Автоматизация тестирования

Лекция 11

Автоматизация тестирования

- Возрастание роли тестирования в процессе разработки ПО потребовало замены традиционного «ручного» тестирования автоматизированным, основанным на использовании специальных инструментальных средств

Автотесты

- Основная идея автоматизированного тестирования заключается в использовании автотестов – записанных на специальных скриптовых языках действий по проверке качества программ
- Современные средства автоматизации позволяют вести запись действий тестировщика и создавать заготовку для автотеста

Преимущества автоматизации

- Экономия времени – программа-робот гораздо быстрее перебирает тестовые варианты, чем любой человек
- Исключение человеческого фактора – вероятность совершения ошибки при выполнении человеком рутинных операций достаточно высока

Преимущества автоматизации

- Отсутствие необходимости в графическом пользовательском интерфейсе – на ранних этапах развития программного продукта интерфейс, как правило, еще не согласован; это существенно также при тестировании обмена данными по протоколам
- Наличие инструментария фиксации ошибок и результатов – это позволяет моделировать различные ошибочные ситуации, строить любые отчеты и диаграммы

Преимущества автоматизации

- Возможность эмулировать многопользовательскую работу – если рабочей нормой считается одновременное обращение к приложению нескольких тысяч пользователей, то средства автоматизации являются единственным способом решить проблему нагрузочного тестирования

Недостатки автоматизации

- Временные затраты на создание, поддержку и тестирование (!) тестов – автоматизированное тестирование всегда начинается с тестирования вручную, поскольку необходимо показать роботу, как, что и с чем он должен делать
- Неприменимость к некоторым объектам, оцениваемым субъективно – с помощью автомата нельзя протестировать, например, эргономику интерфейса приложения

Недостатки

- Необходимость программистских навыков у тестировщика – настоящая профессиональная автоматизация тестирования невозможна без работы непосредственно с кодом тестового скрипта
- Чувствительность к среде, программному и аппаратному окружению тестируемого приложения - один и тот же тест одной и той же версии повторно может проходить совершенно иначе, чем в первый раз

Типы тестирования

- Существуют три типа тестирования, которые можно автоматизировать:
  - **функциональное** (в том числе модульное, или unit-тестирование),
  - **регрессионное** (проверка работоспособности старого функционала и отсутствия ранее исправленных дефектов в новых версиях)
  - **нагрузочное** (поведение приложения под рабочей и стрессовой нагрузкой, влияние работающего приложения на системное окружение).

Некоторые принципы

- Не следует пытаться автоматизировать все тесты, т.к. наиболее простые из них вполне могут быть выполнены в «ручном» режиме
- Средства автоматизации – это всего лишь инструмент, поэтому особое внимание необходимо уделять качеству тест-плана
- Аккуратное и адекватное планирование - залог успеха автоматизации

Планирование функционального тестирования

- В качестве основы для планирования этого типа тестирования используются явные и неявные функциональные требования к программному продукту
- Функциональные требования разделяют по степени критичности и начинают планирование тестов для самых критичных пользовательских бизнес-прецедентов

Планирование нагрузочного тестирования

- Этот вид тестирования имеет три основные цели:
  - убедиться, что при той или иной нагрузке в работе приложения не возникает сбоев, т. е. отсутствуют ошибки;
  - проверить, сохраняется ли с ростом нагрузки эргономичность приложения;
  - поиск опасных тенденций для системных ресурсов клиента и сервера

Планирование нагрузочного тестирования

- Выделяют три уровня нагрузки:
  - **минимальная** нагрузка (один пользователь) позволяет проверить, что приложение в принципе работоспособно
  - **рабочая** (некоторое количество клиентов, считающееся штатным) - когда приложение должно вести себя безукоризненно
  - **стрессовая** или **пиковая** нагрузка, которую приложение должно выдерживать в принципе
- Необходимо планировать тестирование для каждого из этих видов нагрузки

Средства функционального тестирования

- **Mercury QuickTest**
  - Мощное средство компании Mercury, обладающее удобным и понятным пользовательским интерфейсом для создания тестов без ручной правки скрипта
- **Mercury WinRunner**
  - От QuickTest оно отличается тем, что приходится много вручную работать с кодом, написанным на специальном языке TSL
- **Segue SilkTest**
  - Интересное и относительно удобное средство, предлагаемое компанией Segue Software, предоставляющее широкие возможности для ручной работы со стандартными и нестандартными объектами на объектно-ориентированном языке 4Test

Средства нагрузочного тестирования

- **Mercury LoadRunner**
  - Очень удобный инструмент - однозначный лидер, обладающий широчайшим спектром возможностей
- **Segue SilkPerformer**
  - Хорошее средство со своими достоинствами и недостатками
- **RadView WebLoad**
  - Неплохая программа компании RadView Software для тестирования Web-приложений, но не более того

Утилита NUnit

- Для модульного тестирования применяются специальные утилиты, позволяющие сразу запустить все тесты и увидеть результат
- Одной из наиболее популярных из них является свободно распространяемая утилита NUnit
- Первоначально она была портирована с языка Java (библиотека JUnit) и написана на J#

Утилита NUnit

- Затем весь код был переписан на C# с использованием таких новшеств .NET, как атрибуты
- Существуют расширения оригинального пакета NUnit, большая часть из них также с открытым исходным кодом
- **NUnit.Forms** дополняет NUnit средствами тестирования элементов пользовательского интерфейса Windows Forms
- **NUnit.ASP** выполняет ту же задачу для элементов интерфейса в ASP.NET

Создание тестов для nUnit

- Для написания тестов можно использовать скриптовое расширение любого из .Net языков программирования
- Основными элементами таких расширений, используемых описания тестов, являются *утверждения (assertions)* и *директивы (directives)*
- Тесты оформляются как методы *тестирующего класса,* который снабжается ссылкой на *тестируемый класс*

Утверждения

- Утверждения представляют собой гипотезы, высказываемые тестировщиком относительно результатов выполнения того или иного теста
- Если гипотеза подтвердилась, то начинает выполняться следующий тест (либо тестирование завершается), иначе возникает ошибка

Примеры утверждений

- Все утверждения являются статическими методами класса Assert и, обычно, содержит два параметра – ожидаемый результат и действительный:

```
Assert.AssMethod(expected, actual);
```

- Примеры утверждений:

```
Assert.AreEqual(expected, fMB1.Subtract(fMB2));
Assert.IsTrue(fMB1.Multiply(0).IsZero);
Assert.Greater( x, y );
StringAssert.IsMatch( “Hello!”, MyStr );
```

Параметры утверждений

- Однако для каждого метода существуют перегружаемые варианты, которые содержат дополнительные параметры, позволяющие сформировать строку сообщения

Параметры утверждений

- Дополнительный параметр может быть обычной строкой, либо строкой со списком параметров, добавляемых в сообщение о результатах выполнения теста:

  Assert.AreEqual(int expected, int actual, string message);

  Assert.AreEqual(int expected, int actual, string message, params object[] parms);

Одно утверждение на тест

- Рекомендуется на каждый тест делать только одно утверждение, поскольку при возникновении ошибки в каком-либо из утверждений выполнение данного теста завершается и все последующие утверждения не проверяются

Две модели для утверждений

- В nUnit поддерживаются две модели для утверждений – *классическая* и *закрытая*
- Классическая модель предполагает непосредственное обращение к методам класса Assert так, как это было сделано в вышеприведенных примерах

Закрытая модель

- В закрытой модели (constraint-based model) используется единственный метод класса Assert – метод That
- Этот метод возвращает объект, в котором реализована вся логика, необходимая для проверки утверждения

```
Assert.That( myString, Is.EqualTo("Hello") );
```

Закрытая модель

- При таком вызове создается объект EqualConstraint, реализующий необходимую логику, поэтому вышеприведенный пример можно переписать в виде:

```
Assert.That( myString, new EqualConstraint("Hello"));
```

Основные виды утверждений

- Все утверждения nUnit можно разделить на несколько групп:
  - утверждения равенства (Equality Asserts)
  - утверждения сравнения (Comparison Asserts)
  - утверждения о типах (Type Asserts)
  - утверждения о строках (StringAssert)

Утверждения равенства

- Осуществляют проверку равенства значений двух своих аргументов
- Два основных метода AreEqual и AreNotEqual реализованы для разных типов данных
- При несовпадении типов осуществляется корректное приведение к необходимому типу

Утверждения равенства

- При сравнении вещественных значений в качестве третьего аргумента задается требуемая точность:

  Assert.AreEqual(float expected, float actual, float tolerance);

- Допускается сравнение массивов и коллекций: два массива считаются равными, если равны их размеры и совпадают значения соответствующих элементов

Утверждения сравнения

- Осуществляют сравнение двух величин
- Основные методы:

```
Assert.Greater( int arg1, int arg2 );
Assert.GreaterOrEqual( int arg1, int arg2 );
Assert.Less( int arg1, int arg2 );
Assert.LessOrEqual ( int arg1, int arg2 );
```

- Подобные методы реализованы и для других типов аргументов

Утверждения о типах

- Позволяют проверить принадлежность объекта определенному типу
- Основные методы:

```
Assert.IsInstanceOfType( Type expected, object actual );
Assert.IsNotInstanceOfType( Type expected, object actual );
Assert.IsAssignableFrom( Type expected, object actual );
Assert.IsNotAssignableFrom( Type expected, object actual );
```

Утверждения о строках

- Основные методы:

StringAssert.Contains(string expected, string actual);

StringAssert.StartsWith(string expected, string actual);

StringAssert.EndsWith(string expected, string actual);

StringAssert.AreEqualIgnoringCase(string expected, string actual);

StringAssert.IsMatch(string expected, string actual);

Проверка условий

- Еще одна группа методов, использующих один аргумент служит для проверки различных условий:

```
Assert.IsTrue( bool condition );
Assert.IsFalse( bool condition);
Assert.IsNull( object anObject );
Assert.IsNotNull( object anObject );
Assert.IsEmpty( string aString );
Assert.IsNotEmpty( string aString );
```

Директивы

- *Директивы* или *атрибуты* – это специальные предложения, используемые для структурирования тестовых заданий и описания дополнительных спецификаций теста
- Все директивы содержатся в пространстве имен NUnit.Framework, которое должно быть включено в любой файл, содержащий тесты

Категории директив

- Существует пять категорий директив:
  - Идентифицирующие
  - Селектирующие
  - Модифицирующие
  - Подготовки и очистки
  - Параметризующие
- Далее приведены примеры для каждой из категорий директив (атрибутов)

Идентифицирующие атрибуты

- Класс, содержащий методы-тесты, должен быть снабжен атрибутом TestFixture

  [TestFixture] public class SuccessTests
- Методы-тесты такого класса должны иметь атрибут Test

  [Test] public void Add()
- Атрибут Description() позволяет давать краткие описания тестов

  [Test, Description («Предельные значения»)]

Селектирующие атрибуты

- Атрибут Ignore() позволяет пометить метод-тест как временно невыполняемый без необходимости удаления его из тестирующего класса

```
[Test]
[Ignore("Решить, как обрабатывать ошибку в транзакции")]
public void TransferFundsAtomicity()
```

Модифицирующие атрибуты

- Атрибут ExpectedException() используется для тестов, в которых проверяется возможность возникновения исключительных ситуаций при выполнении тестируемого метода

  ```
  [Test]
  [ExpectedException(typeof(InvalidOperationException))
  ]
  public void ExpectAnExceptionByType()
  ```

- Тест считается пройденным, если возникает ожидаемое исключение

Атрибуты подготовки-очистки

- При выполнении тестов важно, чтобы все данные и объекты, оставшиеся после предыдущих тестов, уничтожались, и для каждого нового теста воссоздавалось исходное состояние окружения
- Для этой цели используются атрибуты SetUp и TearDown
- Атрибутом SetUp помечается метод, обеспечивающий подготовку среды, например, создание экземпляра тестируемого класса

Атрибуты подготовки-очистки

- Атрибутом SetUp помечается метод, обеспечивающий подготовку среды, например, создание экземпляра тестируемого класса

  ```
  [SetUp]
  public void Setup()
  ```

- Как видно из этого примера атрибут SetUp может заменять атрибут Test
- Метод, помеченный атрибутом SetUp будет вызываться перед выполнением любого теста

Атрибуты подготовки-очистки

- Атрибутом TearDown помечается метод, который выполняет завершающие действия, после прогона любого теста

  [TearDown]
  public void Clean()
- Методы подготовки и очистки должны быть единственными в тестирующем классе
- Эти методы могут рассматриваться как конструктор и деструктор тестов

Параметризующие атрибуты

- Наиболее известным атрибутом этой категории является атрибут TestCase(), позволяющий задать набор аргументов для тестируемого метода

  [TestCase(12, 3, 4)] [TestCase(12, 2, 6)]

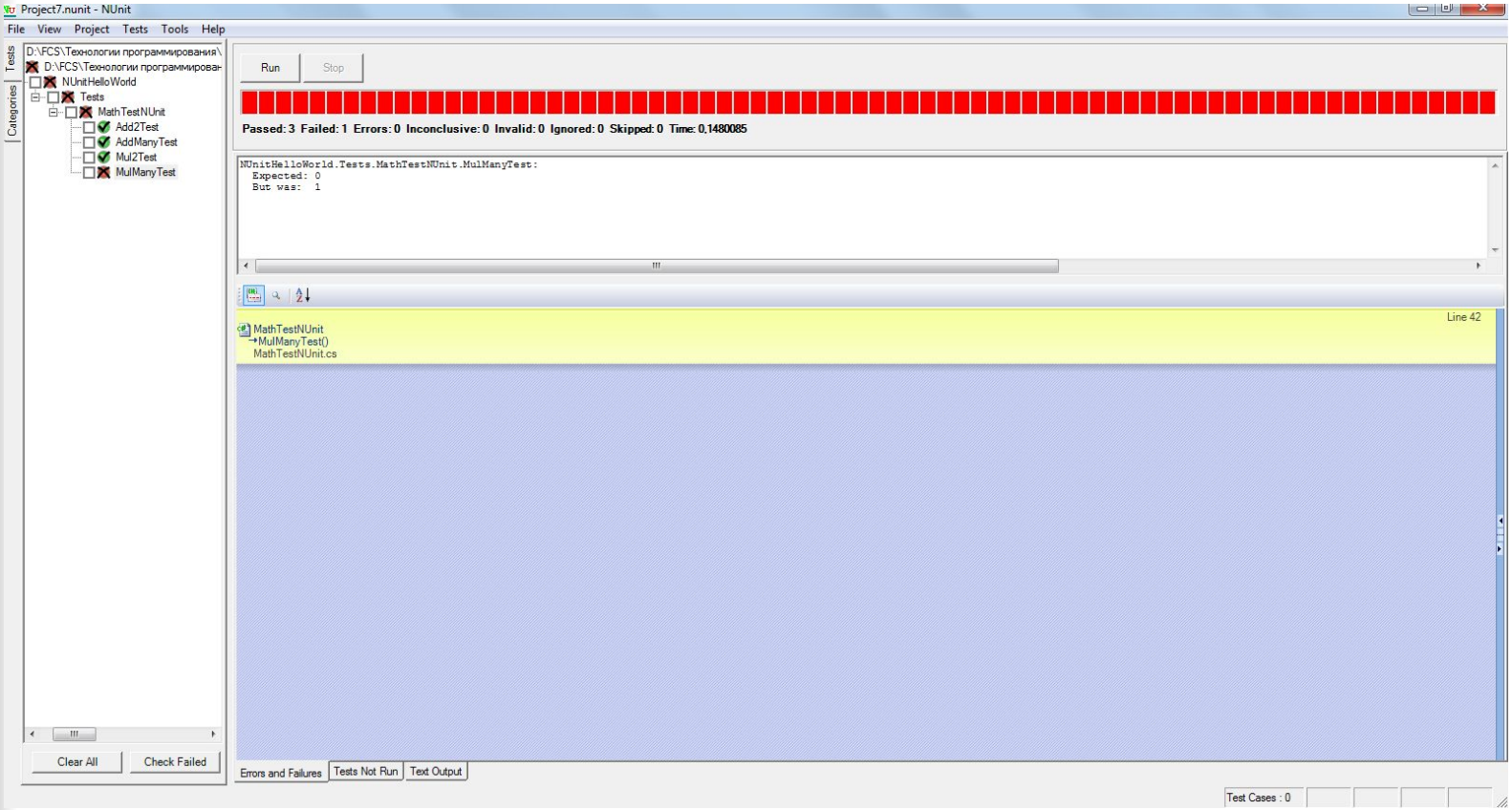
  [TestCase(12, 4, 3)]

  public void DivideTest(int n, int d, int q) { Assert.AreEqual(q, n / d); }
- Тест DivideTest() будет выполнен трижды с разными наборами параметров для тестируемого метода

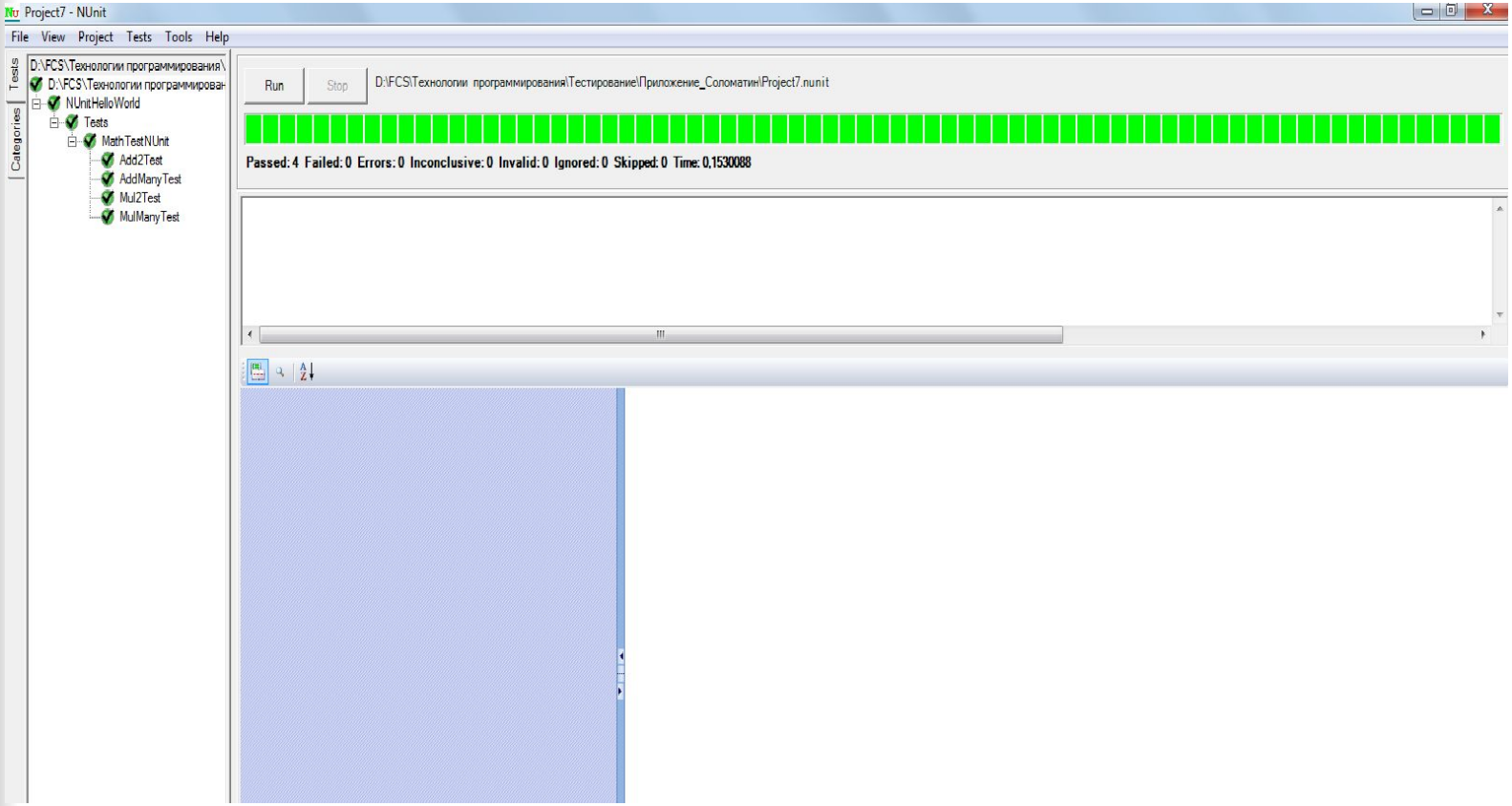
Примеры unit-тестирования

- Тестируемый класс
- Тестирующий класс
- Инструкция и примеры по работе с утилитой

Неудачное завершение



Удачное завершение



Средства тестирования от Microsoft

- Приложение Microsoft Test Manager (MTM), поставляемое вместе с Visual Studio, начиная с версии 2010
- Это приложение используется для работы над командными проектами и требует подключения к Team Foundation Server

Средства тестирования от Microsoft

- Утилита тестирования MSTest, выполняемая в режиме командной строки
- Исполняемый файл C:\Program Files\Microsoft Visual Studio 10.0\Common7\IDE\MSTest.exe

Средства тестирования от Microsoft

- Средства тестирования, интегрированные в среду Visual Studio
- Вызываются из пункта меню Тест
- Проект с примером модульного тестирования в инструментальной среде
  Проект с примером модульного тестирования в инструментальной среде Visual Studio

Конец лекции