-

The Quality of Embedded Software, or the Mess Has Happened

29.10.2013 Andrey Zubinskiy

This article was originally published at the website "Kompyuternoe Obozrenie"; copied and translated by the editors' permission.

I'm warning you right away: don't read this text if your nerves aren't right. It's almost like Stephen King's stories. It's going to be scary - and quite a bit.

A cautionary and very sad story that lasted for more than six years has finally come to its logical conclusion. We all can learn something important from it, while I personally have got a chance to write kind of a continuation of my previous blog-post here, since there is really something to discuss.

The story began six years ago when two elderly ladies in Oklahoma were driving somewhere in their Toyota Camry, but their trip ended tragically: one of them (the passenger) was killed and the other seriously injured in a crash.

Their venerable age complicated the case, allowing the attorneys to drag the trial for many years and blame the car's strange behavior on the driver's error (it was just a bad luck for her, though; I can't imagine an elderly woman intentionally accelerating a Camry which is far not a light and swift car to a speed that Russian drivers, who own these chariots with soft sofas and for some reason treat them almost as "sports cars", find quite normal).

The strange thing about the car's behavior was that it unexpectedly started accelerating despite the driver's

attempts to stop it.

The first reaction of the attorneys for Toyota was obvious: "Sometimes people make mistakes while driving their cars".

But later an unpleasant thing happened. Many cases occurred when stubborn Camries showed a tendency to unintended and uncontrolled acceleration. Hundreds of new lawsuits followed. To blame drivers was now impossible, and Toyota blamed... the native floor mats which were of a wrong size and not flexible enough, so they trapped the gas pedal, leading to unintended acceleration.

Last year (2012) Toyota paid out more than one billion dollars in total to resolve hundreds of lawsuits, though a few cases in several states have been left unresolved - for example a jury in California rejected a 20-million claim by a relative of a woman who had been killed in her 2006 Camry in a crash caused by unexpected and uncontrolled acceleration.

NASA specialists were attracted to investigate the Camry's strange behavior, and it took them 10 months to study all the possible issues with the acceleration control subsystem. Their summary implied quite ambiguous conclusions (see this large pdf-file with the NASA team's report):

The NESC (The NASA Engineering and Safety Center) researching team found two possible failure scenarios (not related to mechanics...) in the ETSC-i throttle valve controller which could cause unexpected acceleration of the vehicle without generating diagnostic codes: the first scenario dealt with specific errors in the acceleration pedal position sensing system; the second with regular software errors in the acceleration controller processor which couldn't be detected by the car's diagnosing system. The second scenario implies unintended opening of the throttle valve while the fuel injection and ignition systems are still in operation. The team didn't find any direct evidence that those failures had really caused the accidents, but it still didn't mean that it had not been possible.

And finally the story ended: October 24-th a jury in Oklahoma found Toyota liable for the car crash that had happened six years earlier and imposed a 1.5-million penalty charge on the company.

After the trial was over, the community of embedded software programmers got a chance to publish the data related to the expertise of the firmware of that ill-fated throttle valve controller.

Those data were far from comforting.

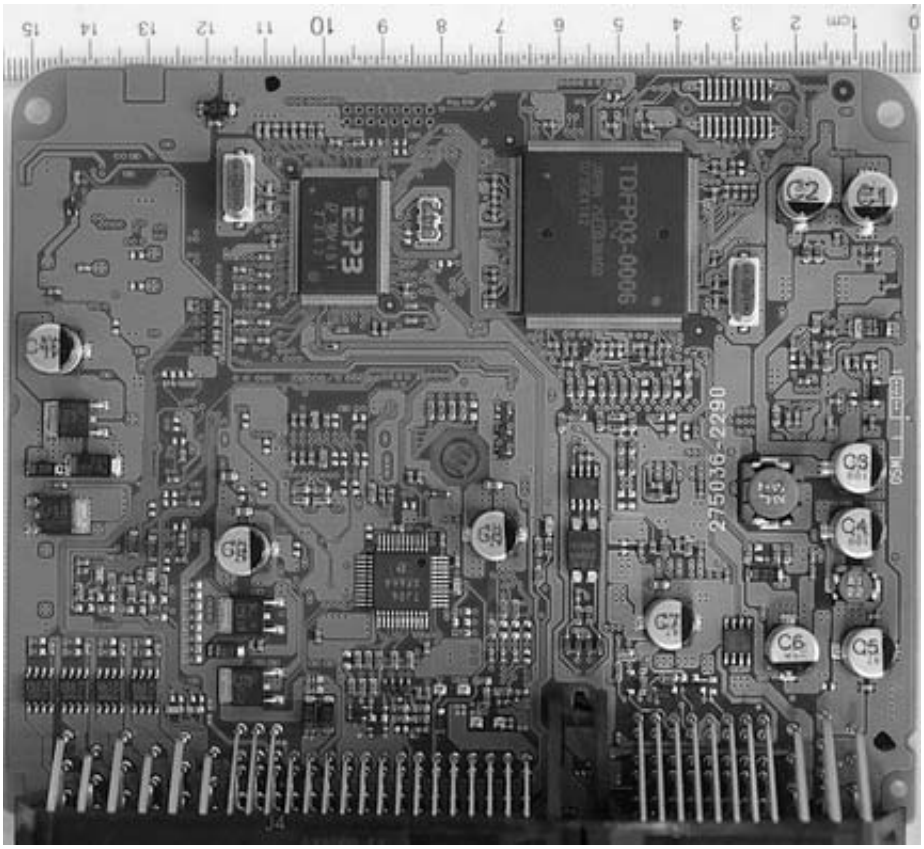
A team of experts (you can read about them at the website "EmbeddedGurus") checked the firmware of the throttle valve controller and found it (literally) "a shameful example of software design and development".

In their conclusions they pointed out a low overall quality of the code, a number of bugs in it which can cause an unintended acceleration of the car; the general architecture of the code execution control and protection systems are built on the "house of cards" principle, and, finally, the verdict which convinced the jury - software defects in the firmware were the reason for the car crash with severe consequences.

During the investigation of the controller, the experts checked and denied a number of suppositions made by Toyota claiming that the defects had been caused by hardware failures in the NEC (Renesas) V850 microcontroller - particularly in the interface of the external memory with a parity check. That it wasn't so is obvious without any investigation, for the Renesas (earlier known as NEC) controller is kind of a reference model for the car industry (and other industries too) and is used in a great variety of car models, so the world

would have known about such a serious defect (which evidently causes memory corruption) long ago and it would have got either fixed on the hardware level or at least mentioned by the manufacturer in Errata (a list of detected defects).

Below is a picture of the processor responsible for all that mess. It's quite a common small embedded computer, no rocket science at all - just a solid board with components common in the car industry (the largest chip is that very NEC-Renesas V850 microcontroller):



The throttle valve controller is not the most probable element where a dangerous defect might occur. It shouldn't be, at least. All it does is read the pedal position (or receive the values from other controllers via an onboard network like CAN or FlexRay, a more complex superstructure over CAN). If it reads the data itself, it generates a CAN-datagram for the other controllers and forms a control signal for the stepper motor of the throttle. And it is naturally integrated into the cruise control system (a system maintaining a steady speed of the vehicle). That's all. The above said is confirmed by a huge document on this subject released by Toyota themselves last year (a large pdf-file, for the fans of hardcore details only; it's interesting because of the explanations the company used to stick to last year).

Now hold on tight: according to the experts' summary, the firmware responsible for handling this task and implemented as a superstructure over a real-time operating system contained... eleven thousand global variables. The code of the firmware implementation is called the name well known to every programmer - "spaghetti". Analysis of the program's cyclomatic complexity reported 67 untestable functions, whereas the key function which read the angle of the throttle valve showed an absolutely incredible value, which makes it impossible not only to test but even maintain this program in any way. The degree of the software's conformance to the industry coding standard (there is an entire family of standards in the car industry called MISRA) is determined by the number of violations of the standard: in our story there were 80 thousand of these (by the way, Toyota has its

own internal standard which adopts only 11 rules from MISRA, whereas the minimum number required at the time the code was written is 93). Besides, the investigation revealed that this complex system didn't have any failure and error monitoring mechanism. In addition to such a nice state of things, the controller's firmware had all the functions responsible for security maintenance implemented by a single process. One more thing to discuss is the watchdog. It's a rare component in desktop systems, but a vital feature in the world of embedded systems. A watchdog (a watchdog timer) is a device external to the processor (even though it may be implemented on the same chip) whose working principle is very simple: if some process hasn't reset the timer previously set to a certain time, then the timer will cause a hardware interrupt to report a failure to the processor or initiate an immediate system reset. The experts found that the watchdog timer in that firmware was almost of no use because the only thing it controlled was one process handling rare interruptions of the watchdog timer, which means that a failure occurring in the handler of all the other interruptions could allow any process to run... for about one second and a half before the processor was reset by the watchdog. However, the experts were not even sure if a reset would occur after that interval - it might have never occurred at all. Another nice detail: the return codes of the most RTOS calls, intended to generate error messages, were completely ignored in the firmware.

Now let's discuss various architecture-related troubles which are also pretty nice. The main processor (that very NEC-Renesas V850 that Toyota unfairly blamed for everything in the beginning) is monitored by an additional microcontroller with a firmware by a third-party manufacturer which is beyond Toyota's responsibility. It's wonderful of course when you have an independent monitoring system, but how did it happen that the only analog-to-digital converter whose task was to read the analog signal of the acceleration pedal position appeared to be both non-redundant and integrated into that additional microcontroller? I can't even imagine how that could occur to anyone. Converters of this class don't have to be very precise (how precisely can the acceleration pedal be pressed, after all?), they are pretty cheap and well-established; but the manufacturer still decided to economize, thus creating an extremely dangerous single point of failure. Such a smart solution was adequately supported at the code level: the fail-safe code of the monitoring co-processor turned out to be dependent on a function executed by the main microcontroller, the name of the function concealed for the sake of commercial confidentiality. By the way, the engineers made this function handle a huge variety of tasks - from the pedal angle to throttle valve angle conversion to the vehicle control in the cruise control mode and even the diagnostics.

I'll be straight with you - when I was reading the original papers and read the paragraph about eleven thousand global variables used in the firmware, I felt somewhat thrilled. Any single wrong state shared by all the processes in a real-time system is in itself a big trouble indeed. Keeping this in mind, I'd like to remind you what I wrote about last time.

No doubt, those who wrote that strange code for Toyota and designed such an exotic processor architecture were not students or novice employees. Of course - and thanks God - those troubles occurred just in few car models, and all the failures were found, and Toyota must seriously improve firmware development and testing from now on in the wake of all those nightmares (there is special staff in companies whose job is reputational management - well, I wouldn't envy those guys who do this job for Toyota).

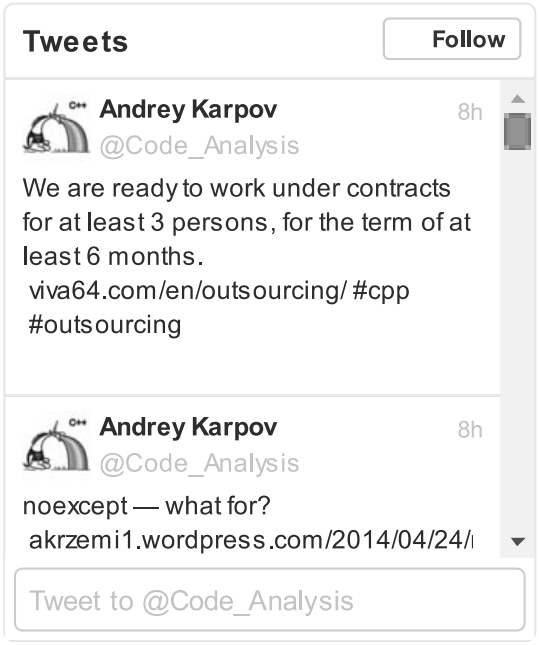
But we should not forget this particular example in the light of the IoT (Internet of Things) which is rapidly expanding in all directions. I hope manufacturers won't ignore it. After all, they won't be able to - because the scandal made a stir throughout the entire world.

Yours sincerely.

For Developers:

- All Resources
- Our Articles
- 64-bit Lessons
- Knowledge Base
- Terminology
- Reviews
- Blog

Twitter:



Blog:

- 24.04.2014

  A Check of the Open-Source Project WinSCP Developed in Embarcadero C++ Builder

  We regularly check open-source C/C++ projects, but what we check are mostly projects developed in the Visual Studio IDE. For ...

  Read more
- 18.04.2014

  Checking the Qt 5 Framework

  Static code analysis tools can help developers eliminate numbers of bugs as early as at the coding stage. With their ...

  Read more

- 16.04.2014

A Boring Article About a Check of the OpenSSL Project

Some time ago, a vulnerability was revealed in OpenSSL, and I guess there's no programmer who hasn't been talking about ...

Read more

Our Articles:

- 15.03.2014

Updatable List of Open-Source Projects Checked with PVS-Studio

Below is a list of open-source projects we have checked by now with PVS-Studio.

Read more

- 12.03.2014

How we compared code analyzers: CppCat, Cppcheck, PVS-Studio and Visual Studio

We have carried out a thorough comparison of four analyzers for C/C++ code: CppCat, Cppcheck, PVS-Studio and Visual Studio's built-in ...

Read more

- 25.02.2014

Readers' FAQ on Articles about PVS-Studio and CppCat, 2014

In the comments to our articles, readers would often ask the same questions. We decided to make a FAQ to ...

Read more

News:

- 03.10.2013

The standard PVS-Studio license now available for a team of 9 developers instead of 5 at the same price

We have changed the conditions of using the standard PVS-Studio license: now customers can get more benefit at the same ...

Read more

- 16.09.2013

We Hit a Mark of 1000 Error Samples Collected from Open Source Apps!

Our bug database where we collect samples of software errors detected in open source projects by our PVS-Studio static analyzer ...

Read more

- 09.07.2013

We received a status of Technology Partner of Embarcadero Technologies!

OOO "Program Verification Systems", Russia-based company that specializes on static code analysis and developer of PVS-Studio, C++ code analyzer, announced ...

Read more

Our Customers



support@viva64.com Contact Us