embedded
cracking the code to systems development

Why every embedded software developer should care about the Toyota verdict

David Cummings - December 03, 2017

If you develop embedded software for a living, and especially if you work for a large company with deep pockets, you could wake up one day to see the quality of your software being publicly disparaged based on highly questionable accusations. This happened to the Toyota engineers, who saw a flurry of articles published about their software with titles such as "Toyota Unintended Acceleration and the Big Bowl of "Spaghetti' Code" and "Toyota's killer firmware: Bad design and its consequences." This could happen to any one of us, as I will now explain.

Several months ago I wrote an article for Embedded.com describing my technical analysis of the 2013 Oklahoma jury verdict that Toyota's embedded software was to blame for unintended acceleration that resulted in a fatal accident. The full analysis appeared in a peer-reviewed IEEE article I had previously published, which I summarized for Embedded.com. The analysis in the IEEE article reveals that the accident theory presented to the jury by the plaintiffs' software expert was technically flawed and thus not credible, and therefore the jury reached the wrong verdict.

I was delighted to receive a number of thoughtful comments from readers of the Embedded.com article, as well as from readers of a reposting of that article on EETimes.com. A common issue raised by a number of those readers was the low quality of Toyota's embedded software, as alleged by the plaintiffs at the trial and discussed in a number of articles and presentations after the trial. The feeling expressed by several of these readers was that even if the plaintiffs did not adequately establish that the software caused the accident, surely the low quality of Toyota's software can't be ignored. Some readers expressed the opinion that, since the software was so bad, it must have caused the accident, even if the specific accident theory offered by the plaintiffs falls apart under technical scrutiny (which it does).

Due to space limitations for the IEEE article, I only had room to address the plaintiffs' flawed causation theory. I was not able to address the plaintiffs' criticisms of Toyota's software quality, which are also highly questionable. I subsequently published another article about the trial in IEEE Consumer Electronics Magazine in which I address one of my many concerns about the plaintiffs' criticisms of Toyota's software quality, namely, their criticisms of Toyota's use of global variables. Toward the end of that article, I provide a link to a more complete analysis in which I detail a number of additional reasons why the plaintiffs' quality criticisms are highly subjective and thus highly questionable. I will summarize those reasons in this column, covering the following three areas:

- Toyota's use of global variables
- The number of MISRA C violations found in Toyota's code
- Plaintiffs' contention that unintended acceleration events would be expected to occur in the fleet of Toyota vehicles once every week or two, based on research by Obermaisser

In this summary I will refer to the publicly-available trial testimony of the two software experts who testified on behalf of the plaintiffs. The first expert to testify did not see any of Toyota's source code, yet much of his testimony was directed toward criticizing the quality of that code. The second expert to testify, who did examine Toyota's source code, criticized the quality of the code and also claimed to have found a likely cause of the accident in the code (which is the subject of my original IEEE article). As you will see, the experts' questionable criticisms of the quality of Toyota's code can be used to disparage anyone's code irrespective of its quality, including the code of the experts themselves.

Toyota's Use of Global Variables

The first expert, who did not see any of Toyota's source code, told the jury that others who had examined the code had determined that there were approximately 10,000 global variables in Toyota's one million lines of code. He told the jury that this was an indicator that "Toyota's source code is of poor quality." He further told the jury that "global variables are evil" and the "academic standard is there should be zero" global variables. He then said to the jury: "Well, I know that the right answer academically is zero. 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."

However, it turns out that this same expert's own academic code violates the standards for the use of global variables that he presented to the jury. His academic code exhibits the same use of global variables that he told the jury was an indicator that "Toyota's source code is of poor quality."

On his university website, the expert advertises a software project that he led called Ballista. He provides a download link for the Ballista source code. I downloaded and examined the code.

That source code includes 6 C files, 19 *.h* files, and 21 C++ files, among others. These files comprise a total of 8,552 lines of code. In those 8,552 lines, there are a total of at least 68 global variables. That would scale up to 7,951 global variables in one million lines of code. This is close to the number of global variables (10,000 "“ 11,000) allegedly found in Toyota's one million lines of code, which was severely criticized by both experts. Thus, the first expert, who told the jury that "the right answer academically is zero" global variables, clearly did not apply that standard to his own academic code. He apparently has two sets of standards regarding global variables, one that he applies to other people's code, including Toyota's, and another that he applies to the code developed by his own academic research group.

In fact, there are comments in the Ballista code describing some of the reasons why the expert and his team decided to use global variables. These comments reflect just a few of the many possible considerations that could lead to a decision to make variables global. Such considerations were not acknowledged by either of the two plaintiffs' software experts during the trial testimony. Rather, the impression conveyed to the non-technical jury was that the numbers presented by these two experts ("five, ten" global variables in one million lines of code) are hard and fast, and if those numbers are exceeded then the software is of low quality, period. This is not an accurate reflection of real-world software development, as the first expert's own academic code shows, but nonetheless this is what the jury was told. Furthermore, the first expert also told the jury that "there is no reason" for Toyota to have this many global variables, even though he did not see any of Toyota's code, and even though his own academic code contains explicit comments describing reasons to use global variables.

For additional details, including access to the full trial testimony and the Ballista links, and to verify my analysis for yourself, please go here.

Toyota’s Misra C Violations

The second expert, who examined Toyota’s code, told the jury that he found more than 80,000 MISRA C violations in Toyota’s one million lines of code. Both experts told the jury that this was an indication that Toyota’s software was of poor quality.

On the second expert’s website, I found two bodies of C code that the expert has written and made publicly available for use by the embedded systems community, which would include developers of safety-critical embedded systems (the focus of MISRA C). One body of code computes CRCs, and the other performs memory testing. I downloaded both bodies of code and checked them for MISRA C violations.

These two bodies of code together, which comprise a total of 572 lines, exhibited 136 MISRA C violations. At a violation rate of 136 violations per 572 lines of code, these two bodies of code together would exhibit over 237,000 violations if they were scaled up to one million lines (compared to roughly 80,000 in Toyota’s one million lines of code).

Thus, the second expert’s own C code, which he advertises on the web for use by the embedded systems community, has a violation rate of 2.9 times the violation rate he claims to have found in Toyota’s code. Therefore, if the number of MISRA C violations in Toyota’s code (80,000 out of one million lines) is an indication of quality problems, as conveyed to the jury by both experts, then the number of MISRA C violations in the second expert’s own code (237,000 out of one million lines) is an indication of much more severe quality problems. The second expert apparently has two sets of standards when it comes to his use of MISRA C as an indicator of software quality, one that he applies to other people’s code, including Toyota’s, and another that he applies to his own code.

Both experts also testified that the number of MISRA C violations in a body of code is a predictor of the number of bugs in the code. They told the jury that for every 30 MISRA C violations, one can expect an average of three minor bugs and one major bug.

According to this metric, the second expert’s CRC and memory test code together should contain roughly 14 minor bugs and 5 major bugs. One would reasonably expect that the expert would not have made his code available to the public if he knew or suspected that it contained roughly 19 bugs. But he did make it available to the public even though it exhibits MISRA C violations that, according to his testimony to the jury, predict that it contains these 19 bugs. Thus, he apparently has two sets of standards when it comes to the use of MISRA C as a predictor of bugs, one that he applies to other people’s code, including Toyota’s, and another that he applies to his own code.

The first expert told the jury that based on the number of MISRA C violations found in Toyota’s code, that code would be expected to have roughly 2,717 major bugs (81,514/30). He also told the jury that, based on this metric (but without actually seeing Toyota’s source code), Toyota “has far, far too many bugs.”

Because the predictive relationship between MISRA C violations and number of bugs that the two experts presented to the jury is not limited to safety-critical code, I ran a MISRA C check on one of the first expert’s Ballista source files that was written in C, to see how many bugs it contains according to what he told the jury. I chose the largest of his C source files, which, together with 2 Ballista *.h* files that it includes, comprises 591 lines. This file exhibited 121 MISRA C violations in 591 lines of source code. According to the metric conveyed to the jury by both experts, this would mean that this C file has roughly 4 major bugs and 12 minor bugs in just 591 lines of C code. Scaled up to one million lines, it would have roughly 6,820 major bugs, which is roughly 2.5 times as many bugs as he predicted for Toyota’s one million lines of code (2,717 major bugs). One wonders how he would characterize his own 6,820 major bugs, given that he characterized Toyota’s supposed 2,717 major bugs as “far, far too many.”

The first expert’s Ballista code is academic code, created by computer science researchers who advise others on how to build high quality software. If, as the expert told the jury, MISRA C violations are a predictor of bugs, one would reasonably expect that he would ensure that his own academic code had zero MISRA C violations, to minimize the risk of it containing bugs. But he did not do that, resulting in this one file alone containing roughly 4 major bugs and 12 minor bugs, according to what he told the jury. Thus, he apparently has two sets of standards when it comes to the use of MISRA C as a predictor of bugs, one that he applies to other people’s code, including Toyota’s, and another that he applies to his own code.

For additional details, and to verify all of my numbers for yourself, please go here.

The Obermaisser Paper

The first expert told the jury that, based on an academic paper from a researcher named Obermaisser, the fleet of Toyotas at issue in the trial can be expected to experience a “dangerous fault” every week or two. He testified:

“For example, hardware faults are about every 10,000 to 100,000 hours per chip. … And out of those faults, maybe only two percent are dangerous. … Sometimes it is a software bug, sometimes it is one of these cosmic ray things, and you go on. But sometimes it corrupts something that is critical. And for safety critical systems of this type, Obermaisser was actually studying cars. He said about two percent tend to be dangerous. So this is going to happen, ballpark, one time per million hours. … But if you have a half million vehicles out on the road, and they are driving about an hour a day, that is a pretty typical number, then you will get maybe 31 dangerous faults a year across all 430,000 cars. That is an approximate number, but it is in the ballpark, or maybe 314. … So you're talking about a dangerous fault every week or two, and so you need to do something about it.”

The expert’s calculation that every week or two a dangerous fault would occur relies on Obermaisser supposedly having said in his paper that two percent of faults are “dangerous.” The expert used this two percent value in his calculation. However, Obermaisser never said that two percent of faults are dangerous. Obermaisser characterized two percent of failures as “arbitrary,” not “dangerous.” The word “dangerous” does not appear anywhere in Obermaisser’s paper, even though the expert told the jury: “Obermaisser was actually studying cars. He said about two percent tend to be dangerous.”

An arbitrary failure is not necessarily dangerous. An arbitrary bit flip due to a single event upset, for example, could change a critical variable to a numeric value that is not dangerous (e.g., reducing the magnitude of the throttle opening). Or, as another example, it could change the contents of a memory location that has no impact on the safe operation of the vehicle (e.g., changing one character of an ASCII string). Or, as a further example, it could change the contents of a memory location to an erroneous value that is then overwritten with the correct value before dangerous behavior occurs.

The jury was apparently not made aware that the expert misquoted the Obermaisser paper on which he based his calculation, changing the word “arbitrary” to “dangerous.” Thus, the jury was led to believe that there was scientific research behind his calculated result that every week or two a dangerous fault would occur in the fleet of Toyotas, when in fact his calculation was based on misquoted scientific research, and therefore his calculated result was not supported by the actual scientific research.

Moreover, the expert told the jury that the percentages of dangerous faults he used in his calculation (two percent) are “the standard percentages.” In fact, the expert had to change the wording of the Obermaisser paper in order to come up with his percentages of dangerous faults. Thus, those percentages are hardly “standard,” despite what he told the jury.

The expert went even further under cross examination, telling the jury that an unintended acceleration event would be expected in the fleet of Toyotas every week or two based on the Obermaisser paper, even though Obermaisser never discussed unintended acceleration (or acceleration of any sort) in his paper. For more details, including all of the relevant trial testimony and a link to the Obermaisser paper, please go here.

Conclusion

Many of us know from experience how easy it is to criticize someone else’s software. Software experts who testify at trial can capitalize on this, as they did in this case, and bombard a non-technical judge and jury with criticisms of the defendant’s software that are highly subjective and misleading (if not downright hypocritical). This strategy appears to have been very successful in this trial, where the financial stakes were quite high (reportedly more than a billion dollars).

As a result of this success, plaintiffs in future high-stakes trials involving software can be expected to employ the same strategy. And as we become increasingly reliant on embedded software in our daily lives, these sorts of attacks on software based on highly questionable quality assessments will become increasingly likely. (Self-driving cars come to mind as one obvious target.)

What can we do to reduce the likelihood that false accusations regarding software will lead to similar miscarriages of justice in future trials? We can speak out and make it clear that trial testimony by software experts must be held to a higher standard than was the case at this trial. In fact, as I write this, the Association for Computing Machinery (ACM) is in the process of updating their Code of Ethics and Professional Conduct, and they are soliciting input from the public. The most recent draft they released earlier this year is silent on issues regarding expert testimony by members of our profession, which I believe is a serious oversight. It is time to change this. If you agree, write to them (as I have) and make your voice heard. Otherwise, one of these days you, too, could find yourself squarely in the crosshairs of plaintiffs’ attorneys and their software experts. No matter how hard you and your fellow developers may have worked to ensure the quality and safety of your software, these experts will always be able to find ways to criticize that software and, in so doing, disparage you and your work.

DISCLAIMER: I have not seen any of Toyota’s source code, and therefore I have not formed an opinion one way or the other regarding the quality of that code. Rather, I am shining a spotlight on what I believe is an important (albeit little known) issue for our profession, using as an example the highly questionable attacks on the quality of Toyota’s software made by the plaintiffs in this trial.

David M. Cummings *is the Executive Vice President of the Kelly Technology Group in Santa Barbara, CA. He has over 35 years of experience in the design and implementation of software systems, many of which are embedded systems. Nine of those years were spent at the Jet Propulsion Laboratory, where he designed and implemented flight software for the Mars Pathfinder spacecraft. He holds a bachelor's degree from Harvard University, and a master's degree and a Ph.D. from UCLA. For more information, please see www.kellytechnologygroup.com.*