Unit Testing for ESRI Developers

A test case for ArcUnit

By Brian Noyle and David Bouwman, DTSAgile

While some may hope that this column might define unit testing as a component of the software development process that we, as geo-developers, are allowed to ignore, our contention is that it is a critical component in any GIS software development effort. It is a facet of development that cannot be ignored.

An informal industry survey performed in 2008 by one of the coauthors of this column revealed that 48 percent of developers in the GIS industry do not write any unit tests. This statistic, when paired with the fact that maintenance costs for custom development projects typically exceed 50 percent of project life cycle costs, indicates that testing standards require a bit of attention in the GIS industry.

The basic principle underlying unit testing is simply to take the smallest executable segments of code—typically at the method level—and prove that the code works as expected under as many anticipated circumstances as possible. Most often this process is performed by writing code that can be repeatedly executed using an automated testing framework such as NUnit or MbUnit. While there are other methods, the primary reasons for this automated testing regimen are threefold:

- The developer must prove that custom code works.
- The developer must prove the overall design works (e.g., follows good OOP/OOD practices, separation of concerns, single purpose classes).
- The developer must have a concise and rapid means of tracking and managing regression as the code base evolves.

At the conceptual level, the unit testing process is very simple and is illustrated in Figure 1. In a typical scenario, a test class is written that instantiates a class to be tested. The test fixture class calls methods on the class under test and validates the results of those method calls. Most automated testing frameworks will prepare a report of test results so the developer can quickly identify and address problem areas in the code.

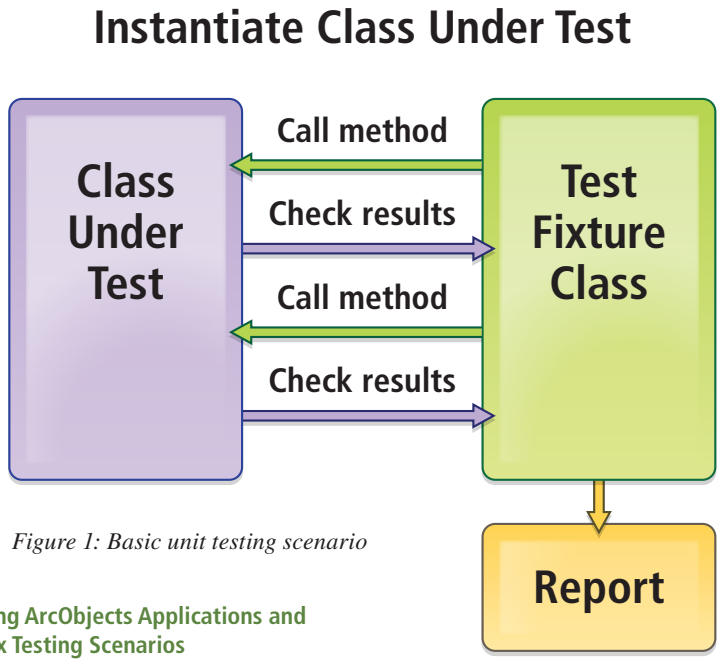

Figure 1: Basic unit testing scenario

Designing ArcObjects Applications and Complex Testing Scenarios

Simple methods with simple arguments mean simple unit test fixtures. Nearly everyone reading this article will understand that testing a summing function, a user authentication routine, or a method to fetch data is a relatively straightforward process. At the same time, testing complex methods that accept complex inputs results in complex test fixtures with results that are frequently difficult to reduce to a Boolean good or bad result.

While the authors work primarily in the Web realm on a daily basis, we have written literally hundreds of thousands of lines of ArcObjects code over the years and recognize the integral role such applications will continue to play in the GIS enterprise for the foreseeable future. Unit testing these applications is critical. Most ArcObjects code falls into the second, more complex, testing category. For example, how does the developer test that a custom edit sketch task returned a valid geometry or that the resulting geometry intersected with a target layer to return the correct number of features?

The first step in effective unit testing of ArcObjects applications is designing the application using patterns that facilitate unit testing from the start. In general this means following good object-oriented design patterns as shown in the diagram in Figure 2.

Methods should be single purpose and have no side effects or reliance on global variables (IApplication, for example). Wherever possible, application logic should be separated from eventing and the general "wiring" that makes the application go.

In the case of ArcGIS Desktop applications, the developer must be careful to separate ArcMap from ArcObjects. Keep custom code out of the ArcMap event handlers and encapsulate logic in business objects and utility classes that can be independently instantiated and tested. Conversely, keep event delegates and sinks out of your custom logic. Let's just all agree that ArcMap is going to raise that OnSketchFinished event and the .NET CLR is going to pass it off to your delegate. If they don't, what are you going to do about it?

Simply pass needed data into your custom classes from your unit test fixtures and validate what you have control over.

For stand-alone ArcGIS Engine and ArcGIS Server applications, the developer typically controls all the code so a custom app. will typically already contain code to create an instance of everything a unit test needs. While there is no IApplication lurking in the background to muddy the waters, validation of custom functions is still complex because we still need a way to validate a geometry or a feature class.

When developing applications for ArcGIS Server, custom code should be kept out of code-behind files and *.asmx files. This allows migration of custom components to the server object container (SOC) and will increase ease of unit testing.

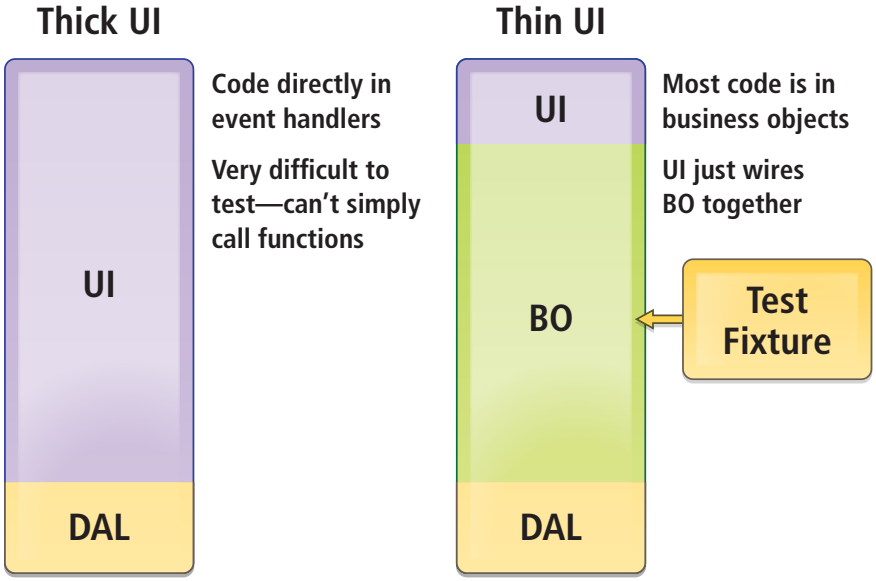

Figure 2: Proper abstraction of business logic away from UI and event handlers to facilitate testing

Unit Testing Geometry Operations: A Test Case for ArcUnit

No matter how much we design for unit testing, we are still faced with the problems of how to collect the needed objects/data to pass in to a test method (feature class, geometry) and how to validate the results of the test (e.g., check that a given geometry result is correct). ArcUnit, a community-based open source project, provides tools and utilities to address this need. Currently hosted on Assembla at http://svn2.assembla.com/svn/arcdeveloper/TestingUtilities, the ArcUnit effort consists of a series of utility classes and tools designed and implemented to assist the developer in manipulating and validating data when unit testing ArcObjects code. These tools can be freely downloaded by developers who are encouraged to use the tools provided and contribute new functionality for the benefit of the developer community.

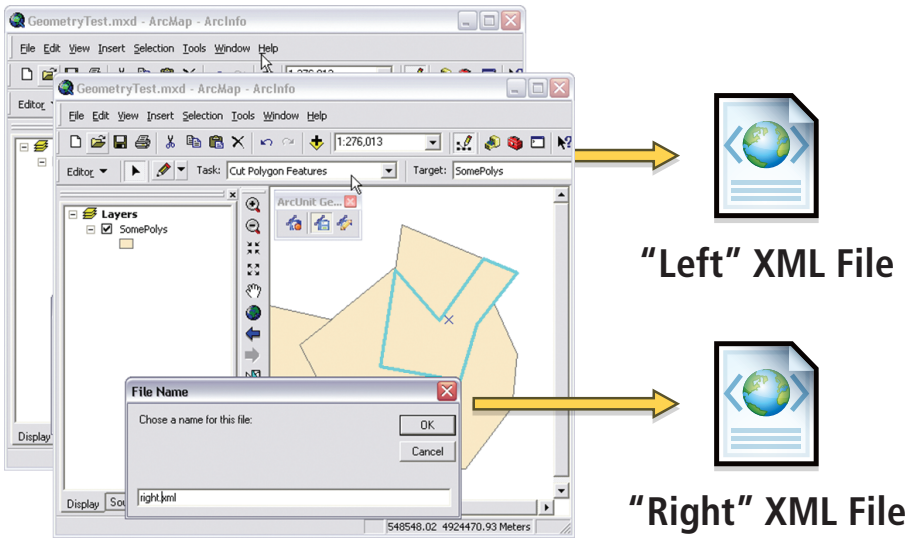

Figure 3: Using ArcUnit custom editing toolbar to serialize geometries for use in unit tests

The ArcUnit code base has been started with tools and utility classes to unit test custom geometry editing functions. To date, the effort has focused on how to simulate sketches, store and retrieve geometries, and create tests against independent datasets not tied to a specific instance of a geodatabase. A custom ArcGIS Editor extension and toolbar are included in the source code, as well as utilities for serializing and deserializing ArcObjects and classes to simulate commonly

Continued on page 40

Unit Testing for ESRI Developers

Continued from page 39

used ArcObjects interfaces such as IFeatureClass and IObjectClass.

Using ArcUnit

As a test case for illustrating how to use ArcUnit utilities, assume that a developer must validate a custom Split Polygon edit task in ArcMap. Using the design principles discussed above, the developer separates the implementation of the polygon split from the eventing in ArcMap. To test the custom logic of the split edit task, the developer now needs

- A source polygon to be split
- A polyline to be used to cut the source polygon
- The resulting "pieces" from the split operation to use for validation
- A series of invalid polylines to test negative split cases

How might the developer get the information required for effective unit testing without being tied to user interaction and a geodatabase instance? The answer lies in the IXMLSerialize interface. More than 200 ArcObjects classes implement this interface (including many aspects of the geodatabase), and any of these objects can be stored as an XML representation. A custom editing toolbar supplied in the ArcUnit source code allows a developer to serialize sketch geometries or selected geometries from a map and load/draw geometries from existing XML files as shown in Figure 3.

The basic workflow for unit testing geometry operations with ArcUnit is simple. Using the ArcUnit editing toolbar in ArcMap, needed geometries are created and serialized into XML files for use within unit test fixtures. The resulting XML files are then stored as embedded resources within the unit testing project inside a Visual Studio solution (illustrated in Figure 4), so that they are source controlled and have no dependency on user action within ArcMap or on a specific instance of a geodatabase.

To write unit tests to validate the geometry operations involved in the Split Polygon edit task test case, serialized geometries are loaded from resource files at test startup and passed-into a function under test from within an individual unit test, and output geometries are compared using IRelationalOperation::Equals(). The test source code example shown in Figure 5 leverages GeometryStorage and GeometryRelations utility classes included with ArcUnit to assist with serialization/deserialization and unit test assertions, respectively.

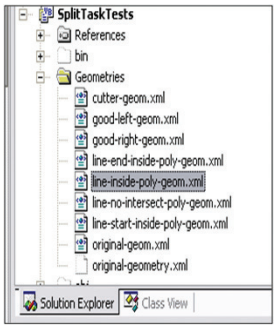

Figure 4: Captured geometries are serialized as XML and used as embedded resources in the test project.

Conclusion

The authors have generally found unit test coverage for custom GIS development initiatives within the ESRI realm to be comparatively low relative to other software sectors. In defense of the geodeveloper community, unit testing for ESRI applications presents a special case where typical spatial operations are complex and difficult to test without dependencies on the containing application, user interaction via the GUI, and geodatabase instances. We believe that unit testing is still a critical component of our custom development efforts and that the ArcUnit initiative can serve as a starting point for a whole host of mock objects, data serialization/deserialization routines, and test patterns that will assist the community in guaranteeing high-quality code against the ESRI COM-based APIs by providing a library of functions covering many common GIS testing scenarios.

For more information, contact

Brian Noyle, Senior Software Architect
DTSAgile
Fort Collins, Colorado
E-mail: bnoyle@dtsagile.com

David Bouwman, CTO and Lead Software Architect
DTSAgile
Fort Collins, Colorado
E-mail: dbouwman@dtsagile.com

```
1   [Test()]
2   public void SplitPolygonTest()
3   {
4
5       IGeometry cutter = GeometryStorage.RetreiveFromXml(GetXML("Geometries.cutter-geom.xml"));
6       IGeometry original = GeometryStorage.RetreiveFromXml(GetXML("Geometries.original-geom.xml"));
7       IGeometry goodleft = GeometryStorage.RetreiveFromXml(GetXML("Geometries.good-left-geom.xml"));
8       IGeometry goodright = GeometryStorage.RetreiveFromXml(GetXML("Geometries.good-right-geom.xml"));
9       //Call the function
10      SplitPolygon output = geometryOperations.SplitPolygon((IPolygon) original, (IPolyline) cutter);
11
12      //Compare the outputs
13      Assert.IsTrue(geometryRelations.AreGeometriesSame(output.LeftGeometry, goodleft), "Left Geometry was incorrect", null);
14      Assert.IsTrue(geometryRelations.AreGeometriesSame(output.RightGeometry, goodright), "Right Geometry was incorrect", null);
15
16  }
17
18  [Test(), ExpectedException(typeof(ArgumentException))]
19  public void SplitPolygon_LineInsidePoly()
20  {
21
22      IGeometry cutter = GeometryStorage.RetreiveFromXml(GetXML("Geometries.line-inside-poly-geom.xml"));
23      IGeometry original = GeometryStorage.RetreiveFromXml(GetXML("Geometries.original-geom.xml"));
24      //Call the function
25      SplitPolygon output = geometryOperations.SplitPolygon((IPolygon) original, (IPolyline) cutter);
26
27  }
```

Figure 5: Test source code for the Split Polygon edit task in the test case leverages GeometryStorage and GeometryRelations utility classes included with ArcUnit to assist with serialization/deserialization and unit test assertions, respectively.

About the Authors

Brian Noyle

Originally trained as a global change biologist and tundra botanist, Brian Noyle has nearly 10 years' experience as a GIS software developer and architect. His professional and technical interests are primarily focused on moving clients toward more standard architecture and development practices and patterns to facilitate a closer integration of GIS with the standard IT enterprise. Noyle has extensive experience in full software life cycle management with a focus on delivering through Agile project management methods. When he's not in the office, he can be found on his mountain bike, picking a bluegrass lick on a guitar, or standing in a river waving a stick at amused trout.

Dave Bouwman

Dave Bouwman has been designing and developing GIS software for the last 12 years with projects ranging from small Web sites to statewide enterprise forest management systems. Over the last few years, he has been leading a team of developers in the pursuit of great software built in a sane manner. The combination of an Agile process with pragmatic development practices taken from extreme programming has led to a highly optimized methodology of creating solid software that he and his staff are proud to put their names on. When not attached to a computer, Bouwman is often found mountain biking on the trails around Fort Collins, Colorado.