Cut Unit Test Costs with Compuware DevPartner and Typemock Isolator

DevPartner Studio Professional Edition is an award-winning suite of software development and testing tools that enable Windows application development teams to build reliable, high-performance applications, components and web services for Microsoft .NET and native Windows platforms. DevPartner Studio automatically detects and diagnoses software errors and performance problems early in the development process as cost-effectively as possible, providing built-in expert coding advice, coding standards and best practices to improve development skills and ensure software reliability and performance. DevPartner Studio enhances Microsoft Visual Studio and Visual Studio Team System with advanced capabilities that allow development organizations to improve software quality, adopt coding standards and best practices, and maximize developer productivity.

Typemock Isolator (www.typemock.com) is a .NET unit testing tool which enables developers to write simpler and more maintainable unit tests. Isolator provides capabilities for creating fake objects and testing against them, instead of testing against external code. This helps the test author to eliminate dependencies, test their code in isolation and achieve better test readability and code coverage.

Using DevPartner Studio (release 9.01 or later) with Typemock Isolator (release 5.2 or later) can significantly reduce the cost of any continuous unit testing effort. Typemock lets .NET developers easily adopt .NET unit testing practices without difficult and time consuming code refactoring. Any .NET developer can start unit testing code immediately, with minimal learning curve and with an increase in test coverage. The profiling power of DevPartner Studio provides detailed coverage, performance and memory usage data during Isolator test runs. When unit testing with DevPartner Studio and Typemock's tools, you find bugs earlier, dramatically reducing defects discovered in the QA cycle. This makes your integration and system testing, and the final QA test cycle faster and more predictable.

Product Installation

Refer to each product's web site for more information on specific product installation.

Note: Typemock Isolator requires .NET framework version 2.0 or later.

Setting up a Test Project

Using unit tests written with Typemock Isolator requires referencing Typemock API assemblies in your test project. For C# projects add references to Typemock.dll and Typemock.ArrangeActAssert.dll. For VB.NET projects add references to Typemock.dll and Typemock.Isolator.VisualBasic.dll.

To add the references to Visual Studio:

1. Select the test project in the **Solution** view.
2. Right-click and select **Add Reference**.
3. Select the **.NET** tab.
4. Select the **Typemock Isolator** assemblies in the .NET tab and click **OK**.
5. Add a Typemock directive to your test file code. In the Solution View, right-click on the applicable C# or VB.NET test file and select **View Code**.

6. Add the appropriate Typemock directive to the code.

   - For C#:

     ```
     using TypeMock.ArrangeActAssert;
     ```

   - For VB.NET

     ```
     Imports Typemock.Isolator.VisualBasic
     ```

Writing Unit Tests with Typemock Isolator

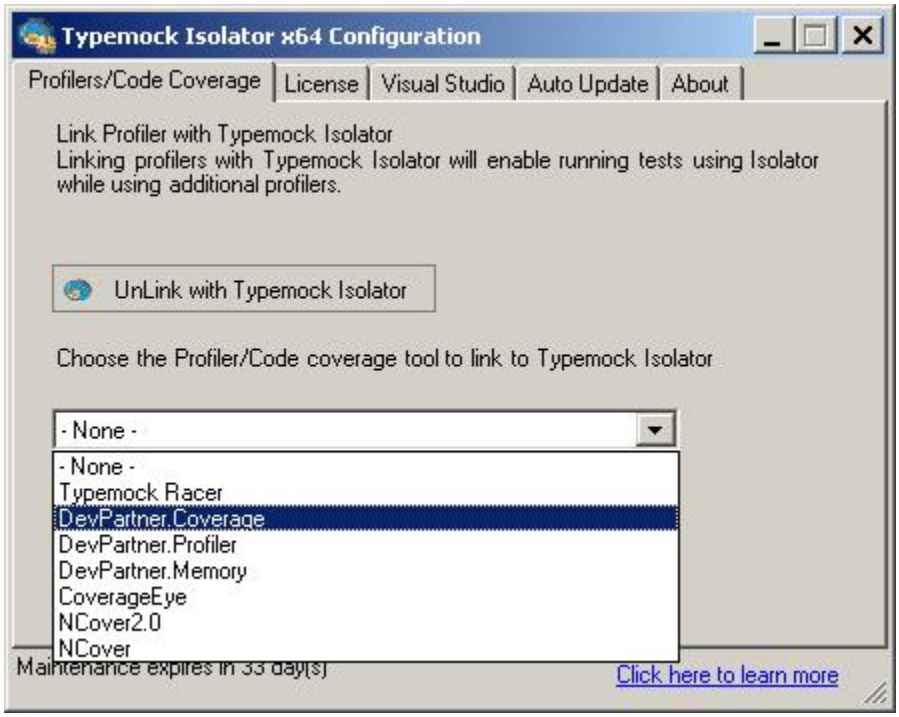
When writing a unit test with Typemock Isolator, developers should eliminate external code dependencies and test the unit in isolation. When discovering external code dependencies, use the Isolator API to create a fake object instead of the object depended on by the code. See the section "Writing a Unit Test with Typemock Isolator" for a complete example on writing a unit test with Typemock Isolator.

Linking Typemock Isolator 5.2 with Compuware DevPartner 9.0.1

Typemock Isolator acts as a profiler that intercepts calls to fake objects and methods within DevPartner Studio, whose set of performance, coverage, and memory analysis features also acts as a profiler. The profilers must be linked to run Typemock Isolator enabled unit tests through DevPartner. Use the Typemock Isolator configuration utility or the command line runner to link the profilers.

Linking Using the Typemock Isolator Configuration Dialog Box

1. In Windows, click **Start > All Programs > Typemock > Isolator > Typemock Isolator Configuration** to open the **Typemock Isolator Configuration** dialog box.
2. Select the **Profilers/Code Coverage** tab.
3. Select the profiler you want to link to. Available DevPartner profilers are DevPartner.Coverage, DevPartner.Profiler (for performance profiling) and DevPartner.Memory.
4. Click **Link with Typemock Isolator** to link.



You can now run Isolator-enabled unit tests using DevPartner.

To unlink a selected profiler/code coverage tool:

1. In the Typemock Isolator Configuration dialog box, select the **Profilers/Code Coverage** tab.
2. Press **UnLink with Typemock Isolator** to unlink.

Linking and Starting Tests from the Command Line

TMockRunner.exe is a command line utility provided with the Typemock Isolator installation. To link and run tests with DevPartner use TMockRunner as follows:

```
TMockRunner.exe –link <Profiler Name> -first <DevPartner Command> <Test Runner> <Test Assembly>
```

The following table lists the TMockRunner.exe command parameters and a description of each.

Parameter	Description
Profiler Name	For DevPartner integration this can be: - DevPartner.Coverage - DevPartner.Profiler - DevPartner.Memory
DevPartner Command	The path to DPAnalysis.exe and command line parameters to configure its run
Test Runner	The console command to run the test runner, i.e. mstest.exe*, nunit-console.exe, etc.
Test Assembly	The assembly containing the tests

* Currently mstest.exe is not supported by Typemock Isolator

The following shows an example of how to run DevPartner Coverage with Typemock Isolator from the command line.

```
TMockRunner.exe -first -link DevPartner.Coverage DPAnalysis.exe /COV /P
nunit-console.exe
```

The named test assembly executes its test scripts against the test application. The tests execute against the fake objects set up in the test using the Typemock Isolator API. Test execution collects data as defined by the profiler type specified in the command line.

Multiple test sessions can be executed against the test application. When testing completes, you can:

- View and analyze test data using DevPartner Studio in the Visual Studio IDE
- Combine data collected from multiple sessions into one session file
- Identify performance issues
- Use coverage results to increase code coverage of test suites

Refer to the ***Understanding DevPartner Guide*** for more information.

Chapter 4: Automatic Code Coverage Analysis instructs how to use the Coverage Analysis viewer, merge coverage session data, and view source code to identify application functionality that requires more coverage.

Chapter 6: Automatic Performance Analysis instructs how to use the Performance Analysis Viewer, merge performance data from multiple sessions, and identify performance issues by analyzing call graph and other performance data.

Chapter 7: In-Depth Performance Analysis gives a more detailed approach to performance issue resolution and includes usage scenarios that provide a practical methodology for resolving complex performance issues.

Writing a Unit Test with Typemock Isolator

In this example, an Authentication class internally calls various providers: a data access layer, a cryptography class and a logger:

```
public bool IsAuthenticated(string name, string password)
{
    Logger.Log(Logger.NORMAL, "Entering Authentication");
    try
    {
        string storedHash = DataAccess.GetUserPasswordHash(name);
        string providedHash = CryptoClass.DoHash(password);
        bool isOk = storedHash == providedHash;
        if (isOk)
        {
            Logger.Log(Logger.NORMAL, "User Authenticated " + name);
        }
        else
        {
            Logger.Log(Logger.SECURITY, "User login failed " + name);
        }

        return isOk;
    }
    catch (Exception ex)
    {
        Logger.Log(Logger.FAILED, "Login system failure", ex);
        throw;
    }
}
```

With Typemock Isolator, eliminate external dependencies and test only the IsAuthenticated code. To do this, use the Isolator API to fake method calls on DataAccess, CryptoClass and Logger:

```
[Test]
public void TestAuthenticationSuccess_CorrectMessageLogged()
{
    // ignore any calls to the logger regardless of parameters
    Isolate.WhenCalled(() => Logger.Log(Logger.Any, "")).IgnoreCall();
    // set the data access and crypto provider to return the same hashes
    Isolate.WhenCalled(() => DataAccess.GetUserPasswordHash("name")).
        WillReturn("ABC");
    Isolate.WhenCalled(() => CryptoClass.DoHash("password")).
            WillReturn("ABC");

    // call the code under test
    Authentication target = new Authentication();
    bool result = target.IsAuthenticated("TestUser", "password");

    // authentication should pass
    Assert.IsTrue(result);
    // verify the correct log entry has been written
    Isolate.Verify.WasCalledWithExactArguments(() =>
    Logger.Log(Logger.NORMAL, "User Authenticated TestUser"));
}
```

Writing the same test with Typemock Isolator's VB.NET friendly API:

```
<Test()> _
Public Sub TestAuthenticationSuccess_CorrectMessageLogged()
    ' ignore any calls to the logger regardless of parameters
    Using TheseCalls.WillBeIgnored()
        Logger.Log(Logger.Any, "")
    End Using

    ' set the data access and crypto provider to return the same hashes
    Using TheseCalls.WillReturn("ABC")
        DataAccess.GetUserPasswordHash("name")
        CryptoClass.DoHash("password")
    End Using

    ' call the code under test
    Dim target As New Authentication
    Dim result As Boolean = target.IsAuthenticated("TestUser", "password")

    ' authentication should pass
    Assert.IsTrue(result)

    ' verify the correct log entry has been written
    Using AssertCalls.HappenedWithExactArguments()
        Logger.Log(Logger.NORMAL, "User Authenticated TestUser"))
    End Using
End Sub
```