AutoNMock. Плагин для ReSharper.

**Цель создания плагина:**

1. Получить дополнительный опыт в разработке.
2. Снизить время на разработку и поддержку unit-тестов.
3. Приобрести опыт создания плагинов для ReSharper.

**Описание плагина:** плагин «AutoNMock» (далее плагин) предоставляет возможность автоматически создавать тестируемый класс, замещая его зависимости моками NMock.

**Идеальный сценарий использования плагина (описывает функциональность, к которой стремимся, но которая, скорее всего, не будет реализована полностью):**

**Действие 1.1.** Программист с помощью контекстного меню и команды «Add item» создает класс с unit-тестами (далее класс тестов). Вместе с unit-тестами автоматически создается сам тестируемый класс в той же директории. Имя класса совпадает с unit-тестом, только без суффикса «Tests» на конце.

internal sealed class <Class>

{

}

В классе тестов создается вложенный класс Mocks

private sealed class Mocks{

{

public Mocks(MockFactory mockFactory)

{

Sut = new<Class>();

}

public<Class> Sut { get; private set; }

}

, где Class – тестируемый класс.

Также в классе тестов создается unit-тест со следующим кодом:

[TestMethod]

public void NewTest()

{

var mocks = new Mocks(MockFactory);

}

**Действие 1.2 (альтернатива действию 1.1).** Программист с помощью контекстного меню и команды «Add Mocks class» создает в существующем классе тестов вложенный класс Mocks

private sealed class Mocks{

{

public Mocks(MockFactory mockFactory)

{

Sut = new Sut();

}

public <Class> Sut { get; private set; }

}

**Действие 2.** Программисту для написания теста потребовалось добавить новую зависимость в класс. Он вызывает контекстное меню и выбирает пункт «Add dependency». Результат:

private sealed class Mocks{

{

public Mocks(MockFactory mockFactory)

{

<DependencyName1> = mockFactory.CreateMock<DependencyInterface1>();

Sut = new Sut(<DependencyName>);

}

public <Class> Sut { get; private set; }

public <DependencyInterface1> <DependencyName1> { get; private set; }

}

, где DependencyInterface1 – интерфейс, который описывает зависимость, DependencyName1 – имя зависимости.

Также в тестируемом классе будет добавлено поле readonly для новой зависимости, а в конструкторе будет добавлен нужный параметр:

internal sealed class <Class>

{

public <Class>(<DependencyInterface1> <dependencyName1>)

{

\_<dependencyName1> = <dependencyName1>;

}

private readonly DependencyInterface1 \_<dependencyName1>;

}

где dependencyName1 – то же, что и DependencyName1, только с маленькой быквы.

Программист повторяет команду «Add dependency». Результат:

private sealed class Mocks{

{

public Mocks(MockFactory mockFactory)

{

<DependencyName1> = mockFactory.CreateMock<DependencyInterface1>();

<DependencyName2> = mockFactory.CreateMock<DependencyInterface2>();

Sut = new Sut(<DependencyName>);

}

public <Class> Sut { get; private set; }

public <DependencyInterface1> <DependencyName1> { get; private set; }

public <DependencyInterface2> <DependencyName2> { get; private set; }

}

**Действие 3.** Программист удаляет зависимость с помощью команды контекстного меню «Delete dependency».

**Действие 4.** Программист обновляет класс Mocks в соответствии с тестируемым классом. Он вызывает контекстное меню и выбирает пункт «Auto configure dependencies»

**Сценарий в минимально-достаточном функционале:**

**Действие 1:** Программист в классе, помеченном атрибутом TestClass, создает переменную, куда помещает экземпляр тестируемого класса. Наводит мышью на переменную и с помощью команды Решарпера «Add Mocks class» создает вложенный класс Mocks (см. выше).

**Действие 2:** Программист с помощью команды Решарпера «Auto configure dependencies» конструирует класс Mocks в соответствии с тестируемым классом.