Gallio: Crafting a Toolchain

Jeff Brown
jeff.brown@gmail.com

About Me

- Jeff Brown
- Lead Software Engineer at Yellowpages.com

- Creator of Gallio Open Source Project
- Lead of MbUnit Open Source Project
- Coding is fun!

Outline

- Gallio and MbUnit
- Demo
- What is a Toolchain?
- Implementation Challenges
- Under the Hood
- Roadmap
- Questions

Gallio

- Gallio is a neutral test automation **platform**.
  - Open Source. (Apache License)
  - Microsoft .Net platform.
  - Aims to provide great tools integration for many different test frameworks.
  - Started in October 2007 as a spinoff from MbUnit.
- Current release: v3.0.5.
- Website: www.gallio.org

Gallio

- Vision: Gallio will be the foundation for a rich suite of interoperable tools.
  - Test frameworks.
  - Test runners.
  - Test case managers.
  - Test generators.
  - Test reports and analytics.
  - Test editors and IDE integration.
  - Continuous integration facilities.

Gallio

- Lingua franca for test tools.
  - Common object model.
  - Support for many different workflows.
  - Extensible.
  - Evolving.
  - Owned by the community.

- Objective: To unite, not to control.

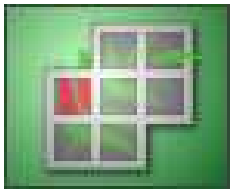
Gallio

- Tools support
  - Frameworks: CSUnit, MbUnit v2, MbUnit v3, MSTest, NBehave, NUnit, xUnit.net
  - Runners: GUI (Icarus), Command-line (Echo), TestDriven.Net, ReSharper, Visual Studio Test Tools, MSBuild, NAnt, PowerShell.
  - Other: CruiseControl.Net, TeamCity, TypeMock, NCover, Pex, AutoCAD
- 3rd party Contributions: DXCore runner (RedGreen), MSpec, and more…

Gallio

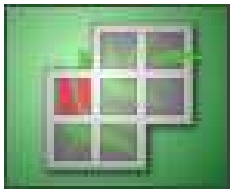
- Trivia
  - Original code name provided by Andrew Stopford was to be “Galileo” but it was corrupted to “Gallio” due to a misspelling in an early email thread.

MbUnit



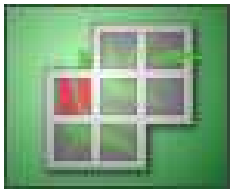
- MbUnit is a test automation **framework**.
  - Open Source. (Apache License)
  - Aims to provide a powerful framework for unit testing and integration testing for .Net.
  - Started by Jonathan “Peli” de Halleux in 2004.
  - Complete rewrite for MbUnit v3 in 2007/2008.
- Current release: MbUnit v3.0.5.
- Website: www.mbunit.com

MbUnit



- MbUnit is mostly NUnit compatible but improves on it in many ways:
  - Focus on usability and clear reporting.
  - Data-driven testing. (eg. [Row]-test, pairwise and combinatorial testing)
  - Supports unit testing and integration testing.
  - Easily extensible by creating new attributes.
  - Dynamic test structure.

MbUnit



- Trivia:
  - Original name was GUnit but that name turned out to already be taken.
  - Model based Unit Testing Framework.
  - Or if you prefer…
    - Much better Unit Testing Framework. ;-)

Gallio and MbUnit Demo

What is a Toolchain?

- A collection of tools that work together.
- Did you notice how many different interactions there were between tools in the demo?
  - A lot.

Implementation Challenges

- Test code is hostile!
  - By definition, subject under test may contain bugs.
  - Could kill the test process due to stack overflows, out of memory, and other side-effects.
- Test code must be isolated.
  - Run in separate process, ideally.
  - Run in separate AppDomain, at a minimum.
  - Be prepared to abort the test run.

Implementation Challenges

- Test frameworks define tests differently.
  - xUnit.Net: A fixture is just a .Net class, a test is just a .Net method.
  - NUnit: Tests are declared statically using metadata in a .Net assembly.
  - MbUnit: Tests are declared statically but data-driven test instances are generated dynamically.
  - RSpec: Tests are generated dynamically when a test script is evaluated.

Implementation Challenges

- Test frameworks define tests differently…
  - Some use XML.
  - Some use databases.
  - Some use collections of standalone programs.
  - Some generate tests on-the-fly from a model.
  - etc...
- Test representation must be flexible.
- Test platform must not be “opinionated” though it must support frameworks that are.

Implementation Challenges

- Test frameworks make assumptions
  - Working directory contains test assembly.
  - Application base directory is same as working directory.
  - Can resolve references to test assembly dependencies.
  - If a test assembly has an associated configuration file, it is loaded.
  - x86-only tests run in x86 mode on x64.
  - Full trust security model.

Implementation Challenges

- Extensibility model mismatches.
  - ReSharper wants plugins to be installed in C:\Program Files\JetBrains\ReSharper\...
  - Visual Studio Test Tools must be able to load custom test types from GAC or from Visual Studio Private Assemblies.
  - TestDriven.Net can load a test framework assembly from anywhere but it might not be able to resolve the references of that assembly.

Implementation Challenges

- Test platform must be prepared to establish its own private hosting environment regardless of how it is loaded by 3rd party tools.

Implementation Challenges

- Test hosts make assumptions
  - Everything is running in a single process (for debuggers and code profilers).
  - Test code does not have lasting side effects upon the host.
  - Test structure:
    - Dynamic metadata rich hierarchy (Gallio)
    - Static hierarchy rooted at project (ReSharper)
    - Static flat list (Visual Studio)
    - Ignored and not reported in results (TestDriven.net)

Under the Hood

Under the Hood: Test Model

- At test exploration time…
  - All we have is static metadata.
  - Some data may not be available.
  - Might not be able to execute code.
  - Create a **ITest** Tree to describe exploration.
- At test execution time…
  - We have all the information.
  - Create a **ITestStep** Tree to describe execution.

Under the Hood: Test Model

- Tests have:
  - Unique stable identifier.
  - Name.
  - Metadata.
    - Kind: Namespace, Fixture, Suite, Test, etc…
    - Description.
    - Documentation.
    - Arbitrary key/value pairs.
  - Children.
  - Dependencies.
  - Ordering.
  - Parameters.

Under the Hood: Test Model

- Caveats:
  - We might define new tests at runtime.
  - Tests might have dynamic substructure
    - eg. Instances of parameterized tests.
  - Test runners cannot assume static and dynamic test structure are the same (but many do..)

Under the Hood: Test Log

- Want to report test results uniformly.
- Simple document format.
- Primitives:
  - Streams (Containers)
    - Failures, Warnings, Console Output, etc...
  - Sections (Delimiters)
  - Markers (Embedded Metadata)
    - Diffs, Links, Highlights, Stack Traces
  - Text
  - Attachments

Under the Hood: Reflection

- Tests can be explored before compilation!
  - JetBrains ReSharper™ unit test runner
- Abstract reflection layer
  - Native .Net reflection wrappers.
  - ReSharper Code Model wrappers. (x2)
  - Visual Studio Code Model wrappers.
  - Cecil wrappers.
- Enables test explorer to be polymorphic.

Under the Hood: Reflection

- Abstract Reflection API
  - **IReflectionPolicy**
  - **ICodeElementInfo**
    - **IAssemblyInfo, IAttributeInfo, IConstructorInfo, IEventInfo, IFieldInfo, IGenericParameterInfo, IMemberInfo, IMethodInfo, INamespaceInfo, IParameterInfo, ITypeInfo**
  - **CodeLocation, CodeReference**
  - Access to Xml documentation comments.
- Unexpected bonus: Uniformity
  - IFunctionInfo: constructor or method.
  - ISlotInfo: field, property, parameter, or generic parameter.

Under the Hood: Pattern Test Framework

- Reflection-based frameworks are common.
- Demand for different syntaxes: BDD, etc…
- Typical solution:
  - Syntax adapters: wrap another test framework.
- Better solution:
  - Reuse code test exploration strategy.
  - Define completely custom syntax.
  - Better user experience and branding potential.

Under the Hood: Pattern Test Framework

- **IPattern**
  - Basic compositional unit of structure in a test.
    - “Abstract syntax”
  - Three basic methods:
    - IsPrimary: Does this pattern declare something new?
    - Consume: If this pattern is primary, produce something new from a given code element.
    - Process: Enrich something else created by another pattern using a given code element.

Under the Hood: Pattern Test Framework

- **[PatternAttribute]**
  - Associates a pattern with a code element.
  - Most MbUnit v3 attributes are Pattern Attributes.
    - You can make a custom framework the same way.

Under the Hood: Pattern Test Framework

- **MbUnit v3 Examples:**
  - **[Test]**: Primary pattern that creates a new tests.
  - **[SetUp]**: Primary pattern that registers the associated method in the “setup chain” of the containing fixture.
  - **[Description]**: Secondary pattern that adds descriptive metadata to a test or fixture.
  - **[Row]**: Secondary pattern that adds a new data source to a test or fixture.

Under the Hood: Pattern Test Framework

- Might help you write a new test framework…
  - Reflection-based test exploration.
  - General-purpose test execution engine.
  - Data binding.
  - etc...
- Or forget all of this and build it from scratch…
  - Just implement **ITestFramework**, **ITestExplorer**, and **ITestController** to explore and execute tests.

Lessons Learned

- Beware of hidden assumptions.
  - All test hosts are a little different.
  - Protect the test frameworks and tools from unanticipated variations in hosting.
- Version independence is mandatory.

Lessons Learned

- Consolidation of the tool chain is very useful.
- This is a lot harder than it looks!
  - If you plan to write your own test framework, consider using Gallio as the underlying platform.

Roadmap

- v3.0: First spike.
  - Self-host.
  - Integrate with many different environments.
  - Challenge assumptions. Gain experience.
- v3.1: Generalize and consolidate.
  - Support non-.Net languages and workflows.
  - Simplify extensibility contracts.
  - Performance.

Gallio Futures

- Gallio is the nucleus of a growing suite of interoperable test tools.
  - Ambience: A database for persistent test data.
  - Archimedes: An automation test case manager including distributed test agent and test data warehouse.
- Dynamic languages, scripting languages, test DSLs and non .Net frameworks in v3.1.

MbUnit Futures

- More built-in assertions.
- Provide “mixins” to assist with integration of 3rd party libraries such as Rhino.Mocks, Selenium and WatiN.
- ASP.Net hosting.
- .Net Framework 4.0 extensions.

Ongoing and Upcoming Projects

- Gallio Book
- Mono Support
- Test Case Manager (Archimedes)
- Test Data Warehouse
- Distributed Test Runner
- Modeling Language for Integration Tests

Volunteers Wanted!

- What do **you** want to do?
  - Build your own stuff using Gallio.
  - Add new stuff to Gallio itself.
  - Promote Gallio.
  - Offer feedback and suggestions.
  - Help us write the Gallio Book.

Questions?