, however, that overflow would cause some form of memory corruption, which would in turn cause some bad

It is not clear what impact recursion has with respect to the larger UA problem. Whether one recursive loop or two, there are other sites of recursion in the ETCS-i that remain unanalyzed.

... and NASA didn’t know there was so little safety margin!

Faced with this limitation, Toyota added an extra margin of safety allocating 4096 bytes for the ETCS-i stack—more than double the...

NASA, Appendix A, pp. 20, 129-134
Toyota botched its worst-case stack depth analysis

- Missed function calls via pointers *(failure to automate)*
- Didn’t include any stack use by library and assembly functions
  
  Approximately 350 functions ignored
- HUGE: Forgot to consider OS stack use for context switching!

On top of that... Toyota used dangerous recursion

absence of recursive procedures, which is standard in safety critical embedded software.

And... Toyota failed to perform run-time stack monitoring

- A safety check that the cheaper 2005 Corolla ECM had!
“OSEK” is an international standard API
- Specifically designed for use in automotive software
- Multiple suppliers of OSEK operating systems
- Compliance tests ensure compatibility across versions

But Toyota’s Rx-OSEK850 version is **non-standard!!!**
- Was not certified as OSEK compliant
- Certified products for V850 were available by 2002
TOYOTA FAILED TO COMPLY WITH STANDARDS
Automotive Industry Coding Guidelines

MISRA-C - motor industry software reliability coding rules for C

- By 2004, “the successes and global use of MISRA-C across automotive, aerospace, medical, and other industries has been staggering.”
- “In Japan, we have worked with representatives of JSAE, JAMA, …”

From 2002-2004, Toyota said in public they followed MISRA-C

- But NASA reported > 7,000 violations of some of the rules (p. 29)
- I checked the full set and found > 80,000 in violations in 2005 Camry L4

Toyota’s coding standard only has 11 MISRA-C rules

- And 5 of those are violated in the actual source code
VIOLATING CODING RULES CAUSES BUGS

In the words of Toyota itself:

- Estimate the number of bugs, using evaluation tool
  - (Static code checker (MISRA-C))

- Measure the number of rule violations and the value of software complexity (f1)

VANALFEN006972 (Kawana, 2004)
TOYOTA FAILED TO COMPLY WITH STANDARDS
Internal Coding Standards

Toyota maintains a set of company internal coding rules

- Specifically for “power train” ECM software developers to follow
  
  Mr. Ishii’s statement about 50% MISRA-C overlap was found false

- NASA reported Toyota didn’t follow some of its rules (p. 22)
  
  I found at least 32% of Toyota’s coding rules were violated

Enforcement is the most important part of having a rule

- Demonstrates lack of engineering discipline at Toyota

- Part of a larger pattern of inadequate software process/oversight
  
  Inadequate and untracked peer code reviews

  No bug-tracking system

Barr Chapter Regarding Toyota’s Coding Standards
A: When it comes to software, there are going to be bugs, and [that] is the case not just with Toyota but with [any] software in the automotive industry and any software. So the issue is not whether or not there is a bug but rather is the bug an important material bug.

- Ishii 5/24/12 Deposition, p. 91

Indeed there are bugs, including “important material bugs”
NASA’S SOFTWARE AREAS OF CONCERN

NASA, Appendix B, pp. 36-39

= Defects Found by Barr Group

Figure B7-1. Software Error Fishbone Diagram

Barr Chapter Regarding Task Death and UA
TOYOTA’S DEFECTIVE “SAFETY LAYERS”

Layer 1: Mirroring of Critical Variables
- Barr Chapter Regarding Toyota’s Memory Protections

Layer 2: DTCs and Fail-Safe Modes
- Barr Chapter Regarding Toyota’s Fail-Safe Modes

Layer 3: Watchdog Supervisor
- Barr Chapter Regarding Toyota’s Watchdog Supervisor

Layer 4: ESP-B2 Monitor CPU
- Barr Chapter Regarding Toyota’s Monitor CPU
Toyota’s engineers sought to protect numerous variables against software- and hardware-caused corruptions
- e.g., by “mirroring” their contents in a 2nd location

But **FAILED TO MIRROR** several key critical variables
- OSEK’s critical internal data structures
- THE target throttle angle global variable!
  - Commands a part of the software to open the throttle
    - Recalculated every 8 ms (when the tasks are all alive)
  - Corruption is indistinguishable from a driver gas pedal press!
THROTTLE COMMAND DESIGN

Task X → write throttle command (e.g. 20%) → Motor Control Task → read

accelerator

Barr Chapter Regarding Toyota’s Software Architecture
Task X death causes loss of throttle control by driver
- Changes at the accelerator pedal have no effect on throttle angle
- Cruise control switches have no effect

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
TOYOTA’S DEFECTIVE THROTTLE CONTROL

Death not Detected

Memory Corruption

unmirrored command (e.g. 50%)

Motor Control Task

“Fail-Safes” Monitoring This Portion Only (no knowledge of driver’s actual intent)

accelerator

Barr Chapter Regarding Toyota’s Software Bugs
NASA talks about 5 fail-safe modes (pp. 79-83)
- Limp home modes 1-3 (*degrees of gas pedal sensor mistrust*)
- Idle mode fuel cut (*2,500 rpm limit at idle*)
- Engine off (*via several different “class 2” failures*)

However, all 5 fail-safes are in same Task X
- Throttle control and fail-safes in same fault containment region
  Unreasonable design; alternative structures well-known

Most diagnostic trouble codes need Task X too!
A “watchdog timer” is hardware to auto-reset software

- Healthy software should periodically “check-in” to prevent reset

With multiple tasks, health of all tasks must be checked
Toyota’s watchdog supervisor design is unreasonable
- Incapable, ever, of detecting death of majority of tasks
- Incapable of properly and reliably detecting CPU overload
- Allows vehicle misbehavior due to overloads lasting up to 1.5s
- Resets the watchdog timer hardware in a timer tick ISR
- Explicitly ignores and discards most operating system error codes

Ignoring error codes violates a MISRA-C rule (1998: #86; 2004: #16.10)

Reasonable design alternatives were well known
- Indeed the primary purpose should’ve been to detect task death
- 2005 Prius (HV-ECU) watchdog is better
“System Guards”
- All (3) useless after Task X death (don’t know driver intent)

“Brake Echo Check”
- Depends on the driver to take action—after UA has already begun!
  - Sometimes a counter-intuitive/dangerous action
  - Clearly this is not a “designed” fail-safe for UA or task death
- Takes the wrong actions (*should’ve reset ECM not stalled car*)
- Not 100% reliable

Does not detect all main CPU malfunctions
A: With respect to [the monitor CPU], the development process is completely different. When it comes to the source code that would be embedded in [the monitor CPUs] we, Toyota, don’t receive them. ... there would not be a design review done on the software.

Q: Now, the monitoring software for the electronic throttle control system is in the [] ESP-B2 chip; correct?

A: Yes.

- Ishii 5/24/12 Deposition, pp. 36-37
The critical “monitor CPU” that checks the main CPU has never been independently reviewed

- Toyota doesn’t even have a copy of the source code
- NASA didn’t review that critical system component either ESP-B2 source code was not provided to NASA

Barr Group has reviewed Denso’s ESP-B2 source code

- Monitor CPU for 2005-2009 Camry L4 (and some other models)
But ESP-B2 monitor CPU could have included a proper UA defense:

- IF (driver is braking & throttle is not closing) THEN reset ECM
  
  *Something is not right with the main CPU when that happens!*

  Resets of main CPU barely noticeable at speed (brief rpm drop)

- CRITICAL to ending UA in vehicles with potential vacuum loss

Per car cost to add this safety feature is $0.00 (it’s just bits)

- There was enough memory and CPU bandwidth for these instructions
- All of the required electrical inputs and outputs were already present
- In line with E-Gas Level 3 recommendations
TOYOTA’S DEFECTIVE SOFTWARE PROCESS

FMEA was incomplete; single points of failure are present
  ■ Because: Toyota didn’t adopt a formal safety process
Peer reviews not done on OS code and ESP-B2 code
  ■ Because: Toyota didn’t perform code reviews; used non-standard OSEK
Toyota’s own “power train” coding standard not enforced
  ■ Because: Toyota didn’t follow through with software suppliers

Watchdog supervisor doesn’t detect most task’s deaths
  ■ Generally costs less to push the limits than upgrade to faster CPU
No EDAC protection against hardware bit flips
  ■ Generally costs less to make memory chips without EDAC
    If confident, why let NASA believe there was EDAC?
TOYOTA’S INADEQUATE SOFTWARE PROCESS

- Toyota failed to exercise a safe standard of care for software
- Relied too much on vendors
- Lacked internal expertise
- Inadequate supervision and training of software

There is a process in place for hardware, but not software:
- Process IN/OUT, timeframe not decided.
- There are processes at Toyota yielding no knowledge.
  ⇒ Large deviation in product quality among vendors.

Barr Chapter Regarding Toyota’s Code Complexity
From: Hideo Inoue.
To: [ ] Masashi Takagi.
Cc: [ ] Shigeyuki Kawana; hosotani@n1hs.tec.toyota.co.jp.
Bcc: [ ].
Subject: Re: A quick report on Vice President Takimoto’s failsafe progress report.

This is Inoue.
Kawana-san

What Takagi-san has written is right. Takimoto-san was Positive [...] to us.

This time it is just an explanation, but I would definitely like him to experience it on an actual vehicle. Inputting this kind of information to Takimoto-san is important and I think passing along information with relative frequency is a good thing. It looks like he would become a good supporter so I believe at some point it is a good idea to pass our wishes along to executives.

In truth, technology such as failsafe is not part of the Toyota Engineering division’s DNA, but isn’t it good that it is recognized as one of the major strengths of Toyota in the system controls industry? ....

However, thinking about the future, continuing on as is will not be a good thing. We will need to benchmark companies such as Bosch to gauge shortcomings and strengths.
<table>
<thead>
<tr>
<th>1) Conditions necessary for Failure to Occur, Failure Mode</th>
<th>2) Failure Conditions and Failure symptoms found in Real World? Note 1</th>
<th>3) Physical or Electronic Evidence, Failure Detection</th>
<th>4) Range of throttle opening</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software unilaterally opens throttle with Accelerator released. Idle Fuel Cut not active. Watchdog serviced, no EDAC error. Sub-CPU does not Detect Failure</td>
<td>No. Cannot engineer a test. No place found in software where a single memory/variable corruption results in a UA</td>
<td>Theoretical Fault Escapes Detection</td>
<td>Openings up to wide open throttle conceptualized although not found in real world</td>
</tr>
</tbody>
</table>

"Single memory corruption results in UA"

"Fault escapes detection"

- "No EDAC error" *(because there is no EDAC!)*
- "Idle fuel cut not active" *(because in same task)*
- "Watchdog serviced" *(because defective design)*
- Monitor-CPU "does not detect failure" *(because not designed to)*

"Openings up to wide open throttle"
UNREASONABLE SINGLE POINTS OF FAILURE

Safety critical systems shouldn’t have single points of failure

- This is the normal mode of design in automotive industry

Toyota tried to mitigate such risks, including in software

- But missed some dangerous single points of failure
  Failed to prevent or contain faults ...
- There are single points of failure in the ETCS
  Some demonstrated in 2005 and 2008 Camry L4 vehicles
  Unpredictable range of vehicle misbehaviors via task death
  Other memory corruptions can be expected

Barr St. John Report
## INDIVIDUAL TASK DEATH OUTCOMES
(Watchdog should have detected them all!)

<table>
<thead>
<tr>
<th>Task Death</th>
<th>Response (Fail-Safe)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ms task</td>
<td>ECM Reset (watchdog)</td>
</tr>
<tr>
<td>wheel speed</td>
<td>Not Detected</td>
</tr>
<tr>
<td>crank speed</td>
<td>Not Detected</td>
</tr>
<tr>
<td>engine speed</td>
<td>Not Detected</td>
</tr>
<tr>
<td>sigma task</td>
<td>stall (comm. Check)</td>
</tr>
<tr>
<td>motor control</td>
<td>if accel change stall (sys guards)</td>
</tr>
<tr>
<td>spark on cyl. 1</td>
<td>Not Detected</td>
</tr>
<tr>
<td>spark off cyl. 1</td>
<td>Not Detected</td>
</tr>
<tr>
<td>spark on cyl. 2</td>
<td>Not Detected</td>
</tr>
<tr>
<td>spark off cyl. 2</td>
<td>Not Detected</td>
</tr>
<tr>
<td>spark on cyl. 3</td>
<td>Not Detected</td>
</tr>
<tr>
<td>spark off cyl. 3</td>
<td>Not Detected</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Task Death</th>
<th>Response (Fail-Safe)</th>
</tr>
</thead>
<tbody>
<tr>
<td>spark on cyl. 4</td>
<td>Not Detected</td>
</tr>
<tr>
<td>spark off cyl. 4</td>
<td>Not Detected</td>
</tr>
<tr>
<td>fuel injection</td>
<td>stall (mechanical)</td>
</tr>
<tr>
<td>10° task</td>
<td>Not Detected</td>
</tr>
<tr>
<td>30° med</td>
<td>stall (mechanical)</td>
</tr>
<tr>
<td>Task X</td>
<td>if brake change cut-stall (echo)</td>
</tr>
<tr>
<td>duty solenoid</td>
<td>Not Detected</td>
</tr>
<tr>
<td>rcv a task</td>
<td>if accel change cut (echo)</td>
</tr>
<tr>
<td>rcv b task</td>
<td>if brake change cut (echo)</td>
</tr>
<tr>
<td>8 ms task</td>
<td>stall (immobilizer)</td>
</tr>
<tr>
<td>30° low</td>
<td>Not Detected</td>
</tr>
<tr>
<td>idle task</td>
<td>Not Detected</td>
</tr>
</tbody>
</table>

Sources: Arora and Loudon Vehicle Testing; source code analysis. Legend: “Not Detected” means in at least one vehicle test.
THE TEST SPACE IS EFFECTIVELY INFINITE

There are >16 million combinations of task death
- Memory corruption can kill 1, 2, or all 24

Each task can die in thousands of different states
- Vehicle operational states (e.g., cruise on/off; accel 5% vs. 50%)
  And what happens next; driver reactions to misbehaviors; etc.
- Internal software states

Test “samples” so far confirm
- Claimed fail-safes inadequate!

Barr Chapter Regarding Task Death and UA
UA FOREVER IF BRAKE ON AT TASK DEATH

Vehicle speed is ~ 45 mph

Gas pedal does not affect speed any more!

Brake on (even lightly) at start of task death

None of the “fail-safes” act > 30 seconds

Fail-safe acts only after driver removes foot (fully) from brake

Source: Loudon St. John Report
CASE-SPECIFIC OPINIONS

ETCS misbehavior is more likely than other causes
- Car should have stopped in less distance if throttle not open (*McCort*)
- Eyewitness testimony of alert driver using brakes (*Mrs. Schwarz*)
- No evidence of pedal entrapment by a floor mat (*photos*)
- No mechanical problems found at any vehicle inspection (*experts*)

Cannot identify with 100% certainty the specific software defects
- Toyota’s software design “deletes” evidence of software problems
  *Restart car and engine is fine* (*Toyota should have logged errors*)

More likely than not undetected Task X death
- Many brake pumpers don’t fully release the brake pedal (*Cooper*)
- “Car sped up when brakes were pumped” makes sense
OTHER SIMILAR INCIDENT CRITERIA

Vehicles with substantially similar ETCS software
- e.g., 2005-2009 Camry

Incidents with no apparent mechanical cause
- Lack of support for floor mats trapping accelerator pedal
- No indication of any mechanical issue before or after

Driver and witness statements describe UA
- And no evidence contradicting correct use of pedals

OSI Sources: NHTSA complaint database, Toyota FTRs, claims

Barr St. John Report
TOYOTA’S EXPERT’S EVOLVING STATEMENTS

ETCS contains “layers of protection” (Jul 2012)
- True, but misses the key point: there are gaps thru those layers

Brake echo is a “designed fail-safe” (Sep 2012-Aug 2013)
- No, IF it were “by design” the fail-safe
  would NOT require the driver to act before the fail-safe!
  would NEVER require removal of foot from brake pedal
  counter-intuitive (in an emergency!) and likely to increase (!) risk of harm
  would NOT stall the engine (given ECM reset is correct & safer)

“It depends on how much fuel” (Sep 2013)
TOYOTA’S EXPERT HAS NOT REBUTTED

My Operating System opinions/chapter
My Software Bugs opinions/chapter
My Memory Protections opinions/chapter
My Software Architecture opinions/chapter
My Watchdog Supervisor opinions/chapter
My Fail-Safe Modes opinions/chapter
My MISRA-C Violation opinions/chapter
My Coding Standard opinions/chapter
My Code Complexity opinions/chapter
My Stack Analysis opinions/chapter

Most of Dr. Koopman’s opinions/report