The NUnit 3.0 Extended Testing Platform

NUnit is well established as a unit-testing framework for .NET and Mono environments. Through the 2.4 series, we have restricted the scope of NUnit in order to keep it within the limits of its current Vision statement and the resources of a small project.

Not so with NUnit 3.0. It will wrap and extend the existing capabilities of NUnit to create a broad platform for team-based, test-centric development. **Diagram 1** on the following page shows the relationships among the components of the platform.

Plugin Architecture

NUnit 3.0 will be completely plugin-based, allowing a lot of flexibility in devising new ways to test. We'll encourage as many people as possible to develop add-ins, provide a central place to download them and bundle those that attract the most general interest.

NUnit 2.4 already supports addins for loading various kinds of tests, but the architecture is limited. NUnit 3.0 will completely replace the existing addin mechanism and make use of plugins for virtually all of its processing, just as apps like Eclipse and MonoDevelop do today. All new features will be developed first as a plugin or addin.

Layering

The basic architecture of the NUnit platform consists of three layers: the Test Runner/UI Layer, the Test Engine Layer and the Framework Layer (see **Diagram 1**).

Test Runner Layer

The Test Runner or UI layer contains various runners, some provided by the NUnit team, others by independent projects leveraging the NUnit platform. Some runners are stand-alone programs, while others are tasks or plugins running under the control of an IDE or other application. This diversity of runners is part of the reason we refer to NUnit 3.0 as a Testing Platform – we expect many different runners to come into existence and will facilitate their development by providing reusable controls for several key environments.

Programs in this layer are able to participate in the NUnit platform plugin architecture, providing extension points that allow them to be extended. Plugins at this level will usually add some functionality to the UI and may require the presence of specific test engine plugins in order to operate.

The NUnit project will continue to provide both a console runner and a WinForms-based GUI with extended capabilities beyond the NUnit 2.4 GUI. In addition, two new GUI runners will be developed, one based on WPF, the other on GTK#.

We'll work with the NAnt project to provide updates to the NAnt task for use with NUnit 3.0, with the goal of keeping that task current as new versions of NUnit are released. We will participate in the new Gallio project, which aims to provide a common UI for use with a variety of test frameworks to provide an NUnit plugin for their environment.

In the area of IDE integration, we will deliver a Visual Studio addin or package for running NUnit tests. Since other folks are already providing open source plugins for SharpDevelop and MonoDevelop, we'll work with them to ensure compatibility.

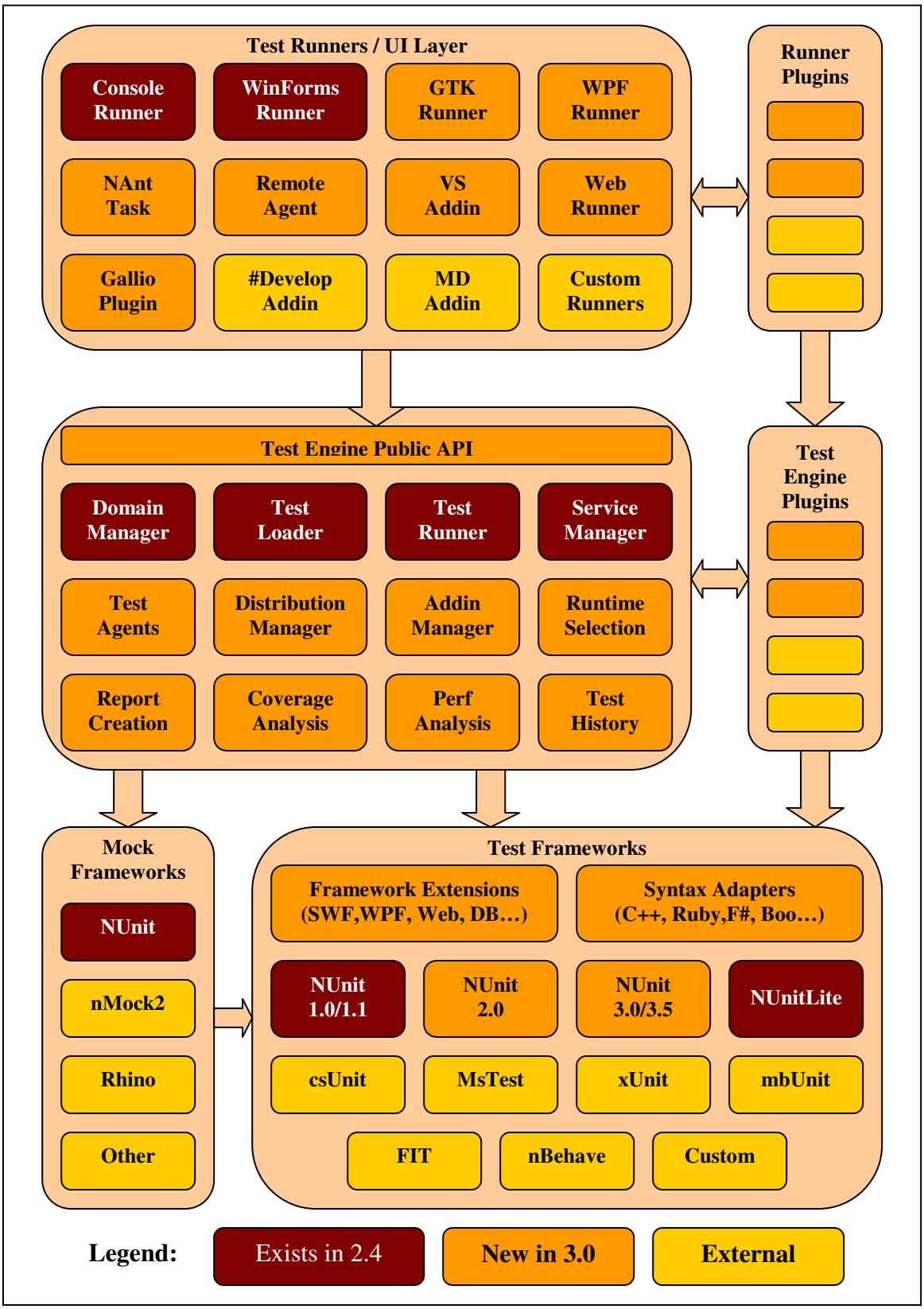

Diagram 1 – NUnit 3.0 Extended Platform

Test Runner Layer (continued)

There is a long-standing need for a runner that executes tests in an actual or simulated web server environment. While good practice calls for extracting as much functionality as possible into separately testable assemblies, more complex applications often contain code which can only be tested in such an environment. For that reason, NUnit 3.0 will feature a web runner, which allows tests to be executed on a web server, reporting results back to the desktop.

Each of the runners will have the option of participating in the NUnit plugin architecture. For the GUI runners developed directly by the NUnit team, this capability will be used to the max, allowing others to add GUI features that function by themselves or in conjunction with other plugins operating at the level of the test engine.

Test Engine Layer

The Test Engine Layer is the core of the NUnit platform. It provides a public API for use by applications that want to locate, load and run tests and display test results. Many aspects of the Test Engine are already present in NUnit 2.4, while others are new. I'll focus on the new features here.

NUnit 3.0 will support running tests in a separate process, allowing selection of the CLR version under which the test is to be run. It will also support distribution of test execution over a network to remote Test Agents. This facility is intended to be used in several distinct scenarios:

- Simple load sharing when test execution time is excessive
- Testing applications on multiple platforms
- Testing applications, which are themselves distributed

NUnit will support performance and coverage analysis, test result reporting and maintenance of a history of test results. These features will function only as called upon by the user. That is, we will not carry on data collection activities with the potential of impacting performance when the user simply wants to run tests.

Through use of plugins, NUnit will be able to support a wide variety of test types beyond low-level, isolated programmer tests. What is available in this area will be dependent on the interests of users and their willingness to contribute their efforts to creating them. Examples of the kinds of things we envision, some of which we will provide directly, are:

- Randomization of test execution order
- Deterministic test ordering for integration tests
- Parameterized (data-driven) tests
- Transactional test behavior
- Timed tests and parameterized timeout failures
- Dynamic test generation based on data input

- Repetitive test execution
- Tests written as non-managed code
- Test generation for legacy applications

Framework Layer

In NUnit 3.0, the NUnit framework itself – the assembly that is referenced by user tests – will be split along two dimensions. First, there will be separate framework assemblies for different version levels of the Common Language Runtime. Currently, we envision one assembly for .NET 1.0/1.1 and the Mono 1.0 profile, a second for 2.0 support and a third for 3.0/3.5 support. However, this split may change as we get into the newest releases. By splitting the framework in this way, we will be able to take advantage of newer features, and allow users to take advantage of them, without compromising basic support for older runtimes.

The second split we plan is between the core framework capabilities and the syntactic features that make it easy to access those features. A key example of this is the fluent interface introduced in NUnit 2.4 – the "Assert.That" syntax. One thing we learned through that experiment is that the same syntactic "sugar" does not work well for different language environments. Many of the 2.4 constructs are unusable or very difficult to use in other languages – C++ for example. By a combination of separate namespaces and separate assemblies, we will allow users to select the appropriate syntax for the work they are doing. Other people will be able to build on the syntax we provide or create entirely new syntactic overlays for their own purposes.

Through use of plugins in the Test Engine layer, NUnit will be able to recognize, load and run tests written using other frameworks. Our focus will be on facilitating the creation of plugins in support of external frameworks by the projects that produce those frameworks or by interested users. For frameworks where that sort of support is not possible – commercial frameworks, for example – we have the option of creating the plugins ourselves.

In some cases, individuals with an idea for a new framework may be able to create them more simply by writing an adapter on top of the NUnit framework itself. We will encourage and support this by giving priority to framework modifications that provide the necessary infrastructure for such projects.

NUnit will also provide or re-package some framework extensions for specific types of applications, including Windows Forms development, WPF development, web page access, XML testing, performance measurement and load testing. At this point, it is not yet possible to state which of these will involve existing third-party packages and which will be developed from scratch, because discussions are ongoing.

The NUnitLite framework will be supported running on devices and reporting results back to the Test Engine through a communications channel.

Various popular mock frameworks will be supported. One framework will be selected for use in by NUnit's own self-tests and will be packaged with NUnit.

Platform Support

NUnit 3.0 will continue to support tests using all versions of Microsoft .NET, the Mono CLR and the compact framework. Compact framework support will be through NUnitLite.

We will continue to support both Windows and Linux directly. We will make every attempt to fix problems that arise on other platforms, and will add direct support (testing our own builds on them) to the extent that we are able to find developers and testers with access to those platforms. Both 32-bit and 64-bit environments are supported.

The NUnit Framework - the part of NUnit that is called by user tests - will be available in versions for each CLR supported. The test engine and runners will be built with .NET 2.0 or higher.

Compatibility

NUnit 3.0 will be able to run tests written with any of the prior 2.x releases. Users will have the choice of two approaches:

1) Re-compile under the new framework.

2) Continue to run under the old framework.

Most tests should compile without change under the 3.0 framework, but those that make use of features currently marked as obsolete will require modification.

Most command-line options will continue to work with nunit-console. A few that are highly specialized may be removed, but we'll discuss this with the community before doing it.

License

In support of this broader platform, NUnit 3.0 will be released under a new license, which is currently being selected. We want a license that makes clear the ability of users to create and run tests without risk or restriction. We believe that our current license gives this right to users, but some companies have reported uncertainty, due to the lack of an explicit grant of those rights.

We are currently reviewing a number of Open Source licenses and will select the actual license to be used before the first Alpha releases of NUnit 3.0.

Next Steps

Because this is a platform for new ideas, we can only give you a general notion of the directions in which we believe it will grow. As more people are involved and as the work that has already been done is made available, we hope to see an influx of ideas that will expand the scope of possibilities enormously. After a period of review for this vision document, we will update it and publish a roadmap for feature implementation in the immediate future. Because much of the infrastructure is already under development, you can expect fairly rapid release cycles beginning in early 2008.