Code Assurance:

Dynamic Verification, Code Coverage, and Static Analysis, Oh My!

Audience: This is a basic introduction to a range of code verification techniques for developers who may not have deep knowledge of these topics. The type of testing we teach in this course is contextualized among other verification techniques. Regardless of your knowledge level, we present perspective, advice, and cautions that may be new to you.

Dynamic Verification

The style of testing supported by Unity's assertion model and CMock's mock generation is termed Dynamic Verification. Dynamic Verification is any sort of executable test code. In the Unity / CMock world, this verification occurs outside source code in dedicated test code. Test code, source code, and testing framework code are compiled and linked into small individual test executables (i.e. many test code files map to many test executables). That is, source code is treated as a library that test code makes calls against.

Dynamic Verification is great for exercising the intent of source code and verifying that it behaves as expected. Of course, it's essentially impossible to test every case, condition, and variation of your source under test. But in practice, smart testing techniques implemented by smart developers will eliminate most bugs and will aid design, documentation, maintenance, and refactoring.

Code Coverage

In a fashion, Code Coverage is related to Dynamic Verification. Specialized tools and libraries (or features of a programming language itself) allow source code to be instrumented "under the hood" to track the exercising of code. With Code Coverage you can easily determine which decision paths and functions or methods in your source code have been called when an entry point in the containing code is executed. When used in conjunction with Dynamic Verification that separates test and source code (as described in the preceding section), Code Coverage can give developers a good gauge of how much of their source code is exercised by their tests. Though it will not provide the same level of detail, Code Coverage can also be used in conjunction with system testing where test cases (usually executable scripts but also scripts of checklist-style operations to be performed manually) execute the final production build artifact for high level input / output checks. And, of course, Code Coverage need not be implemented with the style of testing described above at all.

A Word of Caution

Code Coverage numbers can cast a spell on you and trick you into thinking that you should strive for 100% coverage. In reality, it's worse to have needless and even outright bad tests solely for sake of coverage than to have less than 100% coverage. When used in parallel with unit testing, Code Coverage has two main purposes. First, it serves as an aid in finding and evaluating gaps in your testing. Second, it can help identify overly complex code. That is, if writing tests to advance a function past, say, 20% test coverage is challenging, with rare exception, that function is doing too much and is in need of refactoring.

Static Analysis

Static Analysis is a type of non-executable software verification where a tool (often separate from a compiler) parses source code looking for problematic coding in terms of omissions, logical problems, and known poor practices. Static Analysis is implemented by tools such as Lint and is able to identify problems that Dynamic Verification cannot and that most compilers do not check for. For instance, Static Analysis can find unreachable code, buffer overflow scenarios, gaps in comparison logic (i.e. incomplete coverage of if / then or switch / case flow control), assignments confused with comparisons ("=" confused with "=="), looping edge case omissions, and off-by-one indexing.

When to Use Each

So when should you use each? That's up to your good judgment. There's certainly nothing preventing you from using all three. These three techniques of code assurance are complementary. Used intelligently they can give you a great picture of the health and wellbeing of your code base. That said, there is also an overhead to using each that requires a commitment to these approaches. Each usually requires special tools and this then requires licenses and some sort of automation to support a reasonable workflow. Automation invariably means its own maintenance. Of course, each approach also leads to reports and reviews that lead to meetings and processes.

If you ask our opinion (thanks for asking, by the way), we'll tell you these things:

- Dynamic Verification gives you the most bang for your buck. Good testing in the tradition that inspired Unity and CMock seems to magically lead to good design, ease of maintenance and refactoring, and a form of documentation in the test statements themselves. Because you're actually exercising the source code in executing it through tests you naturally gain confidence in its behavior. Further, good executable testing practice tends to overlap at least some of the simple features of Static Analysis.

- We've found that the easiest to manage option for running Code Coverage is in conjunction with segregated test and source code (such as explained in the first section of this document). Test code isolates the source under test giving an atomic view of your coverage results (i.e. free of coverage overlap from multiple execution paths in a monolithic release executable). The automation necessary for dynamic verification tends to provide good support for adding Code Coverage to your build process. If you choose to use Code Coverage, please carefully consider our earlier word of caution in how to approach it.
- Do not misunderstand us. Static Analysis is very valuable in its own right. Two scenarios in particular are well served by Static Analysis. The first scenario is that of secure code. Many security vulnerabilities (e.g. buffer overflows or underflows, stack and pointer corruption, etc.) are found in code problems that Static Analysis is well suited for exposing. The second scenario is with regard to various industry or government quality checks and certifications. These tend to rely on concepts tied to Static Analysis.