**Project #2 - Local Test Harness**

**Purpose:**

Developing software consisting of multiple packages requires frequent testing. If the software has complex features, we want to build it incrementally. First design and implement a very basic core with a small number of packages, then add features one-at-a-time by adding new packages, or adding a few lines of code to an existing package. Each time we add new functionality, we build and test. That way, if additions break existing code, we know where to look, e.g., in the newly added few lines of code. A test harness allows us to use that incremental process efficiently.

A test harness should allow us to define and aggregate many small tests, each of which run with exception handling and results logging. Our goal is to do that without littering our code with many try-catch blocks and many logging statements. The technique we use to do that is to define, in the test harness, an execution function that accepts a callable test object - function pointer, functor, or lambda - and invokes the object within the scope of a try block. A catch clause displays exceptions and returns false to indicate test failure. Otherwise the executor returns the test predicate returned by the test object, e.g., true or false for pass or fail.

The test executor also provides results logging with predefined messages. Your test harness should define several levels of logging, e.g., just pass or fail, or more detailed, test-specific messages, or very detailed debugging messages, which display a time-date stamp.

The test harness provides a mechanism to link any number of tests into a test sequence. Typically, during development, we keep adding tests for each new feature, and can rerun tests just for that feature, or all the tests of all the features, to insure that the new addition didn't break existing functionality.

### Requirements:

Your Local TestHarness Project:

1. **Shall** be prepared using the latest version of Visual Studio, and written in the standard C++ programming language, using the standard C++ libraries.
2. **Shall** use [Visual Studio, Community Edition](https://www.microsoft.com/en-us/download/details.aspx?id=48146) available at no cost.
3. **Shall** provide packages for an Executive, TestHarness, Logger, and FileManager (if needed).
4. **Shall** accept a TestRequest XML string defining one or more test elements by naming dynamic link libraries.
5. Each test element **shall** contain a testDriver element and one or more testedCode elements.
6. TestDrivers **shall** implement an ITest interface1 that declares a test method taking no arguments and returning a boolean value indicating test pass or failure.
7. On receiving a TestRequest, the TestHarness **shall** parse the request, load the corresponding libraries, sequentially create instances of each TestDriver, and log pass-fail status of each test, along with a time-date stamp.
8. **Shall** provide one or more TestRequests that conduct testing of at least three of your project packages.
9. The Executive package **shall** provide processing to demonstrate that these requirements are met, by using facilities of the TestHarness.
10. C++ does not provide an interface construct. However, it is common practice to provide a struct with all pure virtual methods and implements a virtual destructor that has an empty body. These "C++ interfaces" have no constructors, no data members, and define no processing other than the virtual destructor. These serve the same role as interface constructs in the managed languages C# and Java.