Использование mock-объектов в модульном тестировании

Все, в том числе и ложь, служит истине.
Франц Кафка

Гавришов Юрий
TulaSoft
www.sqlaccessories.com

TulaDev.NET

«Без тестов жить нельзя на свете, нет»

Тесты хорошо, а TDD лучше

Результат применения:

- Рабочий протестированный код
- Способствует хорошему дизайну.
- Самодокументация

Unit-тесты. Когда невыгодно/неудобно/не получается использовать

- Слишком дорого
- Невысокая цена ошибки
- Надо было сделать вчера

Ситуации, когда сложно использовать юнит-тесты

Нужно протестировать класс, который взаимодействует с

- с базой данных
- с внешним устройством
- с файловой системой
- с внешним сервисом

То есть

- Если класс взаимодействует с внешней средой, от которой его надо изолировать.
- Если класс работает с объектом, методы которого отрабатывают немгновенно.

Особый случай:

- Если класс работает с объектом или группой объектов, у которых сложная инициализация.

Выход – использовать поддельные объекты

Пробуем:

- Сделаем вид, что мы не знаем про моки
- Ура, используем мок-фреймворк

Разновидности поддельных объектов (test doubles)

- Dummy
- **Test Stub**
- Test Spy
- **Mock object**
- Fake Object

Behavior vs state verification

- Mock-стиль проверки (behavior verification) *мы проверям, как происходит взаимодействие объектов.*
- Классический стиль проверки (state verification) *мы проверяем результаты взаимодействия.*

Принцип работы и ограничения «классических» mock-объектов

Возможности и ограничения:

- Можно мочить классы и интерфейсы.
- В классах можно подменять методы и свойства.
- Классы не должны быть sealed.
- Свойства и методы должны быть виртуальными (public or internal)

Существующие фреймворки

- NMock, Nmock2 (RR)
- Rhino Mocks (AAA, RR, ...)
- Moq (AAA)
- Microsoft.Moles
- TypeMock

Record-Replay syntax (RR)

- (пример кода на Rhino.Mocks)

Возможности Moq

Последнее обновление — август 2010. Текущая версия — 4.0.

Не поддерживает Record/Replay.

Минимальная версия .NET – 3.0.

- Один простой вариант использования: mock = new Mock<ICommand>();
- Параметры: явное указание, любые, диапазон, регулярное выражение (пример)
- Генерация исключений (пример)
- Ruturns: значение, отложенная инициализация и делегат (пример)
- Callback. Позволяют накапливать параметры вызванных методов.
- Verification (пример)
- Возможность настройки поведения при помощи перечисления MockBehavior (примеры)
- Можно задавать ожидания по умолчанию в SetUp и переопределять их в тесте
- Перегрузка protected методов (пример)
- Как «мочить» internal сущности (пример)

Microsoft.Moles

- Не является классической mock-библиотекой.
- Может переопределять все, что угодно (пример)
- Можно переопределять члены системных типов
- Есть только заглушки, моков нет
- Можно использовать в NUnit и т. д.
- Подробнее о возможностях (пример)

Microsoft.Moles: заглушки и моли

Feature	Stub types	Mole types
Detour mechanism	Virtual method overriding	Runtime instrumentation
Static methods, sealed types	No	Yes
Internal types	Yes	Yes
Private methods	No	Yes
Static Constructors and Finalizers	No	Yes
Performance	Fast	Slower
Abstract methods	Yes	No

Мы можем указывать, какие типы хотим переопределять

```
<Moles xmlns="http://schemas.microsoft.com/moles/2010/">
<Assembly Name="mscorelib">
  <StubGeneration>
    <Types>
      <Clear />
      <Add Namespace="System!" />
      <Add Namespace="Express" />
      <Add Namespace="SomeUtils*" />
      <Remove TypeName="NotUsedClass" />
    </Types>
  </StubGeneration>
 </Assembly>
</Moles>

  <StubGeneration>
    <Types>
      <Clear />
      <Add AbstractClasses="true"/>
    </Types>
  </StubGeneration>
```

Можно управлять поведением

Чтобы изменить поведение объекта, делаем так:

```
stub.InstanceBehavior = BehavedBehaviors.DefaultValue;
```

- **MoleBehaviors.DefaultValue** — незамещенные члены класса будут замещены пустым делегатом и возвращать дефолтное значение типа возвращаемого результата
- **MoleBehaviors.NotImplemented** — при обращении к незамещенному члену будет возникать исключение *NotImplementedException*
- **MoleBehaviors.Fallthrough** — вызовы к незамещенным членам будут обработаны согласно оригинальной реализации их в замещаемом классе

Microsoft.Moles: выводы

- Мощное средство изоляции
- Но как всегда есть недостатки:
  - Не хватает моков
  - Накладные расходы на поддержание «теневых» сборок
  - Некрасивые правила формирования имен
  - Тормозит рефакторинг

TypeMock

- Платная библиотека для написания тестов в изоляции.
- Лишена некоторых недостатков Microsoft.Moles
- Осуществляет перехват вызовов на уровне всего приложения
- Имеется графический тул

Промежуточные выводы

- Использовать Moles и Typemock не всегда полезно
- При построении новой логики лучше пользоваться DIP и другими инструментами

Выводы

Использование моков и заглушек:

- расширяет применимость юнит-тестов
- позволяют легко тестировать объекты, не имеющие состояния
- упрощают setup-методы тестов
- позволяют тестировать классы в изоляции

Ссылки

- Проект Moq
  http://code.google.com/p/moq/
- Проект Rhino Mocks
  http://www.ayende.com/projects/rhino-mocks.aspx
- Microsoft.Moles
  http://research.microsoft.com/en-us/projects/pex/default.aspx
- Изолятор TypeMock
  http://www.typemock.com/typemock-isolator-product3