Paying off technical debt in safety-critical automotive software

By John Paliotta

Vehicles have evolved from mechanical devices into complex integrated technology platforms with embedded software powering all major systems, including: engine control, powertrain, braking, driver assistance, and infotainment (Figure 1). Now, studies predict that by 2017, four out of five new cars will have an Internet connection. This “always-on” connectivity will result in new challenges as the line between consumer-grade software for infotainment and safety-critical software gets blurred.

For example, the telematics system provides features such as in-vehicle voice control applications, as well as interaction with the GPS system for navigation and traffic features. Soon a vehicle’s GPS system will be used for more than simply directions. As we move into the era of connected and autonomous cars, features such “auto SOS,” which summons help in the event of a crash, will be built on top of this existing telematics architecture.

Another example of this shift from consumer-grade to safety-critical comes to mind when reading recent news of an agreement between major automakers and the U.S. National Highway Traffic Safety Administration (NHTSA) to implement automatic emergency braking (AEB) as standard equipment on most vehicles by 2021. The AEB systems are controlled by software that powers cameras, radar, proximity sensors, and more that all need to operate flawlessly in order to safely stop a vehicle if a driver is slow to respond. This also means that an embedded camera that was previously used for passive driver assistance (parking for example) will now be part of a safety-critical system.

Insurmountable quality issues ahead

Most new software applications are built on legacy code bases. Since substantial monetary and time investments have gone into developing existing applications, there is naturally an interest in leveraging as much of what’s already been done as possible.

The problem with the reuse of existing code is that legacy applications often carry an enormous amount of technical debt. Technical debt is a metaphor for shortcuts taken during the initial design and development of a system. This “debt” is often caused by the continual development of software without the correct quality control processes in place, typically due to the tremendous business pressure to release new versions. The accumulated liability of the technical debt created eventually makes software difficult to maintain.

The key to reducing technical debt and improving quality is to refactor components (the process of restructuring application components without changing its external behavior/API), but developers are often hesitant to do so for fear of breaking existing functionality. One of the biggest impediments to refactoring is the lack of sufficient tests that formalize the existing correct behavior of an application.
Without sufficient tests, it is very difficult to refactor an application and not cause regressions in functionality or performance. According to a Gartner study, “a lack of repeatable test cases limits an organization’s ability to demonstrate functional equivalence in an objective, measurable way.”[2] This lack of sufficient tests ultimately means that a software application cannot be easily modified to support new features.

**Paying off technical debt**

Baseline testing, also known as characterization testing, is useful for legacy code bases that have inadequate testing (Figure 2). It is very unlikely that developers of an already deployed application will go back and implement all of the low-level tests that should be generated. They rightly feel that the deployed application is “working fine,” so why should they spend months of time retesting?

A better option in this case is to use automatic test case generation (ATG) to quickly provide a baseline set of tests that capture and characterize existing application behavior. While these tests do not prove correctness, they do formalize what the application does today, which is very powerful because it allows future changes to be validated to ensure that they do not break existing functionality.

Another benefit of having a complete set of baseline tests is that change-based testing (CBT) can be used to reduce total test cycle times. It is not uncommon for complete application testing to take one to two weeks. With change-based testing, small changes can be tested in minutes. Change-based testing computes the minimum set of test cases needed for each code change and runs only those tests.

As a result, developers are able to make incremental changes to the code and ensure that those changes are not breaking the existing behavior of the software. They are also able to do further analysis if something is broken to work out if a bug has been introduced, a capability has been removed that actually should be there, or if there is a bug that should be addressed because it may have other ramifications.

**Baseline testing to the bank**

In the Internet of Things-enabled world, a great amount of legacy code will find its way onto critical paths in new applications. Without proper software quality methods in place to ensure the integrity of this legacy code, the overall safety of the system could be compromised.

Baseline testing can help reduce technical debt in existing code bases, allow developers to refactor and enhance those code bases with confidence, and ultimately allow the owners of these legacy applications to extract more value.

John Paliotta is Co-Founder and Chief Technology Officer of Vector Software.

References: