

Addressing the Hidden Embedded Software Crisis in the Industry

Posted: 08 Jun 2007

**By Mark Underseth
CTO and Founder, S2 Technologies**

Over the years, embedded software development has evolved into a large-scale, globally distributed endeavor, posing significant engineering management challenges. Embedded projects now involve huge teams of developers, outsourcers, third party software technology vendors, chipset partners and even open source. However, software development methods and practices are largely the same as ten years ago, especially in terms of integration and testing. Hence, companies are struggling with the challenge of managing, integrating and verifying various components from different sources. As a short-term fix, software managers add more engineers and resources but with limited effectiveness and at very high cost. Often, they still end up delivering software releases late and with compromised quality.

The growing complexity of embedded software development requires a more reliable and scalable approach - one that adopts early developer testing practices and implements automated software verification to prevent and detect more defects sooner. The objective is to have all developers and integrators create reusable tests which can be shared and automated throughout the development cycle. This strategy replaces ad hoc testing with continuous automated testing to ensure the on-time delivery of fully tested and properly functioning products.

Embedded complexities

The market for sophisticated embedded software-based products, such as consumer electronics, is exploding. As the industry evolves to support a growing range of capabilities and features, the underlying microprocessors become smaller and faster, and the software content just keeps multiplying.

A few years ago, a typical embedded application included a few thousand lines of monolithic code developed by a few developers at a single location. Today's an embedded application may incorporate millions of lines of code developed by over a hundred developers. It has become a complex software platform comprising numerous software components brought together from various sources and locations.

Before, the embedded software industry was driven by technical competence. There has never really been a push to invest in processes and technologies to address this rapid increase in complexity. Now, in the race to beat the competition, product developers and manufacturers face greater time-to-market pressures and tighter product release schedules. They, however, have more code to manage, limited ability to properly test it, and less time to find and fix problems. This is a recipe for disaster.

It's apparent that the current approaches and technologies that were sufficient in the last decade are no longer adequate today. Software methods, practices and technology have not evolved to meet new complex integration issues.

Possibly, the most alarming observation is the relative time and resources devoted to QA or product testing. In many companies, the time spent on software coding or implementation is relatively short, while integration activities can take twice as long. However, product testing efforts are truly daunting, taking five to ten times as long as implementation, while staffed with very large teams that continue to grow.

In most companies, integration testing is merely a "smoke test" or "sanity test" to confirm a viable software build by manually executing a rudimentary set of tests. Even when integration testing is more extensive, the test coverage is limited by the time-consuming nature of manual testing. Often, the first time all embedded software components are extensively tested as an integrated whole is during QA or production testing.

Hence, QA engineers usually uncover large volumes of defects. A hiatus ensues as managers re-direct developers from other work to isolate, characterize and debug numerous critical and serious defects. Engineers try to salvage a release schedule. By catching defects late, developers are fixing bugs when they are the most difficult, time-consuming and expensive to resolve.

Development teams generally have no metrics or visibility into the health of their software until late, during integration or QA phases. Numerous defects, especially during production testing, put release schedules at risk. With so many critical and serious defects, software managers inevitably not only miss their delivery schedule, but also find it difficult to predict new delivery dates. Worse yet, they cannot be sure that the code they ultimately release is high quality and free from costly or dangerous errors.

New strategy

The embedded industry is overdue for a more scalable and effective software testing and integration strategy. Any effort to improve software integration and verification should address several key objectives:

1. Manual testing by developers or integrators must be minimized. Manual testing is too tedious, time-consuming, error-prone and is not a good use of valuable engineers. Moreover, for companies building multiple products concurrently from the same code base, engineers do not have access to all HW/SW permutations.
2. Achieve phase containment to prevent or catch defects early, before QA. Important metrics of a new process will include lowering defects escaping into QA and reducing the time and resources required in product testing.
3. Implement processes and infrastructure that are truly scalable and that can support integration and testing of components for multiple product lines concurrently.
4. Improve the predictability of releases. Provide earlier visibility into the software health of releases. Enable managers to pinpoint trouble spots and make decisions based on real data or metrics.

To achieve these objectives, the mantra needs to be "Automate, automate, automate." Then, reuse and automate the tests whenever code has changed or at various integration points throughout the development cycle. This strategy sounds conceptually simple but it does require a process change nonetheless, involving adoption of new methods, implementation of automated infrastructure and a change in mentality regarding the importance of developer testing.

Early test drive

Due to the complexity of today's embedded software, developers must actively participate in code verification. As the creators and architects of software components, only developers truly understand the inner workings of their code. Following a best practice from test-driven development (TDD), developers would ideally create tests upfront, before implementing code. Developing tests in advance has the added benefit of fleshing out your design; it especially aids in designing for testability. By thinking about testing upfront, designers take into account and implement facilities to access internal APIs, data structures or other information that aid in testing.

Thus, the first role of an embedded software verification platform is to facilitate developers in creating reusable, automated tests quickly and easily by providing specialized tools and techniques. For example, the verification platform might enable developers to quickly break dependencies by simulating missing code with a GUI, simple scripts or C/C++ code. Or perhaps the verification platform would support recording and playback to automate a series of manual test operations.

An embedded software verification platform also provides value at this stage by enabling developers to validate their tests before code is available. For API-level testing, a software verification platform like STRIDE can execute and verify tests by simulating or modeling APIs through simple scripts, C/C++ code or a GUI. This provides developers with very simple, quick means to execute tests and feed them canned responses to validate them. For example, if code-under-test depends on the return values of another application interface, the developer can dynamically mock-up the desired return values.

Common framework

This new approach succeeds only if developers and integrators create and deliver automated tests which are reusable by anyone, and the tests can be aggregated and automated in large-scale testing. To do this requires that common test framework and test guidelines are used by all software developers, integrators and testers.

The first key is the software verification platform, which serves as the common test framework that supports, manages, and automates tests from all of the developers and integrators. With the diversity of embedded software components, this means that the test framework should be flexible to support various testing strategies. Depending on the type of embedded software component, certain approaches may be more suitable. "Hard" real-time code may require tests written in native code to be directly built into the target while "soft" real-time applications can be exercised remotely from the host, possibly using a scripting language. Meanwhile, network protocols might have internal state machines that should be verified through white-box techniques; data-centric APIs may require facilities to efficiently enter complex data.

Secondly, the development team must conform to a level of uniformity when it creates tests. For example, guidelines might require that tests be written to be self-contained, not dependent on the preceding execution of other tests. Standard entry and exit criteria would guarantee that tests enter and leave the target in a consistent, known state, enabling tests to be executed in any sequence. All tests would leverage the same error handling and recovery mechanisms. Internal policies and conventions would establish naming conventions, archiving and maintenance policies and the standard languages for implementing tests.

Continuous automated testing

Once everyone has made the effort to deliver automated tests, the development team begins to derive major benefits from applying them. There are many opportunities to leverage the

automated tests, virtually for free. Consider the following:

- Developers can perform regression testing of their own code whenever they modify their software.
- Tests can be re-executed at "integration points," or whenever developers integrate their code with other software components.
- Developers' tests can be aggregated and automatically executed with an automated complete target build, at regularly scheduled intervals. This practice is referred to as "continuous integration" and is very effective for achieving and maintaining software stability because defects are uncovered earlier and frequently, providing continual visibility and metrics into the software's overall health.
- The collection of developers' and integrators' tests can also be executed by the QA team to complement their black box testing.

In these cases, the role of the software verification platform is to provide a framework for management, reporting and automation by aggregating, organizing, controlling and executing tests and then collecting, analyzing and displaying results.

Implementing the unified verification approach transforms the software development process and adds the following attributes: Tests from all developers will be managed and automated from a single common framework.

* Anyone can reuse and execute any test for any component at any time.

* A growing portfolio of developers' tests can be automated for regression or verification at various integration points or whenever code is modified.

* Metrics for software health and completeness can be collected much earlier in the development cycle.

Even if this strategy is applied incrementally or selectively to certain development teams, the impact can entail greater volume of defects caught and prevented before product test as well as shorter cycles and fewer resources required in product test. It can also enable an increase visibility and predictability into software health and delivery schedules and higher quality product, on-time.

This article was printed from EE Times - Asia located at:

http://www.eetasia.com/ART_8800467647_499495_NP_83e83a67.HTM