LDRA
Software Technology

Legacy Software Migration

Legacy code in the security-critical age

Article by Bill StClair Originally published in Military Embedded Systems – July/August 2008
Military
EMBEDDED SYSTEMS

Legacy code in the security-critical age

Bill StClair of LDRA

The cost and convenience advantages of legacy code reuse can be diminished or complicated when security and safety-critical risks are considered. If the legacy code is proven to be functionally correct and operationally viable, its acceptance is based on an assumption of what is expected to happen. However, it is the unexpected that typically causes faults, and structural testing provides a means of mitigating the unexpected.

Today, the "as built" acceptance of legacy software is coming under more scrutiny. This is due, in both military and commercial systems, to an increasing emphasis on security and safety evaluation criteria. With respect to software aspects of certification, there is a mandatory requirement for evidence of a repeatable verification process and the analysis that supports it. Structural testing is a mechanism to validate this evidence.

> *Legacy code reuse might seem like a good idea, but what about when security and safety-critical risks are a factor? Legacy code is expected to function correctly, but structural testing provides a way to mitigate the unexpected.*

Although once viewed as an unnecessary cost burden, a rigorous, standards-based development and verification process comes as a consequence of the emerging global perspective on the importance of safety in embedded systems industries worldwide. This perspective is defined by risks associated with activities as diverse as commercial air travel, medical equipment product deployments, global automotive product development standardization, and defense and security. In these applications, the liabilities, costs, and mission impacts associated with unexpected software and system behaviors are considered unacceptable.

As a member of the FAA's international working group on flight software, which is producing the next version of its DO-178 software standard, I have witnessed the growing awareness of the use of legacy software in flight software systems. The working group has strived to ensure that legacy code is properly managed, verified, and that it does not in fact become "dead" or inaccessible code where it could inadvertently be invoked for runtime execution without having been previously and properly tested. Historically, dead code has been seen as a cause of unexpected software behavior and poses a significant risk to flight safety.

With the emergence of object-oriented applications in embedded systems, using languages such as C++, Java, and Ada 2005, the working group has also realized that the possibilities for reuse of legacy code have grown exponentially. Legacy components can share member functions with new components, and the precise behaviors of these shared functions are not actually visible before runtime execution. In object-oriented systems, the unexpected has a higher probability of occurring.

The U.S. military also recognizes the risks associated with unexpected software behaviors, especially in the context of security vulnerabilities. The Air Force Research Laboratory, in cooperation with the National Security Agency, Department of Defense prime contractors, academia, and software suppliers, is managing a Multiple Independent Levels of Security/Safety (MILS) program to combine DO-178B with standards for security. This includes the Common Criteria and Director of Central Intelligence Directive 6/3, Protecting Sensitive Compartmented Information Within Information Systems. Though the MILS program does not directly address legacy code, many of its objectives are being applied to new projects and deployments that incorporate legacy software. The software development and verification guidance for the MILS program, which comes largely from DO-178B, now presents software suppliers and system integrators with the enormous challenge of implementing repeatable verification processes and mitigating the risks associated with unexpected software behaviors.

Given the challenges associated with security and safety-critical software, we need to identify best practices with respect to legacy code and propose a way to maintain and update legacy code. Such challenges are met through structural testing. Structural testing, sometimes described as "software testing software," provides a runtime environment in which test cases are auto-generated to exercise software behaviors based on a system-wide, path-level analysis of the code. Although in the past structural testing was criticized for not explicitly verifying functional correctness, this opinion fails to recognize that the goal of structural testing is to exercise the entirety of software structures, trap exceptions, and measure the resulting code coverage - not to explicitly test software functionality.

Unless the "as-built" architecture of legacy software is correctly analyzed, the impact of changes cannot be predicted nor can changes effectively be applied. Fortunately, the static analysis inherent to structural testing can also produce graphical representations of the architecture including call tree graphs, control flow graphs, data coupling tables, and set/used tables. These visualizations are especially helpful to engineers working with code from multiple origins such as modeling tools, hand code, and software libraries. Another byproduct of the static analysis dimension of structural testing is the automated application of coding rules to the source code, assuring implementation consistency between legacy and new code.

Unless the "as-built" architecture of legacy software is correctly analyzed, the impact of changes cannot be predicted nor can changes effectively be applied.

Advances in test technology have spawned a new generation of tools, not just another breed. These advances have arrived just in time to meet the needs of international software standardization, the globalization of embedded software markets, and the rising emphasis of security and safety-critical verification criteria. Now legacy software users can squeeze out unexpected software behaviors and help keep us safe and secure.