

Mediocre Embedded Systems? by Dick Selwood

Mediocre Embedded Systems?

by Dick Selwood, Embedded Technology Journal

“Over the last twenty five years, the general public have become self-conditioned to accept mediocrity in software-based electronic systems.” Christopher Smith, VP of Marketing at Green Hills Software, feels strongly that, not only in consumer electronics, but in other embedded and enterprise areas, there is complacency, not just among the users, but also among the developers. He fears that only something on the level of a catastrophic failure of public infrastructure will shake the industry and its customers out of that complacency. He spoke to European editor, Dick Selwood.

ETJ. When you say that the public are conditioned to accept mediocrity, what do you mean?

CS. Let me turn the question round. Have you bought some electronic equipment, found that it had software bugs and didn't live up to your expectations, and thought, “Well, it wasn't expensive, what do you expect?”

ETJ. Well, there was a set top box that had to be rebooted every week or so...

CS. Exactly. The public has become slowly conditioned, often through experience with their own purchases, to accept (or even expect) that if something is inexpensive, then the low price probably equates to cheap bill of materials, a cheap build, and flawed software. But what is worse is that the system developers and the programmers have also become used to thinking that software shipped in production equipment will always have bugs. And systems developers are not always aware of the real costs and the damage to their corporation of shipping mediocre products.

Look at cell phones: the simple phones that did little except make calls were fairly reliable. Today's multifunction phones are more likely to have software bugs, brought about by software complexity, and with many PDAs and smartphones, you get used to regularly rebooting.

In other areas, poor software quality or weak software development processes can lead to insecure systems: in fact, such a system cannot be secure.

ETJ. But isn't formal proof expensive and time consuming?

CS. It is, but not everyone needs a formal certification - there is a spectrum between no software process and formal proof. At the formal end is the software written for defence or high reliability applications, meeting, for example, FAA DO-178B Level A, IEC 61508, or its new adaptation for the automotive industry, ISO 26262.

But I am not talking about that high end of the quality spectrum. There are straightforward things that can be done to improve software quality that probably won't cost a lot and will even save money during the development process and again over the life of the product.

Let's look at the cost of bugs. At the extreme end, a bug can trigger a product recall, and only slightly lower is the cost of field upgrades to fix a significant bug that only appeared in operation. Apart from the cost of replacing the recalled products or carrying out the field fix, the cost to the company's reputation can be enormous. We can all think of products that have gained a poor reputation, and once gained it is hard to shake off.

ETJ. This is about after the development cycle, and that's what we are really interested in.

CS. The cost of fixing a bug during development increases the later in the development cycle that it is found - fixing a code bug before compilation is inexpensive; finding a code bug during the conventional "debugging" phase will cost a lot more - some studies show it can be up to 10X the cost of pre-compilation cost. This can grow even faster, particularly if the bug is elusive and intermittent: I know a team that spent six months tracking down a few elusive, yet critical problems. That is a lot of man months to pay for, and the significant delay in volume shipments of a product may represent an equally significant loss in revenue.

The first stage in solving the bug problem is to examine the process of software development and look at code writing, code checking and code testing. There is no secret sauce to writing good software, just good process, good tools and good standards to measure success. For example, in avionics, there are independent standards and independent bodies who evaluate the software and tools against those standards.

ETJ. I wondered when we would get around to tools and operating systems. After all, that is what Green Hills Software sells.

CS. But the tools and operating system are only part of the solution. The first thing on my list was good process, before considering any tools. But while we are on tools, just compare the tools that a software developer uses with those of the hardware designer. I recently heard of a site where their most complex EDA software represented close to a million dollars investment plus the annual maintenance charges for a single seat. More usually, investment is in many hundreds of thousand dollars. How many software developers have tools that cost anywhere near that amount of money -- even when the software content in products has gone from ten per cent ten years ago to almost ninety per cent today?

One of these days we will figure out how to get the software engineers properly resourced. They are a valuable resource and they deserve far better tooling than they currently receive.

ETJ. We?

CS. The embedded industry.

But to return to process. It can be something as simple as insisting on peer review of coding. This costs some time, but it has been proved over and over again to improve code quality. An audit/assessment of internal software processes can often work wonders in terms of where and how code quality can be improved. For example, it is worth setting up a test suite to exercise the code more and buying a static code checker. Software process is a big topic; we can't go into detail in the scope of this discussion.

ETJ. How about MISRA C rules checking?

CS. That can be handled by a good compiler or as part of the static code checker and is very useful.

ETJ. But isn't adding all these extra steps, code review, code checkers and test suites, going to lengthen the development cycle?

CS. It might, but not necessarily; it depends on where the problems lie. We have already discussed that the earlier we find a bug, the quicker and cheaper it is to fix it, and the extra steps should make it possible to fix bugs before

they become expensive. And you shouldn't measure development as time to initial production, the time to producing the first one or two products. Instead it should be measured as time to volume production of the final version of the product after feedback from those first sample units.

Wouldn't it be great if we could get the product into volume production a month, or two, or three early? Imagine the financial gain and the impact of an earlier delivery to market. This is definitely possible with the right engineers, the right tools, and the right process.

ETJ. What else should we be looking for to improve the debugging phase of the development process?

CS. The processor selected for the development should, wherever possible, implement an on-chip trace capability, especially since in-circuit emulation of high-speed 32-bit processors is no longer feasible. For example, there is the Nexus 5001 Forum Standard adopted by many processor suppliers and tools providers, and ARM has the Embedded Trace Macrocell (ETM) technology. With these trace standards, the trace non-intrusively monitors the program and often data status too. By connecting the trace to large storage, even gigabytes, it is possible to record seconds or even minutes of real-time data non-intrusively.

Intermittent bugs are expensive to find. Trace and the right tools gives a better handle on the problem. It is now possible to use a new generation of debugger on the trace data to step or run backwards in time and to set backwards breakpoints to help isolate and fix that intermittent bug. These debuggers are extremely powerful, and you can use them with trace always enabled to solve all manner of coding problems throughout the development process. Remember that an intermittent bug has to occur only once with this mechanism for it to be isolated and fixed.

ETJ. Clearly on-chip trace is an extremely valuable capability for the developer -- what about other hardware features that could help further?

CS. There are further steps that can be taken to reduce the impact of a bug, and these are based on the choice of hardware and real-time operating system. Use a 32-bit processor with a memory management unit (MMU) in combination with an operating system that protects in both the space (via MMU) and time domains (by ensuring any given process cannot exceed its allocated time period). This then fundamentally provides a robust platform to build your partitioned design. With this approach, a bug in the TCP/IP partition will not lock up the entire system, but will alert the operating system into taking

appropriate action.

The choice of processor and OS can have an enormous influence on the way that bugs are identified and the effect they have on the system.

ETJ. You earlier said that a poorly engineered system with an external interface cannot be secure.

CS. Yes. At one level, any system with poor software has a problem. But more important is the growing trend of adding Internet connectivity to many products, almost certainly weakening security. As soon as you add a TCP/IP stack, then you have a door into the system. So you should look carefully at what happens when someone accidentally or deliberately and maliciously connects to the system: developers don't always realise the implications, especially if this is a new technical area for them. Again, at the very least, you need an architecture that isolates the communications stack from the rest of the system. But you should be thinking in terms of how to deal with an unwanted connection from the beginning of the design and undertaking a security risk assessment for your system.

ETJ. Back to where we started, what is likely to change this culture where we accept mediocrity?

CS. The worst case would be a catastrophic failure of a public system. This could be through a bug, or it could be through malicious action or natural disaster. We have already seen a software worm infect the network at a nuclear power plant and a hacker use wi-fi to dump sewage from a treatment plant. Imagine a scenario whereby a bug, virus or hacker set all the traffic lights within a city to the same colour. This would cause severe disruption and might bring pressure for change.

Perhaps at a less catastrophic level, pressure will come from product liability cases, where, instead of a product recall, a bug results in someone suing, whether for real injury or for being sold a product that is not fit for purpose. The automotive industry has been conscious of this problem and addressing it since electronics began to take a significant role in the drive train and in safety systems. A glitch in the operation of the satellite navigation might not be considered a major problem; a software lock-up in the engine management system is.

Finally, there may be some pressures from the need to conform to Sarbanes-Oxley: intended to protect the public against corporate malfeasance, the US

Sarbanes-Oxley Act has already made developers and users of enterprise level IT systems look more closely at the accuracy of their systems and the traceability of versions and information. Where the data that feeds these systems is collected by embedded systems, these too will have to demonstrate accuracy and traceability.

ETJ. In summary, is it possible to raise the bar on software quality and reliability so that we can move away from public acceptance of mediocre performance?

CS. I believe so. You don't necessarily have to certify to avionics DO 178B level A or its equivalents in other industries. With investment in good engineers, good processes and good tools, it is possible to produce much higher quality software-based products at least as fast and inexpensively as when producing some of the mediocre software that we, the public, are experiencing in some of our purchased products.

Dick Selwood, Embedded Technology Journal

August 28, 2007