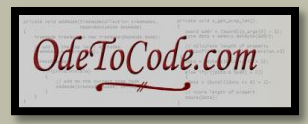
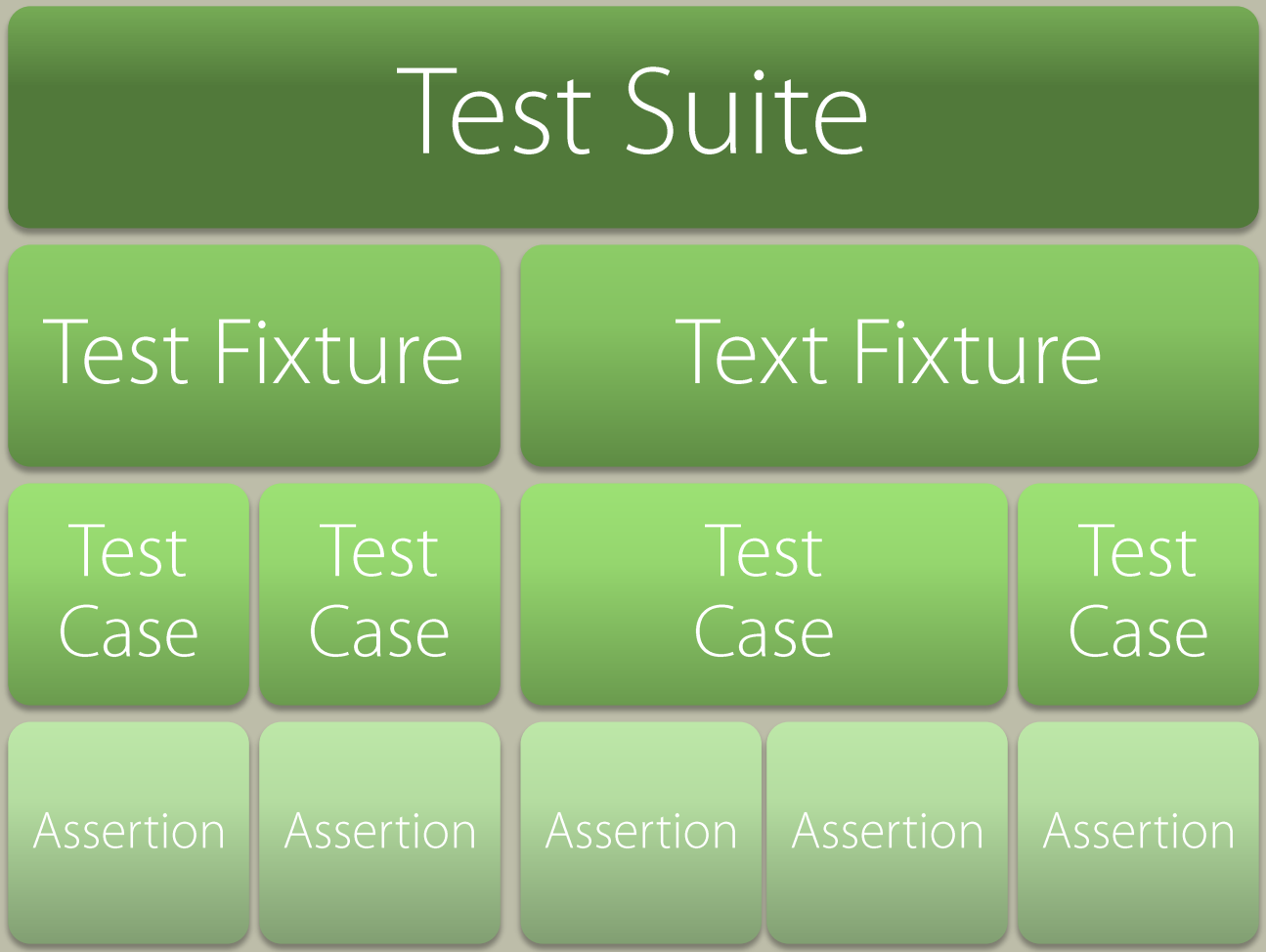
Test Driven Development

Test First Development in .NET



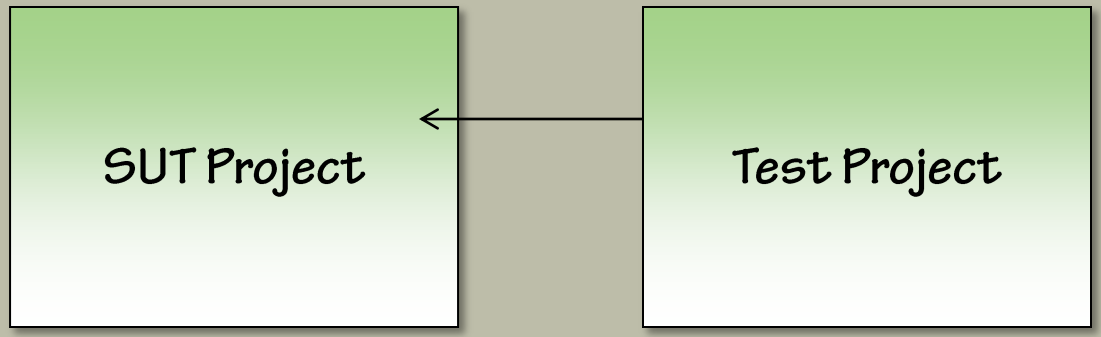
- Test Organization
- Strategies & Tips
- Code coverage and the Sad Path

xUnit Architecture



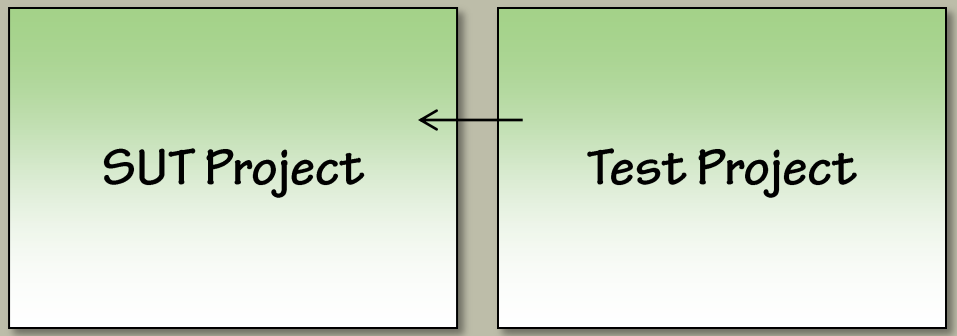
A Test Project

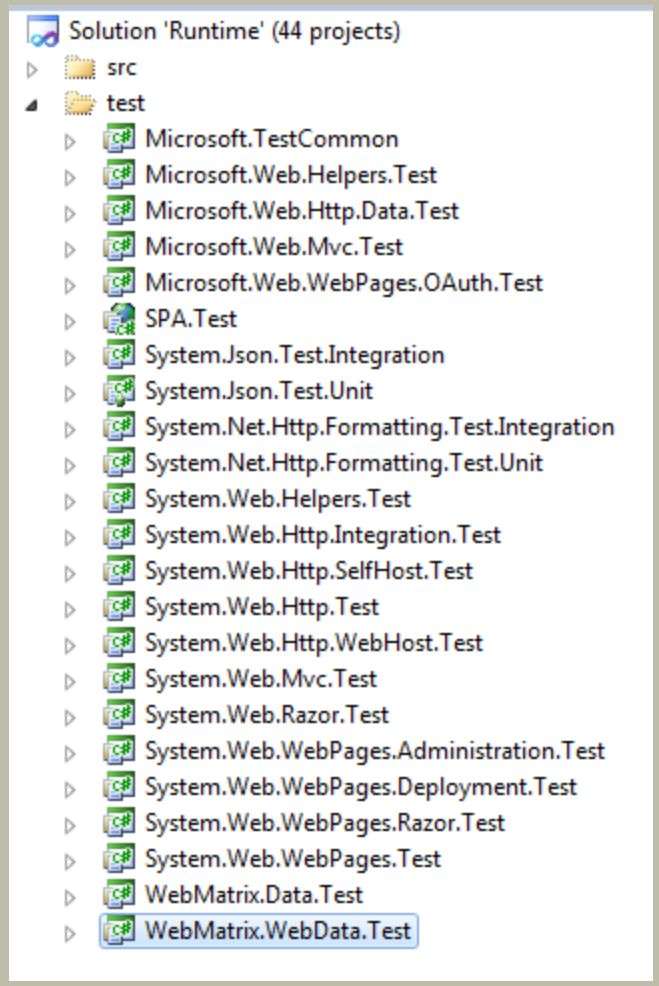
- Test only the public API
- Don't tie tests to implementation details.
- Think like a client



Test Project Organization

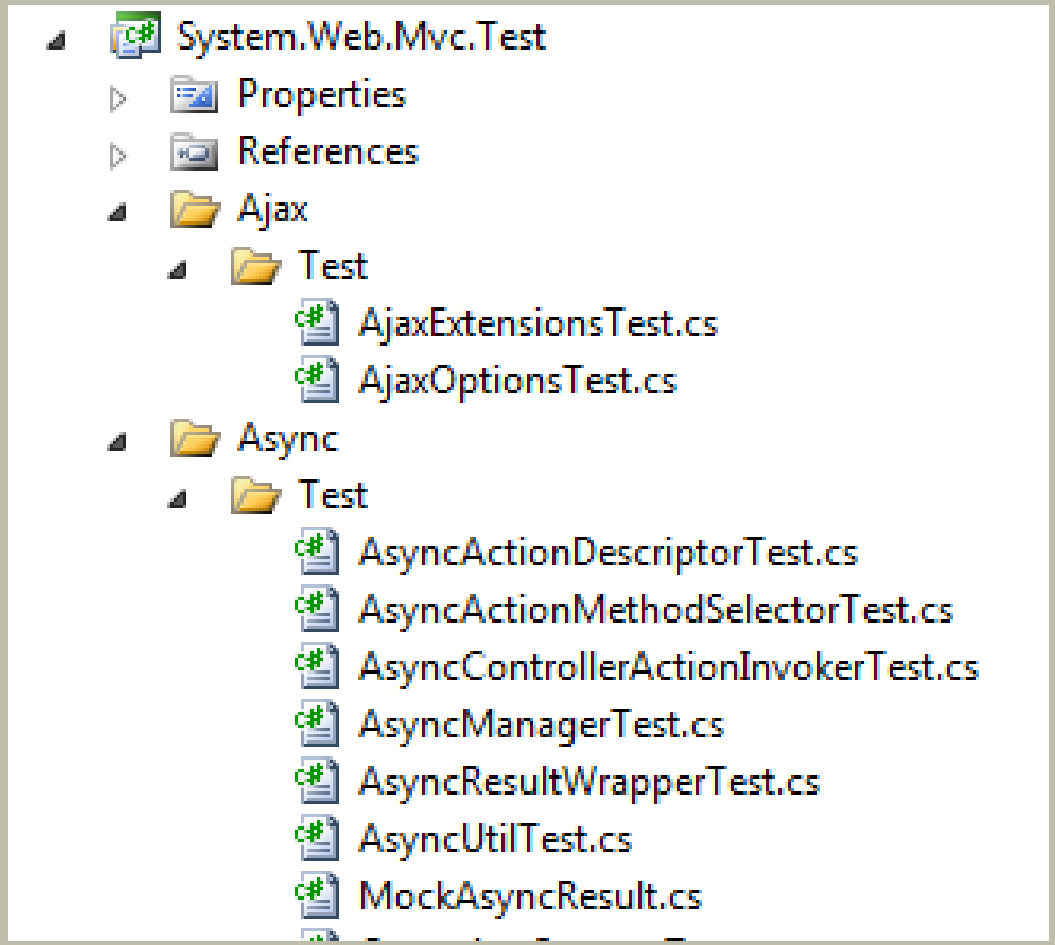
- Test Project Per Solution?
- Test Project Per Deployble Unit?
- No Separate Test Project?





Test Organization

- Fixture per class?
- Fixture per behavior?



Test Method Organization

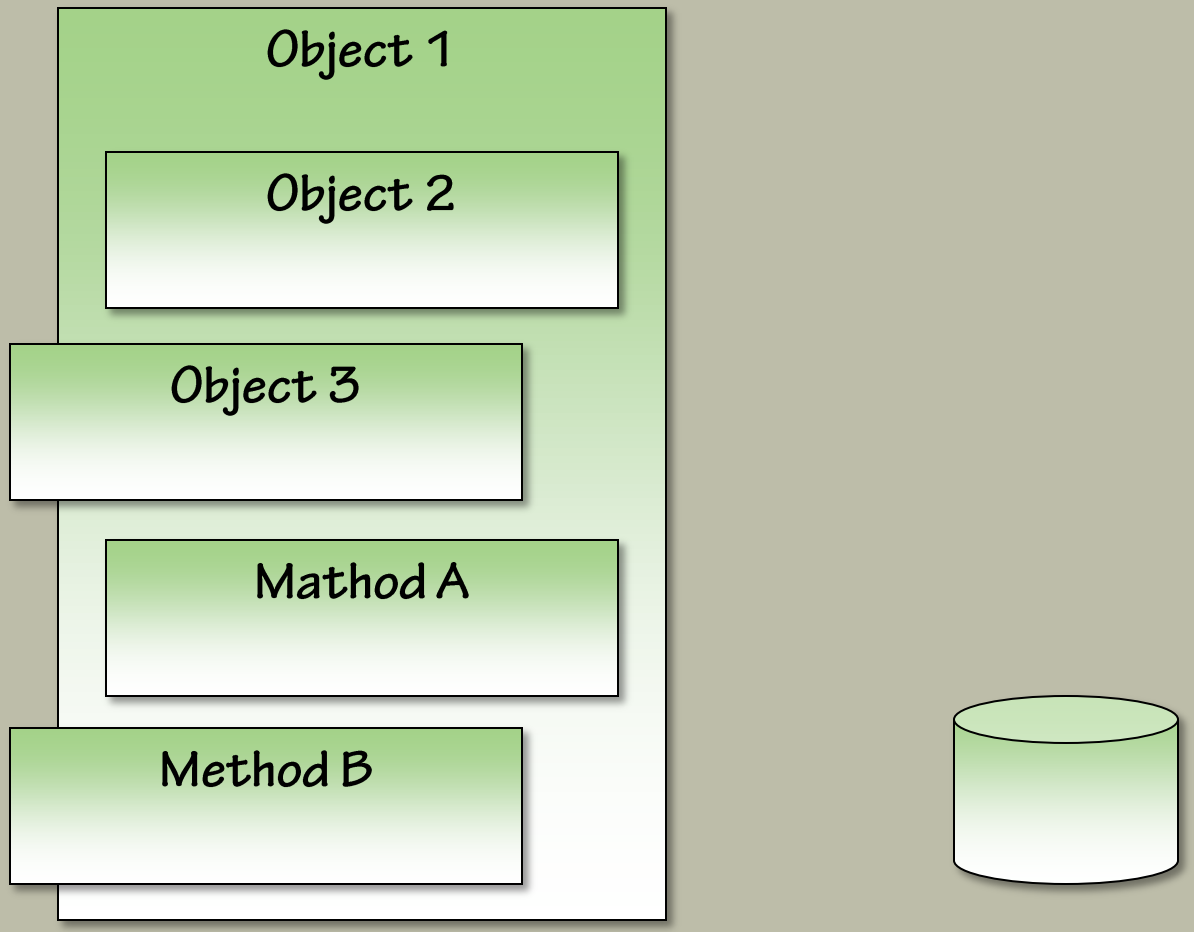
- Arrange
- Act
- Assert
  - One logical assertion per test!

```
[Fact]
public void GlobalizationScriptWithNullCultureName()
{
    // Arrange
    Mock<CultureInfo> xssCulture = new Mock<CultureInfo>("en-US");
    xssCulture.Setup(culture => culture.Name).Returns((string)null);

    AjaxHelper ajaxHelper = GetAjaxHelper();
    AjaxHelper.GlobalizationScriptPath = null;

    // Act
    MvcHtmlString globalizationScript = ajaxHelper.GlobalizationScript(xssCulture.Object);

    // Assert
    Assert.Equal(@"<script src=""~/Scripts/Globalization/.js"" type=""text/javascript""></script>",
}
```

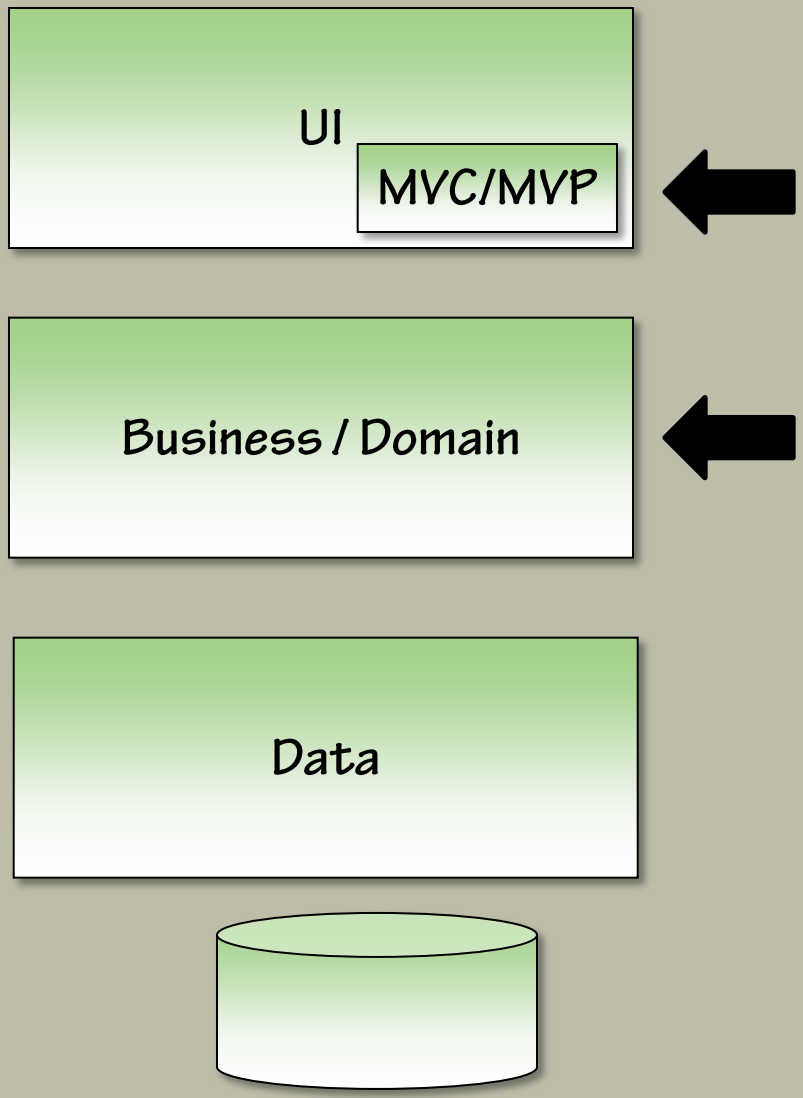
Test Code Is Important, Too!

- Keep it maintainable and ***readable***
- Apply DRY (though maybe not to the same extreme)

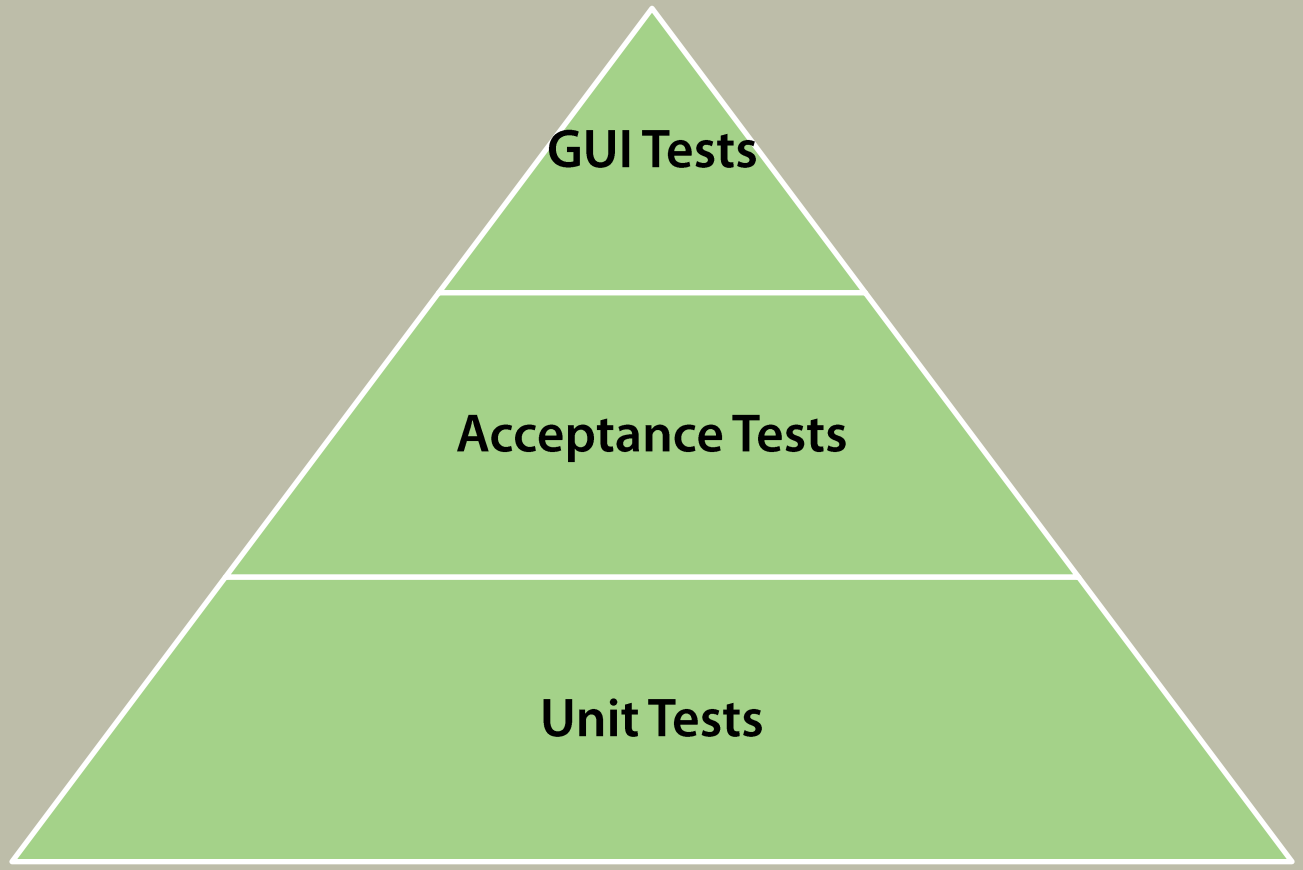
Unit Tests versus Integration Tests



Unit Testing Efforts



Test Pyramid



Test Lifecycle

- Assembly Initialize
- ClassInitialize
- TestInitialize
- TestCleanup
- ClassCleanup
- AssemblyCleanup

```
[TestClass]
public class AssemblyInitializer
{
    [AssemblyInitialize]
    public static void AssemblyInitialize(TestContext context)
    {

    }
```

Happy Path

- How most developers think
- Where we almost always start
- The optimal path of code execution given proper input

Sad Path

- An undesired path of code execution

Often caused by

- Bad method inputs
- Un-foreseen state

Why the Sad Path Matters

- **These things live in the sad path:**
  - Bugs
  - Unknown behaviors
  - Resource leaks
  - Security vulnerabilities

Remember GIGO?

- **We must assure good input**
- **Think of this as "coding defensively"**
- **For each method we create, consider "What would happen if someone … ?"**

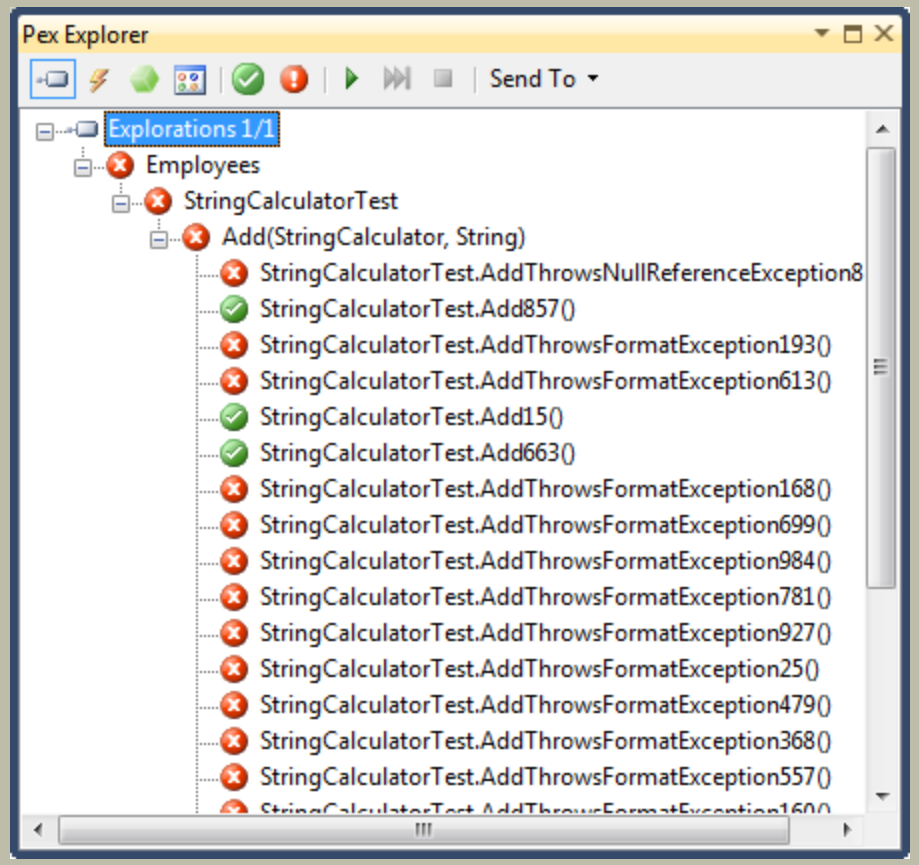
Exceptions

- Exceptions won't halt the test runner
- Use ExpectedException to test for exceptional conditions

```
[TestMethod]
[ExpectedException(typeof(ArgumentException))]
public void Test()
{
    // ...
    throw new ArgumentException();
}
```

Pex

- If you really want to catch the edge cases …



Principles

- Write the tests first
- Use the front door
- Communicate the intent
- Don’t modify the SUT
- Isolate the SUT
- Keep tests independent
- Minimize test iverlap
- Minimize Untestable code
- No test logic in production code
- One logical assertion per test

Summary

- Organization
- Life cycle
- Exceptions
- Principles