How to leverage automotive software development standards to mitigate risk

By Arthur Hicken, Parasoft

This article discusses some of the issues contributing to automotive software complexity, as well as the risks associated with automotive software development. We'll also discuss how implementing known development best practices, such as ISO 26262, helps organizations mitigate those risks.

When average non-engineer consumers think of electronic systems in automobiles, they likely think of integrated GPS, infotainment systems, and probably some vague notion that there is a computer somewhere in the car controlling some of the safety features. Of course, the reality is that modern cars are significantly more complex with software playing an increasingly larger role in all facets of functionality, including many safety-critical functions. In fact, cars have been leveraging electronic systems for critical functionality for decades, and market changes, such as the push toward an Internet of Things, have nudged automakers towards embedding a greater number of complex computer systems that run the gamut of criticality.

The business structures and supply chains associated with system development further adds to the complexity. It's rare, if it happens at all, that a manufacturer engineers and builds every component and subsystem in their cars from the ground up, leading to potential integration issues. A transmission is taken from this model, a good braking system from that one. While they may have worked well in their previous environment, in a totally new complex system they may well have unintended and unexpected results. As a result, automotive software is often a complex hodgepodge of systems that may or may not have been sufficiently tested. Implementing components in an ad-hoc manner without proper testing, especially in safety-critical applications, can be extremely costly.

The upside, though, is that there are known practices for helping automakers mitigate the risk of failure by building software quality into their development processes. According to some estimates, a standard mid-range car can have well over a hundred electronic control units (ECU) processing millions of lines of code - and this number is increasing. It's not uncommon for a manufacturer to have several models of cars with over one hundred million lines of code. There is a perception that the more expensive the car, the more software is embedded - and that most of the software is dedicated to high-end infotainment components. While it's true that these systems become increasing complex as you move up the model line, even introductory lines of cars use software to control steering, brake systems, electrical power distribution, and so on. And even seemingly minor shifts in features, such as Bluetooth, climate control, cruise control, etc, lead to exponential growth of code.

We can assume that more code translates to more complexity - and therefore risk, but the impact may not necessarily be significant. A larger contributor to business risk associated with automotive software is the integration of code developed from a variety of sources across multiple tiers. Most components, including ECU-based components, are subcontracted to second-tier providers who subcontract to third-tier providers and so on. Each preceding tier has specific requirements associated with the component they're developing. Organizations often (but not always) have practices in place for analyzing incoming code to ensure that the components function as expected.

But this assumes that every component along the supply chain is a new development. In reality, downstream tiers are branching off code written for a specific make, model, and year. The mutation and reuse of code takes place throughout the supply chain, which leads to a testing problem. How does the manufacturer implement end-to-end testing in such a chaotic ecosystem of software development? When the ECU in the steering wheel was originally developed for one vehicle and the ECU in the dashboard was developed for another vehicle, and neither ECU was designed for the vehicle they are currently embedded in, what's the impact? How can you ensure that the complete system functions as expected? It is entirely possible for both systems to pass testing as functional but be unable to communicate properly in all situations. What is the risk

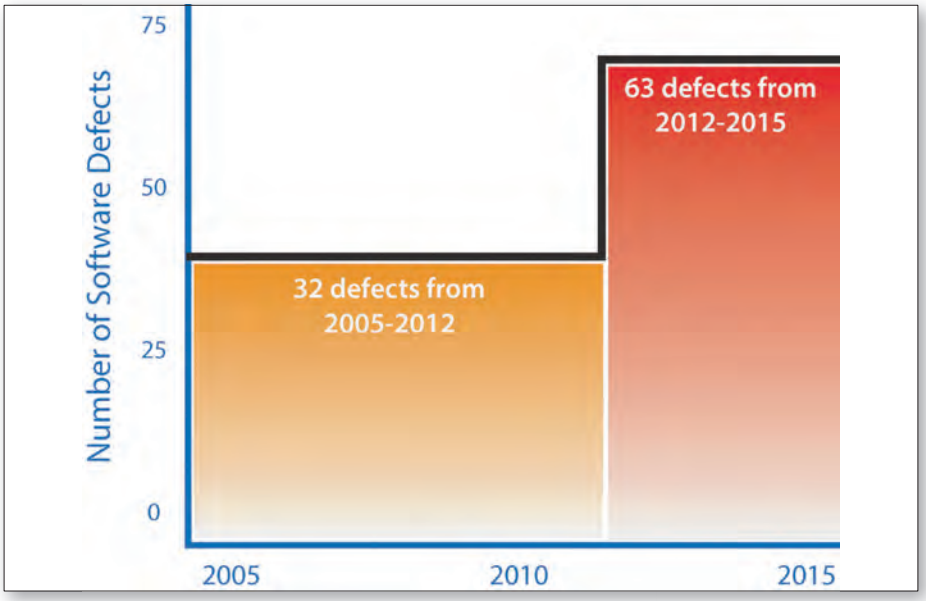

Figure 1. The amount of software defects has doubled in the last years, and NHTSA estimates that recalls and fixes cost automakers $3 billion per year.

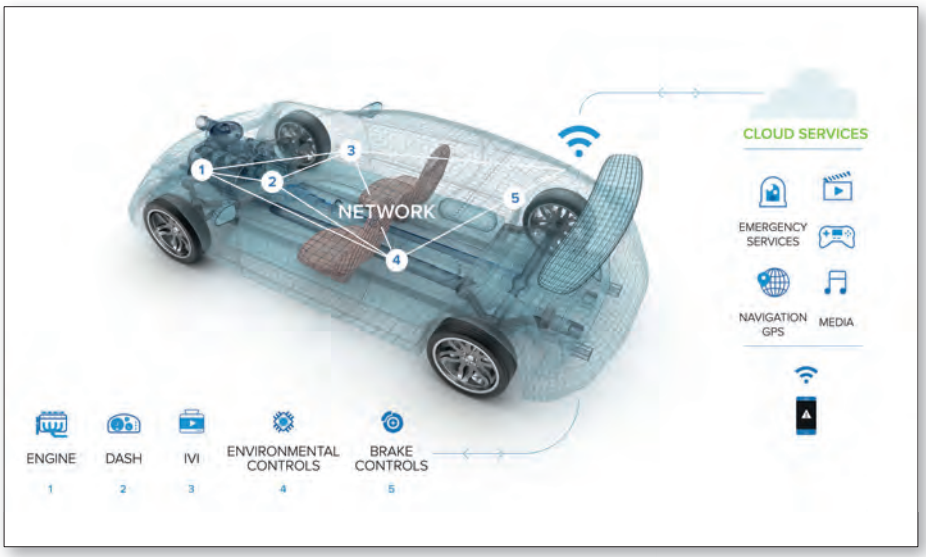

Figure 2. In modern cars, numerous complex computer systems are installed, with well over a hundred ECUs processing millions of lines of code.

associated with this gap? When organizations attempt to measure the cost of software development, they tend to look at general metrics: development time for the engineers; testing time for QA; building materials in the form of acquiring tool licenses, compilers, and other infrastructure components. These are important metrics, but often overlooked are the costs of failure. If the software in the braking system fails, what will it cost the business in terms of rework, recalls, audits, litigation, and loss of stock value? What if there is a loss of life?

We argue that the cost of quality is the cost of developing and testing the software, including all the normal metrics we identified plus the very tangible costs associated with a failure in the field. Defects cost automakers a lot of money. The NHTSA estimates that recalls and fixes across the industry cost automakers $3 billion annually. When it comes to the cost of software-related issues, a 2005 estimate from IEEE put the cost to manufacturers at $350 per car. When you consider the low profit margins across a line of vehicles, it's conceivable that a serious enough software defect can severely hurt the business.

The bottom line is important, but even more important is that people can become seriously injured or even die as a result of a software defect. And it doesn't matter how far down the supply chain the defect may originate, defects and all their associated consequences become the responsibility of the automaker. As such, any cost analysis around software development needs to take the potential costs of failure into consideration. We've argued that the complexity of the tiered supply chain for automotive software contributes to the overall risk associated with safety-critical systems. We've also reiterated the potential costs to automotive businesses. But there's another dimension to this issue that reside in the cultural difference between engineering and software development. Software development is almost never engineering. That is, certain concepts from engineering principles, such as repeatability, well-exercised best practices, and reliance on building standards have yet to become firmly established in software development. Additionally, training for software developers can be inconsistent - even non-existent - and organizations would have to go through great lengths to verify that their developers possess adequate knowledge to build safety-critical software.

This is in contrast to engineering in which the attitudes, mindsets, and history of the discipline enforce a process that is less prone to defects when compared to software development. That is not to say that engineers know what they're doing and software developers don't. Rather, it's to say that automotive engineering as a field is twice as mature as software development, and that the intangible, temporal nature of software perpetuates a cavalier attitude in which if it works, then it's done.

The emphasis in software development is around faster delivery and functional requirements - how quickly can we have this functionality? There is little incentive from management to implement sound engineering practices into the software development lifecycle. Achieving functional safety in software requires operationalizing certain engineering principles: functional safety must be proactive, processes must be controlled, measured, and repeatable, defects should be prevented through the implementation of standards, testing must be effective, deterministic, and should be done for complex memory problems.

The good news is that the attitudes around software development have been evolving. ISO 26262, MISRA, and other standards seek to normalize software development for automotive applications by providing a foundation for implementing engineering concepts in software development processes. Some organizations view compliance with ISO 26262 and other standards as an overhead-boosting burden without any direct value, but the truth is that the cost of failure associated with software defects is much, much greater than the cost of ensuring quality. As in electrical standards that specify a specific gauge of wire to carry a known voltage, coding standards can provide the guidelines that help avoid disaster. ■